From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Sunday, July 25, 2010 7:25 PM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: CubeHash

This note responds to two recent eprint papers that discuss CubeHash.

1. http://eprint.iacr.org/2010/262, "Cube test analysis of the statistical behavior of CubeHash and Skein," claims that a particular test detects "nonrandom behavior overall in both CubeHash and Skein."

I sent email to the author saying "I have a number of questions, but for the moment I'll ask merely the most obvious question: Do you have an example of a function where you tried the same test and it did _not_ declare 'nonrandom behavior'? In other words, do you have evidence that the problem is with CubeHash and Skein rather than with your test?"

The answer was no. Rumors state that the author subsequently tried other SHA-3 candidates and found that all of them flunked the same "test." The eprint paper has not been withdrawn, but it is nevertheless reasonably clear that this "test" is bogus; there is no evidence of any problem with CubeHash, Skein, or the other functions that have been "tested."

2. CubeHash is defined with two tunable parameters: CubeHashr/b applies r rounds of invertible transformation after every b bytes of input. The original CubeHash submission in October 2008 included implementations of CubeHash{1,2,4,8}/{1,2,4,8,16,32}, and proposed CubeHash8/1 for SHA-3.
An official tweak in July 2009 instead proposed

```
   CubeHash16/32-224 for SHA-3-224,
   CubeHash16/32-256 for SHA-3-256,
   CubeHash16/32-384 for SHA-3-384-normal,
   CubeHash16/32-512 for SHA-3-512-normal,
   CubeHash16/1-384 for SHA-3-384-formal, and
   CubeHash16/1-512 for SHA-3-512-formal.
```

The SUPERCOP benchmarking suite includes CubeHash{8,16}/{1,2,4,8,16,32}.

All of these proposals and implementations used powers of 2 for both r and b, but cryptanalysts are of course welcome to consider the strength of intermediate values of r and b. The paper "Inside the hypercube" by Aumasson, Brier, Meier, Naya-Plasencia, and Peyrin (November 2008 on eprint; subsequently ACISP 2009) outlined an attack that needed

```
   "256 equations" for "33<=b<65" (thus taking time roughly 2^256) or
   "384 equations" for "17<=b<33" or
   "448 equations" for "9<=b<17" or
   "480 equations" for "5<=b<9"
```

(see Section 3.4 of the paper) whereas the similar attack discussed in the original submission document achieves these complexities only for b=32, b=16, b=8, etc. One might guess from the hypercube structure that choosing a power of 2 for b---as every CubeHash proposal has done, both before and after the 2008 attack---achieves the best tradeoff between security and speed.

http://eprint.iacr.org/2010/262, "Symmetric states and their structure:
improved analysis of CubeHash" by Ferguson and Lucks and McKay, works out details of an attack that, as far as I can tell, is identical to the
2008 attack. The attack complexities stated in this paper, roughly 2^256 for b=33 and 2^384 for b=32, are not news; they already appeared in the

From: hash-forum@nist.gov on behalf of Paul Crowley [paul@lshift.net]
Sent: Monday, July 26, 2010 5:29 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On 26/07/10 07:58, Jean-Philippe Aumasson wrote:
> I tend to agree with this "common sense argument", to some extent.
> However NIST set those rules before the competition, and we (the
> submitters) all designed our algorithms to suit NIST's requirements.
> Were the security requirements limited to 256-bit, most of us would
> have designed completely different hashes, and I guess NIST would have
> received 90% of "narrow-pipe sponge-like" algorithms. It would be
> unfair to now change the rules of the game.

I greatly appreciate the work that algorithm submitters have put in to this competition,
but I would prefer that NIST treat fairness to submitters as a secondary consideration
after choosing the algorithm that will actually best meet present and future needs in the
real world.
--
   [][][] Paul Crowley
     [][] LShift Ltd
   []  [] www.lshift.net
```

From: hash-forum@nist.gov on behalf of Vlastimil Klima [v.klima@volny.cz]
Sent: Monday, July 26, 2010 6:45 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
> I greatly appreciate the work that algorithm submitters have put in to
> this competition, but I would prefer that NIST> treat fairness to
> submitters as a secondary consideration after choosing the algorithm
> that will actually best meet present and fu> needs in the real
> world.-- [][][] Paul Crowley [][] LShift Ltd []  [] www.lshift.net
```

Dear Paul,
it sounds "practically", but it is naive and in fact you are in a contradiction.

The best way how NIST can "choose the algorithm that will actually best meet present and future needs in the real world" is a competition.

After changing the input requirements
of the competition according to one candidate, nobody would know what "will actually best meet present and future needs in the real world", because there is the only one candidate in such a competition. To have more candidates, a new competition (with changed input reguirements) should be started.

So, fairness seems to be a necessary condition for any meaningful competition.

Vlastimil Klima

From: hash-forum@nist.gov on behalf of John Wilkinson [wilkjohn@gmail.com]
Sent: Wednesday, July 28, 2010 11:12 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
I understand both of the sentiments expressed below, however, I would urge NIST to
reconsider their requirements, or at least the weighting of them. This is not an argument
in favor of Cubehash or any other proposal, but I honestly don't know anyone who considers
preimage resistance greater than 2^256 to be practically important. I don't think that any
candidate algorithm should be disqualified, or even given a demerit, because it "only"
offers preimage resistance of, say, 2^300, *unless* it is felt that this "weakness" is
somehow indicative of an actual, *exploitable* weakness that may be discovered in the
future.

On Mon, Jul 26, 2010 at 6:44 AM, Vlastimil Klima <v.klima@volny.cz> wrote:
>
>> I greatly appreciate the work that algorithm submitters have put in
>> to this competition, but I would prefer that NIST> treat fairness to
>> submitters as a secondary consideration after choosing the algorithm
>> that will actually best meet present and fu> needs in the real
>> world.-- [][][] Paul Crowley [][] LShift Ltd []  [] www.lshift.net
>
> Dear Paul,
> it sounds "practically", but it is naive and in fact you are in a
> contradiction.
>
> The best way how NIST can "choose the algorithm that will actually
> best meet present and future needs in the real world" is a
> competition.
>
> After changing the input requirements
> of the competition according to one candidate, nobody would know what
> "will actually best meet present and future needs in the real world",
> because there is the only one candidate in such a competition. To have
> more candidates, a new competition (with changed input reguirements)
> should be started.
>
> So, fairness seems to be a necessary condition for any meaningful
> competition.
>
> Vlastimil Klima
>
>
>
>
```

From: hash-forum@nist.gov on behalf of Tim Broksma [timbroksma@gmail.com]
Sent: Wednesday, July 28, 2010 3:18 PM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: CubeHash

Hi all,

I am noticing two different issues discussed under the same topic.

1. Paul Crowley was talking about "choosing the algorithm that will actually best meet present and future needs in the real world."
2. John Wilkinson has replied that "he don't think that any candidate algorithm should be disqualified, or even given a demerit, because it "only" offers preimage resistance of, say, 2^300".

I agree with both of them, and I want to add the following:
Candidate algorithms that are offering "only" 2^300 of preimage resistance should be not disqualified because of that property but because of a property (if they have one) that they are not practical for some applications where hash functions are frequently used.

Tim Broksma

John Wilkinson wrote:

```
>
> I understand both of the sentiments expressed below, however, I would
> urge NIST to reconsider their requirements, or at least the weighting
> of them. This is not an argument in favor of Cubehash or any other
> proposal, but I honestly don't know anyone who considers preimage
> resistance greater than 2^256 to be practically important. I don't
> think that any candidate algorithm should be disqualified, or even
> given a demerit, because it "only" offers preimage resistance of, say,
> 2^300, *unless* it is felt that this "weakness" is somehow indicative
> of an actual, *exploitable* weakness that may be discovered in the
> future.
```

On Mon, Jul 26, 2010 at 6:44 AM, Vlastimil Klima <v.klima@volny.cz> wrote:

```
>
>> I greatly appreciate the work that algorithm submitters have put in
>> to this competition, but I would prefer that NIST> treat fairness to
>> submitters as a secondary consideration after choosing the algorithm
>> that will actually best meet present and fu> needs in the real
>> world.-- [][][] Paul Crowley [][] LShift Ltd []  [] www.lshift.net
>
> Dear Paul,
> it sounds "practically", but it is naive and in fact you are in a
> contradiction.
>
> The best way how NIST can "choose the algorithm that will actually
> best meet present and future needs in the real world" is a
> competition.
>
> After changing the input requirements
> of the competition according to one candidate, nobody would know what
> "will actually best meet present and future needs in the real world",
> because there is the only one candidate in such a competition. To have
> more candidates, a new competition (with changed input reguirements)
> should be started.
>
> So, fairness seems to be a necessary condition for any meaningful
> competition.
>
> Vlastimil Klima
>
```

From: hash-forum@nist.gov on behalf of Jonathan Day [jonathan.day.jobs@gmail.com]

Sent: Wednesday, July 28, 2010 8:22 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash

I've suggested different levels of award before. Last time, I suggested an overall winner with an "approved for restricted use"/runner-up award for the best algorithm once performance is not taken into consideration.

I am uncertain here whether to continue just advocating those two categories, or whether it would be useful to have another "approved for restricted use"/runner-up award. My idea remains that there are a lot of users who cannot use non-NIST-approved algorithms, but where security/robustness is of overwhelming importance and speed is not.

As such, I certainly feel that there is a very legitimate need for NIST-recognition for the strongest of the cryptographic hashes, regardless of performance. It is not a case of whether such a thing will get used, but whether such use will be authorized.

But the strongest hash overall may not actually be the strongest in all areas. As hard as it will be to convince NIST that one award is not enough, convincing them to give out two extra may be asking the impossible. Assuming it is not, would preimage resistance be a legitimate area in which to fast-talk NIST into creating a special runner-up category for? Or, given that they absolutely aren't going to countenance more than they absolutely have to for recognition, are there any areas of greater importance that aren't simply covered by the "strongest hash overall" category that we should consider pressuring them on?

On Wed, Jul 28, 2010 at 8:10 AM, John Wilkinson <wilkjohn@gmail.com> wrote:

> I understand both of the sentiments expressed below, however, I would
> urge NIST to reconsider their requirements, or at least the weighting
> of them. This is not an argument in favor of Cubehash or any other
> proposal, but I honestly don't know anyone who considers preimage
> resistance greater than 2^256 to be practically important. I don't
> think that any candidate algorithm should be disqualified, or even
> given a demerit, because it "only" offers preimage resistance of, say,
> 2^300, *unless* it is felt that this "weakness" is somehow indicative
> of an actual, *exploitable* weakness that may be discovered in the
> future.

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Wednesday, August 04, 2010 11:20 PM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: CubeHash

Thomas Pornin's paper "Comparative performance review of the SHA-3 second-round candidates" (dated 21 June 2010) reports slow-sounding performance figures for the "sphlib" implementation of CubeHash on four CPUs: Intel Q6600, PowerPC 750, Broadcom BCM3302, and ARM920T.
The paper also says, in its "What Makes a Hash Function Fast" section, that CubeHash suffers from "register starvation."

The point of this "OFFICIAL COMMENT" is that the "register starvation" documented by Pornin is actually a mistake in the sphlib implementation.
I've now posted software that avoids this mistake and that's much faster than the sphlib implementation on a very wide range of CPUs. See below for technical details.

Any user who cares about speed will take the faster software, so the resulting speed of CubeHash will be the speed of the faster software.
The slowness of the sphlib implementation might be interesting in a programming-techniques competition but is of no relevance to a hash-function-speed competition.

Of course, a hash-function competition is much more than a hash-function-speed competition: I think that CubeHash's large security margin should be taken into account along with its speed. But I don't think that the existence of other hash-function criteria justifies sloppy treatment of the efficiency criterion.

I should also comment that, even though this note focuses on CubeHash, I've peeked at the sphlib implementations of some other SHA-3 candidates and find myself quite unimpressed with the level of optimization for small CPUs. The actual speeds of SHA-3 candidates on these CPUs will often be much faster than the sphlib speeds---and the speedup factors will vary heavily from one candidate to another, undermining any predictive value of Pornin's reports. (This is also true for large CPUs, as has been discussed previously on hash-forum.)

Now the technical details.

For the Q6600, the sphlib implementation of CubeHash was already known to be highly suboptimal. The eBASH benchmarks (utrecht, 20100802) show
CubeHash16/32 running at 13.24 cycles/byte in both 32-bit mode and 64-bit mode. The sphlib implementation runs between three and five times
slower: 39.87 cycles/byte in 64-bit mode and 64.19 cycles/byte in 32-bit mode.

The Q6600, like other modern CPU designs (including several years of ARM designs used in phones, PowerPCs starting with the G4, etc.), has fast 128-bit vector instructions. All of the basic CubeHash operations are 128-bit vectorizable; they run quickly on the Q6600 when this vectorizability isn't hidden by the implementation. (Future Intel CPUs will provide 256-bit vector operations, making CubeHash even faster.)

The story is different for the other CPUs mentioned in Pornin's paper:
older CPUs without vector instructions. This is where the "register starvation" issue shows up. There are only 14 easily usable 32-bit registers on old ARMs, for example---certainly not enough for the 1024-bit CubeHash state.

Most SHA-3 candidates have much larger states, but Pornin singles out CubeHash as allegedly having an unusually large penalty in copying data back and forth between registers and cache. He claims that frequent spills from registers to cache are a consequence of CubeHash's parallelizability, forcing a tradeoff between vector speed and spill minimization.

In fact, CubeHash's parallelizability provides tremendous extra flexibility in the order of operations. When the number of registers is limited, the implementor should of course load several state words that need to talk to each other, and operate on those for as long as possible before moving on to other state words. (Some keywords for this standard optimization in various contexts: "locality of reference"; the "four-step FFT"; "blocking" matrix multiplication.)

The sphlib implementation ignores this flexibility. It sweeps through the entire state in the most straightforward order, rather than in an order that brings related operations together. For example, the lines

```
   xg = T32(x0 + xg); \
   x0 = ROTL32(x0, 7); \
   ...
   xo = T32(x8 + xo); \
   x8 = ROTL32(x8, 7); \
   ...
   x8 ^= xg; \
   ...
   x0 ^= xo; \
```

are spread very far apart in the sphlib code.

My ARM implementation of CubeHash uses just 103 loads, 87 stores, and 129 arithmetic instructions for its inner loop (2 rounds). The sphlib implementation uses 358 loads, 210 stores, and 270 arithmetic instructions for its inner loop (4 rounds). The load/store overhead is thus 1.5x larger in sphlib. The sphlib implementation suffers from another scheduling problem on top of this: it frequently uses values almost immediately after loading those values, stalling the pipeline on typical ARMs.

The bottom line is that, when I tried both implementations on a Nokia N810 with an OMAP2420 (ARM1136) CPU, I found my own implementation to be more than twice as fast. My implementation also fits into just 1996 bytes of code for the complete crypto_hash() function.

I realize that I'm changing CPUs here---Pornin benchmarked an ARM920T inside a "Hewlett-Packard HP-50g scientific calculator"---but I think it's safe to predict that the extra load/store overhead in sphlib is also a major contributor to sphlib's slowness on the ARM920T, the MIPS that Pornin selected, and other small CPUs. I also think that everyone will agree that cryptographic speed on a Nokia N810 is of much more real-world interest than cryptographic speed on an HP-50g calculator.

Let me close wth a quick advertisement: The AppliedCrypto 2010 program, http://2010.applied.cr.yp.to, now includes a "Writing really fast code" lab session from 09:15 to 10:15 on Monday the 16th outside the Crypto lecture room at UCSB. Registration will be free.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Thursday, August 05, 2010 11:53 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Wed, Aug 04, 2010 at 11:19:49PM -0400, D. J. Bernstein wrote:
> I realize that I'm changing CPUs here

Well, you could have asked. My email works. I can run code on my systems.

Nevertheless, I have tried your code on my test systems. You just tell that you "have posted software" but without any indication about where, precisely, said software was posted, so I am taking a wild guess and assume that you mean that you have included your new implementations in SUPERCOP, and that the latest version, which I found there:
   http://hyperelliptic.org/ebats/supercop-20100803.tar.bz2
contains the code you are alluding to. Indeed, that archive contains assembly code for CubeHash16/32 for several architectures (including 32-bit PowerPC, 32-bit MIPS, and ARM).

This yields this (in MBytes/s for an 8 kB buffer):

```
                              sphlib    SUPERCOP

PowerPC 750                    5.81       8.69
Broadcom BCM3302 (MIPS)        1.11       1.57
ARM920T                        0.63       0.78
```

Hence, your "more than twice as fast" on ARM turns out to be something like "24% faster" which, while not negligible, is somewhat less impressive. Also, let's see on a more powerful ARM, a 1.2 GHz Feroceon 88FR131 (kirkwood), an ARMv5 core from Marvell. This yields:

```
ARM Feroceon 88FR131           7.17      11.07
```

again nothing near "more than twice as fast".

Now let's try to see what happens here. sphlib is in C. This means that the order of operations in the source code is, supposedly, _irrelevant_.
Of course, at the assembly level, operations should be ordered so as to stall the processor the least, taking into account dependencies. But that is exactly the point of the C compiler. My job, as a programmer, is to lay out the operations in plain view of the compiler, so that the compiler may see by itself which operation depends on which operands, and computes the optimal ordering. Hash functions tend to have no data-dependent path (at the block level), so simply writing all operations in specification order gives to the compiler all the information that it needs.

Now it so happens that the GCC instruction scheduler and register allocator is, as they say, suboptimal. They know that, see for
instance:
   http://gcc.gnu.org/wiki/RegisterAllocation
and you know that too, since that's one of the reasons qhasm actually
exists: you wanted a tool which allowed you to better optimize register allocation than what GCC does by itself.

So the figures above, with qhasm-optimized output 24 to 49% faster than the production of GCC, are simply a measure of how qhasm (helped with the programmer) does a better job than GCC. It does not point at any mistake in sphlib code, because there is very little that can be done about it with C code (indeed, the C dialect implemented by GCC does not allow the level of fine tuning that qhasm allows).

What does that tell us about the hash functions in general, and sphlib in particular ? Remember that sphlib is a tool to _compare_ hash functions with each other. That's written right in the article title, which begins by "Comparative Performance Review". Using GCC for comparing functions with each other works as long as any slowdown implied by GCC suboptimal register allocation is similar between the compared functions -- an assumption which the measures above tend to corroborate.

Also:

> I also think that everyone will agree that cryptographic speed on a
> Nokia N810 is of much more real-world interest than cryptographic
> speed on an HP-50g calculator.

this is not true, since I would, at least, not agree. This is because this seemingly obvious pearl of wisdom:

> Any user who cares about speed will take the faster software

is generally wrong on embedded systems. In that world, things usually run that way:

  Vendors will take the cheapest hardware which meets the required
  functionalities (including performande goals)

and software choice is not orthogonal. You cannot expect the "typical" embedded system to use the most recent cores.

In the desktop and server worlds, you can expect technological advances to percolate at a fast pace. This is a natural byproduct of an open, generic platform: since the set of possible applications is not bounded, any new gizmo such as, e.g., a SSE2 unit, is likely to improve performance for at least a few applications, giving an edge to the hardware vendor who integrates that new feature. Competition between hardware vendors ensures that they will avidly look for such "edges". In the other half of the feedback loop, application developers (who also compete with each other) will jump on the new features, on the idea that competition between hardware vendors will just make any new feature common enough in the near future. That's how, in the PC world, every x86 now has a FPU, MMX support, SSE, SSE2... and general 64-bit support is soon to be achieved; and yet they still all support 16-bit "real mode".
The features just accumulate. And hash functions are fast on open platforms, because the "hardware arms race" tends to make the CPU oversized with regards to its I/O bandwidth (when you hash, you hash _something_, some data which has to come in or out of the system).

In the world of embedded systems, platforms are not open, and the list of applications which are to run on a given system is small and closed, and all under the control of the hardware vendor. When a vendor designs a router, he tries to find the cheapest combination of hardware and software for the functionalities he intends to offer. Hardware price tends to be more important here, because when you build 50 million units, you pay for software once, and for hardware 50 million times.
This is how many existing hardware systems, right now, happen to perform cryptography and yet lack many "common-place" features (e.g. a FPU, let alone a SIMD unit).

A Nokia N810 is an open system (hey, it's a Debian/Linux) which may run almost all open-source applications that compile on Linux. Conversely, a HP50g system is a closed platform. The HP50g more closely emulates embedded systems and what can be expected in the "embedded world", which is why I consider that performance of hash functions on that HP50g tells us more useful information, economically speaking, that what you get on a N810.

On a related note, smartphones (e.g. the iPhone and all those running
Android) are open systems with regards to application availability (Apple boasts about 200,000 apps) and they illustrate the gap that open platforms have between computing power and I/O bandwidth: under ideal conditions and a 3.5G network, you may dream about 21

MBit/s download speed -- that's less than 3 MBytes/s, and that's a dream. But the typical Android-aware smartphone has a 1 GHz ARM processor, which can easily hash more than twice faster than that with most of the the second-round SHA-3 candidates -- even without using the SIMD units that such processors tend to have, thanks to the dynamics I described above.
With the "fast" SHA-3 candidates, hash function speed becomes a complete non-issue on smartphones.

In brief, no hash function will make a smartphone more expensive, or become a bottleneck in an application; whereas supporting a not-that-fast hash function as part of, e.g., an IPsec implementation in a cheap router may make said router less cheap.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of Danilo Gligoroski [danilo.gligoroski@gmail.com]
Sent: Thursday, August 05, 2010 2:58 PM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: CubeHash

Dear Thomas,

> When a vendor designs a router, he tries to find the cheapest
> combination of hardware and software for the functionalities he
> intends to offer.

I see that you like to give the example of the vendor who is building cheap router in the context of defending the development of portable C library sphlib and measuring the performance of that library on various platforms including ones that will be used in embedded systems.

Not going again into the debate that many of those functions (including Blue Midnight Wish) will be implemented in final products in much efficient way that sphlib is offering, I want to underline that the example about the router vendor is not corresponding with the claim that you have put on page
6 of your report about sphlib: <Quote> ... but sphlib is not (yet) the right tool to evaluate hash function performance on very short messages. </Quote>

Namely, the hash functions used in routers are used for "Route Authentication" and are hashing very short messages, typically less than 64 bytes and in HMAC setup (take any system administrator manual for Cisco routers and you will see how long are messages that are hashed in their routers).

I completely understand why you as a part of Shabal team in this moment is not emphasizing the speed of hash functions on relatively short messages, but then again the example of the router vendor cannot be used as an argument for the way how sphlib is implementing and measuring hash functions.

Regards,
Danilo!

From:	hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent:	Thursday, August 05, 2010 3:45 PM
To:	Multiple recipients of list
Subject:	Re: OFFICIAL COMMENT: CubeHash

Thomas Pornin wrote on 05 Aug 2010 17:53:17 +0200:

> On a related note, smartphones (e.g. the iPhone and all those running
> Android) are open systems with regards to application availability
> (Apple boasts about 200,000 apps) and they illustrate the gap that
> open platforms have between computing power and I/O bandwidth: under
> ideal conditions and a 3.5G network, you may dream about 21 MBit/s
> download speed -- that's less than 3 MBytes/s, and that's a dream. But
> the typical Android-aware smartphone has a 1 GHz ARM processor, which
> can easily hash more than twice faster than that with most of the the
> second-round SHA-3 candidates -- even without using the SIMD units
> that such processors tend to have, thanks to the dynamics I described above.
> With the "fast" SHA-3 candidates, hash function speed becomes a
> complete non-issue on smartphones.

Under ideal conditions, a smartphone is connected to a WiFi network, and if the device supports 802.11n (that's the case of the iPhone 4 according to Wikipedia), you should get a much better rate (I'm a bit confused about the rates offered by 802.11n, but the theoretical value seems to be 150Mbit/s). So hashing speed might very well be an issue.

--
Gaëtan Leurent

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Thursday, August 05, 2010 3:45 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

D. J. Bernstein wrote on 26 Jul 2010 01:25:18 +0200:

```
>    The "SHA3512formal" proposal is aimed at users who are (1)
>    concerned with attacks using 2^384 operations, (2) unconcerned with
>    quantum attacks that cost far less, and (3) unaware that attackers
>    able to carry out 2^256 operations would wreak havoc on the entire
>    SHA3 landscape, forcing SHA3 to be replaced no matter which
>    function is selected as SHA3. The "SHA3512normal" proposal is
>    aimed at sensible users. For all real-world cryptographic
>    applications, the "formal" versions here can be ignored, and the
>    tweak amounts to a proposal of CubeHash16/32 as SHA3.
```

To be honest, I'm not surprised that people misread this, and I also thought that the expected security of CubeHash-512-normal was 2^384.

Could you give a clear statement of the expected strength of CubeHash?
(The document strength.pdf claims 2^512, but it doesn't really make sense...)

```
> I realize that NIST originally asked for 384-bit hashes to provide
> 2^384 preimage security, and for 512-bit hashes to provide 2^512
> preimage security. The "formal" CubeHash proposals are designed to
> comply with these requests, taking b=1 for 384-bit and 512-bit output,
> but I hope that continued discussion can reach consensus that these
> requests should be dropped.
```

Agreed, I don't think we really need a 512 bit hash with 512 bit of security. However, NIST asked for one, and we all tried to build one.
Even the first-round version of CubeHash aimed for 512 security... It wouldn't be fair to make such a significant change to the rules.
Moreover, it wouldn't yield the best possible hash function, because most design were not build for those weaker requirements, and several functions have been eliminated because of attacks in the 2^256-2^512 range (eg. AURORA, Twister).

It should also be noted that when considering long-term security of signatures, the birthday attack is only a concern if the hash function is still used to produce new signatures when the attack becomes practical. If the function has been phased out, old signatures produced with a 512-bit hash are safe until 2^512 becomes practical. That's at least one scenario where it makes sense to have preimage resistance higher than collision resistance.

--
Gaëtan Leurent

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Thursday, August 05, 2010 4:16 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Thu, Aug 05, 2010 at 02:57:57PM -0400, Danilo Gligoroski wrote:
> Namely, the hash functions used in routers are used for "Route
> Authentication" and are hashing very short messages, typically less
> than 64 bytes and in HMAC setup (take any system administrator manual
> for Cisco routers and you will see how long are messages that are
> hashed in their routers).

Actually I am always thinking about IPsec.

The route authentication messages are not only short, they are also not very common; the overwhelming majority of what a router receives and sends does not consist of route authentication messages. Just about any hash function would be appropriate (performance-wise; I am not talking about security here) for these messages, and it would not imply any kind of measurable bottleneck.

With IPsec, however, there are two modes, called AH and ESP, which can be applied to incoming and outgoing IP packets. AH does only integrity checks, while ESP does encryption and integrity checks. The one integrity check algorithm that IPsec makes mandatory is HMAC-SHA1-96 (that's HMAC with SHA-1, and truncating the HMAC result to 96 bits).

One could argue that HMAC is not the best choice for a MAC (especially with ESP, where the MAC could be combined with the encryption), but IPsec is defined with HMAC and it seems plausible that SHA-3 will be used with HMAC in that context.

Anyway, with IPsec, HMAC is applied on all data packets, and most of them (those which actually consume bandwidth) have the size of an ethernet frame; accounting for the IP headers, there are still a bit more than 1400 input bytes per HMAC invocation. 1400 bytes is large enough to be considered a "non-short message".

(On gigabit ethernet, so-called "jumbo frames" are often encountered and reach 9000 bytes, but we are not talking about a _cheap_ router
anymore.)

One could make a similar case about SSL/TLS, which also uses HMAC (or a close ancestor thereof) on "records" with a typical plaintext length of
16 kB (that's the maximum length allowed by the TLS specification). But it would be less convincing since TLS typically applies encryption _and_ authentication, and the encryption cost dwarfs out the HMAC cost.

> I completely understand why you as a part of Shabal team in this
> moment is not emphasizing the speed of hash functions on relatively
> short messages,

When I talk about sphlib I am not part of the Shabal team; I am just "myself". As one of the Shabal designers, I actually expect Shabal to come out as "not that bad" in a short-message contest (Shabal has an average overhead of about three and a half 512-bit blocks, which is more than SHA-2 and, for instance, Blue Midnight Wish, but it is not awfully more either; and Shabal has a fast compression function).

The one reason I am not emphasizing speed on short messages is that I have not optimized code in sphlib specifically for that. And I have not done such specific optimization because I do not know how to measure speed on short messages.

Let me explain: when we talk about speed, we are actually interested in estimating the impact of using a given hash function within an "application". It of course depends on the application, and that's cumbersome, hence we use the "benchmark approximation". In a benchmark, we run the hash function "alone", with full usage of the CPU resources, in particular L1 cache (both instruction and data), and all the other cache-like elements in the CPU (branch prediction, stack frame caching...).

The benchmark approximation is quite good with long messages, thanks to buffering. In an application, the hash function is not alone in the data path, but if messages are long, then we can normally arrange for data to be processed by big chunks (the traditional buffer size being 8 kB).
When the hash function implementation is invoked, it will have to take over the caches; the first block will have an increased processing time, but after a few blocks the function reaches its top speed, up to the end of the buffer. This is what makes the benchmark a reasonably accurate prediction of how a given function would fare in a real-life application.

This approximation breaks down when input messages are really short.
With a short message, the hash function has finished its work without even reaching the point where it runs in "benchmark conditions", with all caches populated with the hash function implementation. For the benchmark to be accurate, it must be so that the complete code on the critical application data path (not just the hash function implementation, but also the rest of the code which handles the data to be hashed and the hash result) fits in the caches. I do not know how to measure that in a meaningful benchmark. The only qualitative conclusion I can infer is that application performance when hashing small messages will favour compact hash function implementations. I find it likely that in such a situation, compactness of the function code would be more important than actual cycle count for hash function execution. If I put on my Shabal-designer-hat again, I could point out that Shabal is amenable to very compact implementations. CubeHash is even better here; it is probably the function which allows for the shortest code (among the second-round SHA-3 candidates).

To be complete, I can think of _one_ application where benchmarking of short message hashing is accurate, and that's brute-forceing hashed passwords, because you can make a tight loop which performs many short message hashes in a row, and very little else. However, "BMW should be selected for SHA-3 because it allows for faster password cracking" does not strike me as the best selling slogan ever.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Thursday, August 05, 2010 4:34 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Thu, Aug 05, 2010 at 03:44:15PM -0400, Gaëtan Leurent wrote:
> Under ideal conditions, a smartphone is connected to a WiFi network,
> and if the device supports 802.11n (that's the case of the iPhone 4
> according to Wikipedia), you should get a much better rate (I'm a bit
> confused about the rates offered by 802.11n, but the theoretical value
> seems to be 150Mbit/s). So hashing speed might very well be an issue.

Apparently, 802.11n, as specified since 2009, allows for a maximum of 600 Mbit/s, but only with some very specific coupling of antennas and multiplexing, which is unlikely to be implemented in any actual device (and the relationshp of the iPhone 4 with antennas appears to be a bit strained). Many routers, access points and interfaces currently marketed under the "802.11n" name multiplex data over two 54 Mbit/s channels, so a realistic rate is 108 Mbit/s. Although I am not completely convinced that an iPhone can do that.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of Zooko O'Whielacronx [zooko@zooko.com]
Sent: Thursday, August 05, 2010 5:53 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

Dear Thomas Pornin:

I like your observations about the real world situation with regard to economics and engineering of embedded systems. I cautiously agree with most of what you wrote in this letter to which I am replying. There is one detail that I wanted to call out, however:

On Thu, Aug 5, 2010 at 9:53 AM, Thomas Pornin <thomas.pornin@cryptolog.com> wrote:
>
> under ideal
> conditions and a 3.5G network, you may dream about 21 MBit/s download
> speed -- that's less than 3 MBytes/s, and that's a dream. But the
> typical Android-aware smartphone has a 1 GHz ARM processor, which can
> easily hash more than twice faster than that with most of the the
> second-round SHA-3 candidates -- even without using the SIMD units
> that such processors tend to have, thanks to the dynamics I described above.
> With the "fast" SHA-3 candidates, hash function speed becomes a
> complete non-issue on smartphones.
>
> In brief, no hash function will make a smartphone more expensive, or
> become a bottleneck in an application;

This argument is not sufficient to prove your conclusion because it takes into account only time and not power or other applications.

Regarding power, suppose that you can get 3 MB/s download speed with your smartphone and that it has a 1 GHz ARM processor. Therefore whether the hash function that is uses can be computed at 5 MB/s or 10 MB/s will make no difference to the time to hash the data while downloading it. But will the 5 MB/s hash function waste more power, draining the battery faster? Or is that not a significant issue in practice? How much power does it cost to hash the data after you've downloaded it over your radio?

Regarding other applications, suppose that with the same setup you want to download the data, hash it, and play it out as a video with software video decoding (that is what my Tahoe-LAFS distributed storage system does when the file in question is a video and the client is a video player). Now if you are using a hash function which can run at 5 MB/s using all 1 GHz of the processor, this suggests that it will take up about 3/5 of the processor in order to hash the data at 3 MB/s, leaving 2/5 of the processor available to compute the video decoding. By comparison the hash function which can run at 10 MB/s with the full processor presumably can hash 3 MB/s using about 3/10 of the processor, allowing 7/10 of the processor to be used for other purposes. If this is right, then that means the faster hash function allows simultaneous applications (for example software video decoders) which require up to 7/10 of a 1 GHz ARM, where the slower hash function allows only those that take up to 2/5.

Again I want to emphasize that downloading bulk data, hashing it, and incrementally streaming it out to a consumer such as a video decoder is currently done in my software. And my software is currently used on ARM CPUs. (Although as far as I know it has never yet been used with a video decoder on an ARM CPU.)

Regards,

Zooko

From: hash-forum@nist.gov on behalf of Danilo Gligoroski [danilo.gligoroski@gmail.com]
Sent: Friday, August 06, 2010 12:02 PM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: CubeHash

Thomas Pornin wrote:
> To be complete, I can think of _one_ application where benchmarking of
> short message hashing is accurate, and that's brute-forceing hashed
> passwords, because you can make a tight loop which performs many short
> message hashes in a row, and very little else. However, "BMW should be
> selected for SHA-3 because it allows for faster password cracking"
> does not strike me as the best selling slogan ever.

No need to be cynical about the BMW speed on short messages, and no need to put words (or slogans) in my mouth.
If you are worried that the speed of SHA-3 would be too fast for password cracking, NIST has already a solution:
"SP 800-132, DRAFT Recommendation for Password-Based Key Derivation - Part 1: Storage Applications".

On the other hand try to play with a secure web server that is using its pool of randomness for producing thousands of random numbers and is serving thousands clients simultaneously that are making thousands of secure transactions from different protocols like: PKI, IPsec, TLS, S/MIME ....

All that activity will need cryptographically strong random numbers (produced by algorithms as Fortuna or similar algorithms and standards for generating cryptographically strong numbers).
And guess what: There, hash functions are used, and those hash functions are hashing short messages.

Best regards,
Danilo!

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Friday, August 06, 2010 2:20 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Fri, Aug 06, 2010 at 12:01:24PM -0400, Danilo Gligoroski wrote:
> On the other hand try to play with a secure web server that is using
> its pool of randomness for producing thousands of random numbers and
> is serving thousands clients simultaneously that are making thousands
> of secure transactions from different protocols like: PKI, IPsec, TLS,
> S/MIME ....

"Thousands" is hardly a worry. "Millions per second" would raise performance issues, but mere thousands are a piece of cake with any of the proposed hash functions.

TLS uses a hash-based deterministic PRF to derive the symmetric encryption keys and IV. TLS 1.0 and 1.1 use HMAC/MD5 and HMAC/SHA-1 (both simultaneously) while TLS 1.2 uses HMAC/SHA-256. Details are rather convoluted, but if you look closely you will find that all hash function invocations (in TLS 1.2) call the hash function with an input of length no less than 96 bytes, and half of them have inputs of at least 128 bytes. That's short, but not very short. More importantly, the PRF is used only during the handshake. Its execution time is negligible with regards to other handshake-related activity (in particular the asymmetric key exchange, even with the help of hardware accelerators); also, the ensuing symmetric cryptography (for the data to be tunnelled:
symmetric encryption, and integrity checks, then again with HMAC, but on large blocks) will be vastly more expensive. You would be hard pressed to detect any performance variation on a TLS server based on the hash function used in the PRF.

IPsec is similar. Random numbers are used as part of the asymmetric key exchanges, but this does not occur often, if only because such key exchanges require several round trips. When using the ESP protection with a block cipher in CBC mode (specified in RFC 2451), you need one random IV per packet, but RFC 2451 includes the following helpful text:

```
   The IV field MUST be same size as the block size of the cipher
   algorithm being used.  The IV MUST be chosen at random.  Common
   practice is to use random data for the first IV and the last block of
   encrypted data from an encryption process as the IV for the next
   encryption process.
```

which means that there will be very few PRNG invocations after all.
There again, no performance bottleneck with the hash function operating on short messages (there _may_ be a performance problem with the integrity checks -- with HMAC in IPsec -- but that's on longer messages, see my previous email).

S/MIME is the application of MIME encoding rules to CMS (RFC 5652), a
PKCS#7 derivative which allows for encryption and signature of arbitrary binary objects. A few random numbers are needed in some parts (asymmetric key exchange, signature with a randomized padding, IV for symmetric encryption...) but they will result in very few hash function invocations per object, and the other object handling costs will be much higher. The size of the needed random data does not grow up with the input data size.

"PKI" is not a protocol; it means "Public Key Infrastructure" which itself relies on a horde of various protocols. For X.509 PKI, most of those protocols are a bit "higher level" and do not bother with describing PRNG, only stating things like "cryptographically secure random source". The CMP protocol (RFC 4210) relies on CRMF (RFC 4211) which itself delegates all the cryptographic work to CMS. As described above, this does not entail any hash function usage where performance on short messages is detectable, let alone critical.

> All that activity will need cryptographically strong random numbers
> (produced by algorithms as Fortuna or similar algorithms and standards
> for generating cryptographically strong numbers).

Fortuna is not a good example here. The bulk work of random number generation in Fortuna is done with a block cipher in CTR mode (usually the AES). A hash function is used only when "reseeding"; when Fortuna is used to generate a lot of random data (a requirement for Fortuna performance to actually have any kind of relevance), reseeding occurs once every megabyte of random, and entails N+1 hash function invocations, where N is the number of "entropy pools" (usually 32). Each hash function invocation processes one pool, except the last, which hashes together the N previous hash function outputs. These hash function invocations may have a non trivial cost with regards to the AES-CTR 1 MB block only if they operate on quite long pools, with at least dozens of kilobytes of data per pool; there are then not "short messages".

So no, there are no hashes-of-short-messages worth speaking of in Fortuna, performance wise. There _may_ be hashing performance issues (if the AES part is very fast, e.g. with the AES-NI opcodes on the newer Intel processors) but such issues may arise only if the entropy pools are big, in which case the hash function processes long messages, not short messages.

A much better example is Hash_DRBG as specified in NIST Special Publication 800-90 (March 2007). With SHA-256, that PRNG will use one
SHA-256 invocation per 256 bits to produce, and that invocation is over a 440-bit input message (there are also other invocation for (re)seeding, but these are rare and do not impact overall performance).

I could imagine some applications using Hash_DRBG to generate bucketloads of random bytes, and where hash function performance on relatively short messages (440 bits for NIST "128-bit security",
888 bits for "256-bit") could have a measurable impact. Such an application would not be one of the protocols to which you allude above, such as TLS; rather, it would look like heavy-duty numeric simulations (I know physicists who need gigabytes of random bytes per second, although they do not require cryptographic security).

It still looks far fetched. RNG do not have interoperability issues: the point of a RNG is precisely _not_ to generate the same bytes than any other instance. With HMAC as used in IPsec or TLS, all involved systems must scrupulously follow the specification to derive the same MAC or subkeys, but in a RNG-using system, the RNG design is quite free. Each system can use its own. It is much easier to advocate using a system-specific, optimized RNG, e.g. one of the stream ciphers from the eSTREAM portfolio; or Fortuna, which is AES-based (especially if said system happens to be a big PC with a recent enough Intel processor with the AES-NI opcodes).

Hash_DRBG is very useful in memory-constrained environments, where reusing a hash function implementation may lead to big savings in code footprint. However, I have trouble imagining a memory-constrained environment which also needs to generate lots of random, to the point that the generation of random bytes is an important part of the overall processing cost. Most system which needs many random bytes also do non-trivial things with said random, which dwarfs out the cost of the random generation.

Conceptually, a hash function does not look like a very good candidate for a PRNG: the PRNG should produce large amounts of data from a small seed, while a hash function does the opposite. Some hash function designs can be "reversed", yielding a PRNG built on the same elements.
Panama did that (but it has been broken, at least as a hash function).
Skein can be used in "PRNG mode". Actually, Skein tries to be a "one size fits all" primitive for hashing, MAC, PRNG, block cipher... and this would be a great asset for memory constrained environments.
However, I fear that most of those extra usages will not be investigated as part of the SHA-3 competition.

To sum up: at some point, I will try to measure Hash_DRBG performance (in MBytes of random data produced per second) with the SHA-3 candidates; but I am still unconvinced about the existence of applications for which such a measure has any kind of relevance.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of Jonathan Day [jonathan.day.jobs@gmail.com]

Sent: Friday, August 06, 2010 2:36 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash

There have been solutions to the password issue for some time (virtually no serious password system uses unsalted passwords, for starters) so problems where rainbow tables and the more trivial forms of brute-force attacks are an issue are almost certainly going to end up being due to a lousy specification of the system and not the speed of the hashing function.

PKI (Public Key Infrastructure) sort-of implies anything using public keys, so all certificate-based systems (eg: TLS, SSL) are encompassed by PKI since certificates use public key encryption. Also covered would be any use of digital signatures or a public key encryption system like GnuPG or PGP Not a big deal, but communication between participants seems to have been a bigger problem than the algorithms and I'd hate to see an extremely valid point (PKI needs strong hashes of short messages) be lost over disputes over what is or is not a protocol versus a format versus a frame of reference. That is not a useful discussion unless you're writing an RFC and trying to decide what kind of RFC it is.

As far as TLS and SSL are concerned, one of the biggest reasons I keep getting given for why sites do not use secure communication is that the overheads are too great. It would kill throughput. The ability to hash faster isn't, in itself, sufficient to solve that problem but it is a necessary component. ESMTP over SSL/TLS has had other issues and general users have never really grasped encrypting e-mails, but web traffic and DNS will likely benefit. (Well, DNS depends on whether you trust DNSSEC, which has finally been adopted. That is most definitely a different - and likely even more heated - discussion.) As for numbers of connections, Facebook has so many other problems with security that SSL isn't going to be a serious factor for a while. Most other heavy-usage sites either don't use accounts (eg: the BBC) or don't have accounts that are absolutely critical (eg: Slashdot). And even then, for all the reputation of either of those as heavy-hitters, I seriously doubt they run into the millions of new connections per second. It's possible YouTube does, but frankly I'd consider those accounts less significant than Slashdot's, with the only useful security really being for logging in.

IPSec faces many of the same problems as IPv6, in that you've got to have some minimum of adoption before it becomes interesting to "regular" Internet users and when used end-to-end it breaks NAT. If I remember the history correctly, IPSec was originally a part of IPv6 that was back-ported to IPv4. That part is by-the-by, though. IPSec, although used on many VPN products for creating the tunnel, lacks general use as a means of securing traffic because of problems other than performance. I don't think anyone uses SK/IP any more (a pity, as one-size-fits-all solutions rarely fit anyone). Speeding up hashes probably won't help much with increasing on-the-fly IPSec.

I'm trying to think of other significant users of cryptographically-strong hashes. True "smart cards" like the Mondo needed them, but SHA-3 is 20 years too late for the original market and the fad right now is to have everything on a single portable device rather than have lots of specialized devices with some logic on each.

On Fri, Aug 6, 2010 at 9:01 AM, Danilo Gligoroski <danilo.gligoroski@gmail.com> wrote:

> Thomas Pornin wrote:

> To be complete, I can think of _one_ application where
> benchmarking of short message hashing is accurate, and
> that's brute-forceing hashed passwords, because you can
> make a tight loop which performs many short message hashes
> in a row, and very little else. However, "BMW should be
> selected for SHA-3 because it allows for faster password
> cracking" does not strike me as the best selling slogan ever.

No need to be cynical about the BMW speed on short messages,
and no need to put words (or slogans) in my mouth.
If you are worried that the speed of SHA-3 would be too fast
for password cracking, NIST has already a solution:
"SP 800-132, DRAFT Recommendation for Password-Based Key
Derivation - Part 1: Storage Applications".

On the other hand try to play with a secure web server
that is using its pool of randomness for producing
thousands of random numbers and is serving thousands clients
simultaneously that are making thousands of secure transactions
from different protocols like: PKI, IPsec, TLS, S/MIME ....

All that activity will need cryptographically strong random
numbers (produced by algorithms as Fortuna or similar algorithms
and standards for generating cryptographically strong numbers).
And guess what: There, hash functions are used, and those hash
functions are hashing short messages.

From: hash-forum@nist.gov on behalf of Kelsey, John M. [john.kelsey@nist.gov]
Sent: Friday, August 06, 2010 2:52 PM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: CubeHash

HMAC_DRBG is similar to Hash_DRBG--the speed of the PRNG is based on how quickly we produce outputs. I think a couple of the KDFs in SP-800-57 work this way, too. Also, we have gotten suggestions to write up a tree-based hashing standard to use with SHA2 and SHA3--the obvious ways to do that would involve doing a lot of hashing of relatively short inputs, and the expensive computation at the end might become relevant.

I'm not sure how big a deal most of this is. I don't think many applications need pseudorandom bits at a rate that would give any of the semifinalists big problems, for example. (Normally, you're doing something like generating a symmetric key and IV or some such thing, right? Performance for keypair generation isn't an issue, since the modular arithmetic dominates the cost of the computation.) Similarly, key derivation functions don't usually require vast speed, since they're usually tied to both communications and public key operations.

--John

____
De: hash-forum@nist.gov [hash-forum@nist.gov] En nombre de Thomas Pornin [thomas.pornin@cryptolog.com] Enviado el: viernes, 06 de agosto de 2010 02:20 p.m.
Para: Multiple recipients of list
Asunto: Re: OFFICIAL COMMENT: CubeHash

On Fri, Aug 06, 2010 at 12:01:24PM -0400, Danilo Gligoroski wrote:
> On the other hand try to play with a secure web server that is using
> its pool of randomness for producing thousands of random numbers and
> is serving thousands clients simultaneously that are making thousands
> of secure transactions from different protocols like: PKI, IPsec, TLS,
> S/MIME ....

"Thousands" is hardly a worry. "Millions per second" would raise performance issues, but mere thousands are a piece of cake with any of the proposed hash functions.

TLS uses a hash-based deterministic PRF to derive the symmetric encryption keys and IV. TLS 1.0 and 1.1 use HMAC/MD5 and HMAC/SHA-1 (both simultaneously) while TLS 1.2 uses HMAC/SHA-256. Details are rather convoluted, but if you look closely you will find that all hash function invocations (in TLS 1.2) call the hash function with an input of length no less than 96 bytes, and half of them have inputs of at least 128 bytes. That's short, but not very short. More importantly, the PRF is used only during the handshake. Its execution time is negligible with regards to other handshake-related activity (in particular the asymmetric key exchange, even with the help of hardware accelerators); also, the ensuing symmetric cryptography (for the data to be tunnelled:
symmetric encryption, and integrity checks, then again with HMAC, but on large blocks) will be vastly more expensive. You would be hard pressed to detect any performance variation on a TLS server based on the hash function used in the PRF.

IPsec is similar. Random numbers are used as part of the asymmetric key exchanges, but this does not occur often, if only because such key exchanges require several round trips. When using the ESP protection with a block cipher in CBC mode (specified in RFC 2451), you need one random IV per packet, but RFC 2451 includes the following helpful text:

    The IV field MUST be same size as the block size of the cipher
    algorithm being used. The IV MUST be chosen at random. Common
    practice is to use random data for the first IV and the last block of
    encrypted data from an encryption process as the IV for the next

encryption process.

which means that there will be very few PRNG invocations after all.
There again, no performance bottleneck with the hash function operating on short messages (there _may_ be a performance problem with the integrity checks -- with HMAC in IPsec -- but that's on longer messages, see my previous email).

S/MIME is the application of MIME encoding rules to CMS (RFC 5652), a
PKCS#7 derivative which allows for encryption and signature of arbitrary binary objects. A few random numbers are needed in some parts (asymmetric key exchange, signature with a randomized padding, IV for symmetric encryption...) but they will result in very few hash function invocations per object, and the other object handling costs will be much higher. The size of the needed random data does not grow up with the input data size.

"PKI" is not a protocol; it means "Public Key Infrastructure" which itself relies on a horde of various protocols. For X.509 PKI, most of those protocols are a bit "higher level" and do not bother with describing PRNG, only stating things like "cryptographically secure random source". The CMP protocol (RFC 4210) relies on CRMF (RFC 4211) which itself delegates all the cryptographic work to CMS. As described above, this does not entail any hash function usage where performance on short messages is detectable, let alone critical.

> All that activity will need cryptographically strong random numbers
> (produced by algorithms as Fortuna or similar algorithms and standards
> for generating cryptographically strong numbers).

Fortuna is not a good example here. The bulk work of random number generation in Fortuna is done with a block cipher in CTR mode (usually the AES). A hash function is used only when "reseeding"; when Fortuna is used to generate a lot of random data (a requirement for Fortuna performance to actually have any kind of relevance), reseeding occurs once every megabyte of random, and entails N+1 hash function invocations, where N is the number of "entropy pools" (usually 32). Each hash function invocation processes one pool, except the last, which hashes together the N previous hash function outputs. These hash function invocations may have a non trivial cost with regards to the AES-CTR 1 MB block only if they operate on quite long pools, with at least dozens of kilobytes of data per pool; there are then not "short messages".

So no, there are no hashes-of-short-messages worth speaking of in Fortuna, performance wise. There _may_ be hashing performance issues (if the AES part is very fast, e.g. with the AES-NI opcodes on the newer Intel processors) but such issues may arise only if the entropy pools are big, in which case the hash function processes long messages, not short messages.

A much better example is Hash_DRBG as specified in NIST Special Publication 800-90 (March 2007). With SHA-256, that PRNG will use one
SHA-256 invocation per 256 bits to produce, and that invocation is over a 440-bit input message (there are also other invocation for (re)seeding, but these are rare and do not impact overall performance).

I could imagine some applications using Hash_DRBG to generate bucketloads of random bytes, and where hash function performance on relatively short messages (440 bits for NIST "128-bit security",
888 bits for "256-bit") could have a measurable impact. Such an application would not be one of the protocols to which you allude above, such as TLS; rather, it would look like heavy-duty numeric simulations (I know physicists who need gigabytes of random bytes per second, although they do not require cryptographic security).

It still looks far fetched. RNG do not have interoperability issues: the point of a RNG is precisely _not_ to generate the same bytes than any other instance. With HMAC as used in IPsec or TLS, all involved systems must scrupulously follow the specification to derive the same MAC or subkeys, but in a RNG-using system, the RNG design is quite free. Each system can use its own. It is much easier to advocate using a system-specific, optimized RNG, e.g. one of the stream ciphers from the eSTREAM portfolio; or Fortuna, which is AES-based (especially if said system happens to be a big PC with a recent enough Intel processor with the AES-NI opcodes).

Hash_DRBG is very useful in memory-constrained environments, where reusing a hash function implementation may lead to big savings in code footprint. However, I have trouble imagining a memory-constrained environment which also needs to generate lots of random, to the point that the generation of random bytes is an important part of the overall processing cost. Most system which needs many random bytes also do non-trivial things with said random, which dwarfs out the cost of the random generation.

Conceptually, a hash function does not look like a very good candidate for a PRNG: the PRNG should produce large amounts of data from a small seed, while a hash function does the opposite. Some hash function designs can be "reversed", yielding a PRNG built on the same elements.
Panama did that (but it has been broken, at least as a hash function).
Skein can be used in "PRNG mode". Actually, Skein tries to be a "one size fits all" primitive for hashing, MAC, PRNG, block cipher... and this would be a great asset for memory constrained environments.
However, I fear that most of those extra usages will not be investigated as part of the SHA-3 competition.

To sum up: at some point, I will try to measure Hash_DRBG performance (in MBytes of random data produced per second) with the SHA-3 candidates; but I am still unconvinced about the existence of applications for which such a measure has any kind of relevance.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of Danilo Gligoroski [danilo.gligoroski@gmail.com]
Sent: Saturday, August 07, 2010 12:43 AM
To: Multiple recipients of list
Subject: RE: OFFICIAL COMMENT: CubeHash

Thomas Pornin wrote:
> "Thousands" is hardly a worry. "Millions per second" would raise
> performance issues, but mere thousands are a piece of cake with any of
> the proposed hash functions.
>

In an isolated benchmarking environment where you measure just the speed of hash functions - YES - "thousands" are not a worry. But in running web servers where you have other thousands of processes serving simultaneously thousands of demanding clients, it will become an issue with hash function that is slow on hashing short messages.

>
> TLS uses a hash-based deterministic PRF to derive the symmetric
> encryption keys and IV. TLS 1.0 and 1.1 use HMAC/MD5 and HMAC/SHA-1
> (both simultaneously) while TLS 1.2 uses HMAC/SHA-256. Details are
> rather convoluted, but if you look closely you will find that all hash
> function invocations (in TLS 1.2) call the hash function with an input
> of length no less than 96 bytes, and half of them have inputs of at
> least 128 bytes. That's short, but not very short.
>

Measuring the speed of hash functions on messages long 100,000 or more bytes and measuring the speed of hash functions on messages less than 1,000 bytes gives pretty different figures. And since HMAC have two invocations of the hash function where the second one is with a length of 64 bytes, HMAC of a message that is 128 bytes long (or even more, like 576 or 1536 bytes) can be treated as "hashing short messages".

Regards,
Danilo!

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]

Sent: Saturday, August 07, 2010 6:41 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash (eprint paper comment)

To change the subject from one bizzaro world to another, the problem
with the eprint paper lies in Kaminsky's probability calculations (and not with the methodology).

The first hint that something is amiss is when (on Page 9) he uses Chi-square table to
deduce probability of a simple highly-biased binomial distribution.

He states that if an event has prob <0.001, then in 529 runs the probability of
3 or more events occurring is itself <.001. Indeed if you used a 1-degree of
freedom chi-square statistic and table, that is what you get as p-value.

However, a simple binomial calculation shows that this prob is at least 0.015.

So, I presume he used the same "trick" to arrive at white dots in his pictorial
representation, which can lead to erroneous conclusions.

Charanjit Jutla

```
hash-forum@nist.gov wrote on 07/25/2010 07:23:29 PM:

> [image removed]
>
> OFFICIAL COMMENT: CubeHash
>
> D. J. Bernstein
>
> to:
>
> Multiple recipients of list
>
> 07/25/2010 07:29 PM
>
> Sent by:
>
> hash-forum@nist.gov
>
> Please respond to hash-forum
>
>
> This note responds to two recent eprint papers that discuss CubeHash.
>
>
> 1. http://eprint.iacr.org/2010/262, "Cube test analysis of the
```

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Sunday, August 08, 2010 3:18 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Sat, Aug 07, 2010 at 12:42:46AM -0400, Danilo Gligoroski wrote:
> In an isolated benchmarking environment where you measure just the
> speed of hash functions - YES - "thousands" are not a worry. But in
> running web servers where you have other thousands of processes
> serving simultaneously thousands of demanding clients, it will become
> an issue with hash function that is slow on hashing short messages.

What I mean is that if a web server is in a position of needing to invoke the TLS PRF function thousands of times per seconds, then it effectively is handling thousands of TLS handshakes per seconds, since the TLS PRF is invoked only as part of the TLS handshake, and if you want to run that PRF, you need to perform a handshake. And the computational cost of the operations involved in those handshakes far exceeds the one implied by the PRF by itself, for whatever hash function you use in the PRF.

You could contrive a specially built hash function which is so abysmally slow that the PRF invocation as part of the TLS handshake could become a measurable and non negligible part of the CPU consumption (even if your server does _only_ handshakes and nothing else); but it would take a hash function which is far slower than the slowest of the SHA-3 candidates. By two orders of magnitude, at least.

> Measuring the speed of hash functions on messages long 100,000 or more
> bytes and measuring the speed of hash functions on messages less than
> 1,000 bytes gives pretty different figures.

Apparently, bandwidth on 1000-byte messages seems to be 5% to 25% smaller than bandwidth on very long messages, depending on the involved hash function. I guess that the meaning of "pretty different" is pretty subjective.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Wednesday, August 11, 2010 2:51 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

Gaëtan Leurent writes:
> Agreed, I don't think we really need a 512 bit hash with 512 bit of
> security. However, NIST asked for one, and we all tried to build one.
> Even the first-round version of CubeHash aimed for 512 security...

The second-round version of CubeHash also provides 512-bit preimage security. This security level restricts the parameter choices, of course, but that's also true for all the other SHA-3 proposals. The claims of limited CubeHash security are completely bogus.

If you want to make an efficiency argument, complaining that CubeHash is very slow with the 512-bit-preimage-security parameter choices, then you're welcome to make that argument. My response is that this type of efficiency is of zero value to cryptographic users, and that CubeHash deliberately sacrifices this type of efficiency in favor of other types of efficiency that are indisputably much more important.

> It wouldn't be fair to make such a significant change to the rules.

There's no change in the rules. Everybody is providing the required range of security levels. But the rules _don't_ say which forms of efficiency will be prioritized! There are tradeoffs between (e.g.)

   (1) efficiency for users who want a 128-bit security level,
   (2) efficiency for users who want a 256-bit security level, and
   (3) efficiency for users who also want 512-bit preimage security.

Some SHA-3 candidates emphasize #1 while doing a bad job of #2 and #3.
Some SHA-3 candidates emphasize #3 even if this means hurting #1 and #2.
CubeHash is in the middle, sacrificing #3 for the sake of #2 and #1.
This does _not_ mean that CubeHash is limited to 256-bit security.

Here's another example to illustrate these tradeoffs. At a 128-bit security level, Luffa has comparable speed to CubeHash on a Core 2. At a 256-bit security level, Luffa is much slower than CubeHash. Does this mean that Luffa is limited to 128-bit security? Of course not! Let's not confuse efficiency complaints with security complaints.

> Could you give a clear statement of the expected strength of CubeHash?
> (The document strength.pdf claims 2^512, but it doesn't really make
> sense...)

strength.pdf says "See the CubeHash specification for recommended parameters r, b." The specification proposes

   CubeHash16/32224 for SHA3224,
   CubeHash16/32256 for SHA3256,
   CubeHash16/32384 for SHA3384normal,
   CubeHash16/32512 for SHA3512normal,
   CubeHash16/1384 for SHA3384formal, and
   CubeHash16/1512 for SHA3512formal.

Of course, if you want 512-bit preimage security, your only option is the SHA-3-512-formal proposal.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Wednesday, August 11, 2010 4:39 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

Stefan.Lucks@uni-weimar.de writes:
> in cryptography, leaving a *security* *margin* means to require (or
> claim) more security than actually required.

No! That's not at all what a security margin means.

Let's look at the classic example. The designers of Serpent predicted that 16 rounds of Serpent, with a full 256-bit key, would provide full
2^256 security. However, they recommended _32 rounds_ of Serpent---a huge security margin against advances in cryptanalysis.

Does the Serpent security margin mean that 32-round Serpent provides more than 2^256 security---that it costs more than 2^256 to attack? Of course not.

Does the Serpent security margin mean that 32-round Serpent is harder to break than 16-round Serpent? No. As far as we can tell, the designers were correct: breaking 16-round Serpent already costs 2^256 simple operations. Extra rounds increase the attacker's costs only linearly.

What the Serpent security margin means is that Serpent isn't using just the bare minimum amount of computation necessary for security against all known attacks. Serpent is feeding the attacker's data through many more computations, to defend against unknown attacks that would manage to get through a few more computations.

Of course, if you keep scaling the number of rounds _down_, then you'll eventually reach a point where Serpent loses security. The 2008 paper by Dunkelman, Indesteege, and Keller claims an attack cost

* around 2^250 for 12 rounds and
* below 2^140 for 11 rounds.

One could describe this as a massive drop in security when the Serpent security parameter changes by just 1---but one could equally well describe it as a massive increase in security when the Serpent security parameter changes by just 1. Does this indicate a problem in Serpent? Of course not! It's an illustration of the fast diffusion in Serpent.

CubeHash is in the same position, providing a massive security margin.
There are a few different options, depending on the user's security requirements, but each option is carefully designed to do much more computation than necessary to protect against all known attacks. For
example:

(1) Many users want 128-bit security, i.e., protection against all
    attacks performing fewer than 2^128 simple operations. For these
    users, the official recommendation is CubeHash16/32--256: i.e.,
    16 rounds after every 32-byte block, with 256 bits of output.

(2) Some users are concerned that 128-bit security won't be adequate
    for the long term, and instead want 256-bit security. For these
    users, the official recommendation is CubeHash16/32--512: i.e.,
    16 rounds after every 32-byte block, with 512 bits of output.

    For comparison, CubeHash6/32 is estimated to be breakable in
    2^180 operations, but CubeHash16/32 has more than twice as many
    rounds, a very solid security margin. The 32-byte block size is

also a solid factor of 2 away from danger: generic attacks are still stuck at the target >=2^256 for block sizes <=64 bytes.

(3) There is a wacky notion that 256-bit security isn't enough, and that hash functions should provide 512-bit security---but that it's okay for this extra security to be only for preimages, not for collisions, and that it's okay for this to be completely broken by quantum computers, as if performing 2^256 operations were easier than building a quantum computer.

For anyone who subscribes to this wacky notion, the official recommendation is CubeHash16/1--512: i.e., 16 rounds after every 1-byte block, with 512 bits of output. A careful analysis of attack costs---see the original CubeHash submission---shows that the 1-byte block size here is overkill, even more conservative than the number of rounds.

In all cases I'm confident that CubeHash also has many more finalization rounds than necessary for security, although this hasn't attracted any third-party analysis yet.

At each of these security levels, scaling down the CubeHash security parameters will eventually lose security---with a sudden jump downwards in attack costs, just as in the Serpent case. Does this indicate a problem with CubeHash? Of course not! It's an illustration of how fast the CubeHash diffusion is.

> The full "normal" cube hash is at b=32, where resistance against
> preimage attacks is already poor. We compared the real thing with the
> variant, where we tuned the security parameter to the worse by one: b=33.

Here you are confusing two completely different security levels, and drawing completely bogus conclusions about CubeHash as a result.

The CubeHash specification tells you to use CubeHash16/32--512 if you're satisfied with 2^256 security against all attacks (as any sensible user will be). The specification also tells you to use CubeHash16/1--512 if you're demanding 2^512 preimage security. In both situations, CubeHash provides the target level of resistance against preimage attacks, and provides a solid security margin; see above.

You are misrepresenting the CubeHash specification when you falsely claim that CubeHash provides "poor" preimage resistance. The CubeHash specification DOES NOT propose CubeHash16/32--512 for people who want
2^512 preimage security; it proposes CubeHash16/1--512. When you complain that CubeHash16/32--512 does not provide 2^512 preimage security, you are attacking a proposal that was never made.

Suppose someone points to the obvious 2^112 collision attack against Skein-224, correctly observes that 2^112 is far below 2^256, argues that 2^256 is the desired security level for future SHA-3 users, and then runs around saying that Skein provides "extremely poor" collision security. How would you like that? That's exactly the same sort of misrepresentation that you're engaging in---taking a proposal for a lower security level and complaining that it doesn't provide a higher security level. Don't you think the SHA-3 discussions would be more productive without this sort of misrepresentation?

> Does the author of CubeHash maintain his claim CubeHash16/32 to be "a
> conservative proposal with a comfortable security margin"?

Absolutely. There are different CubeHash proposals for different security levels, and each of them is a conservative proposal with a comfortable security margin. In particular:

* CubeHash16/32--512, the "SHA-3-512-normal" proposal, provides the target security level of 2^256 against all attacks, and has a confidence-inspiring security margin, in rounds _and_ in block size.

* If you want more---in particular, 2^512 preimage security---then the specification tells you to use CubeHash16/1--512, the

"SHA-3-512-formal" proposal, which again is a conservative proposal with a comfortable security margin.

The first-round CubeHash submission proposed block size 1 but included implementations with block sizes 1,2,4,8,16,32,64 and had a detailed discussion of the security impact of this choice ("Appendix: complexity of generic attacks")---in particular, the 2^(384 +epsilon) preimage attack for b=32. The same 2^(384+epsilon), except for minor differences in the epsilon from varying levels of sloppiness in evaluating attack costs, was confirmed by

* 2008 Khovratovich--Nikolic--Weinmann, presenting exactly the same attack as if it were new;

* 2008 Aumasson--Brier--Meier--Naya-Plasencia--Peyrin, generalizing the attack idea to non-power-of-2 block sizes and confirming the wisdom of choosing power-of-2 block sizes; and

* 2010 Ferguson--Lucks--McKay, your new paper, presenting the same generalized attack as if it were new.

The second-round CubeHash submission recommends b=32 for sensible users satisfied with 2^ 256 security, while preserving the b=1 recommendation for anyone who wants 2^512 preimage security.

You keep using loaded language to suggest that you've found a new attack threatening the security of CubeHash. In fact, this attack has been known for years---it appeared in the original CubeHash submission for all relevant cases!---and the CubeHash parameters have always been fully protected against it.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Jean-Philippe Aumasson [jeanphilippe.aumasson@gmail.com]
Sent: Thursday, August 12, 2010 2:14 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Wed, Aug 11, 2010 at 8:48 PM, D. J. Bernstein <djb@cr.yp.to> wrote:
>
> Gaëtan Leurent writes:
>> It wouldn't be fair to make such a significant change to the rules.
>
> There's no change in the rules. Everybody is providing the required
> range of security levels. But the rules _don't_ say which forms of
> efficiency will be prioritized! There are tradeoffs between (e.g.)
>
> (...)
```

AFAIU, facts are:

1. Dan did suggest to change the rules of the competition:
"(...) The "formal" CubeHash proposals are designed to comply with these requests, taking b=1 for 384-bit and 512-bit output, but I hope that continued discussion can reach consensus that these requests should be dropped."

2. NIST did not make any statement confirming that the requirements published in the Federal Register Notice wouldnt be modified.

Now, there are two possibilities:

3. NIST agrees with Dan's pro-CubeHash common sense arguments and changes the rules of the competition, such that CubeHash16/32512 fits in the requirements for SHA-3.

4. NIST sticks to the present requirements, which we can expect, and thus CubeHash's tradeoffs become a non-issue, as CubeHash16/32512 does not provide the required 512-bit security.

Will NIST make a statement to close the discussion?

From: hash-forum@nist.gov on behalf of Stefan.Lucks@uni-weimar.de
Sent: Thursday, August 12, 2010 5:00 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Wed, 11 Aug 2010, D. J. Bernstein wrote:

>
> Stefan.Lucks@uni-weimar.de writes:
> > in cryptography, leaving a *security* *margin* means to require (or
> > claim) more security than actually required.
>
> No! That's not at all what a security margin means.
> [...]
> What the Serpent security margin means is that Serpent isn't using
> just the bare minimum amount of computation necessary for security
> against all known attacks. Serpent is feeding the attacker's data
> through many more computations, to defend against unknown attacks that
> would manage to get through a few more computations.

No contradiction: Specifying many more computations is something I definitively would
count as "offering more security".

> Of course, if you keep scaling the number of rounds _down_, then
> you'll eventually reach a point where Serpent loses security. The 2008
> paper by Dunkelman, Indesteege, and Keller claims an attack cost
>
>    * around 2^250 for 12 rounds and
>    * below 2^140 for 11 rounds.
>
> One could describe this as a massive drop in security when the Serpent
> security parameter changes by just 1---but one could equally well
> describe it as a massive increase in security when the Serpent
> security parameter changes by just 1. Does this indicate a problem in
> Serpent? Of course not! It's an illustration of the fast diffusion in Serpent.

Agreed! But if Serpent where specified at 12 rounds, instead of 32, these results would
actualy indicate a very severe problem for Serpent. If the authors of Serpent would have
specified it at 12 rounds, and such results whould have been known during the AES
competition, this would surely (and rightfully, IMHO) been the reason to drop Serpent from
any further consideration as a potential AES.

Actually, in SHA-3, CubeHashX/32 is in a similar situation as a 12-rounds Serpent would
have been: highly agressively optimised to be about as performant as the competitors. The
authors of Serpent preferred the larger security margin and specified Serpent at 32
rounds, but then, Serpent turned out to be a bit slow.

Now, you recommend different variants for different security levels:

   (1) CubeHash16/32--256 for "128-bit security",
   (2) CubeHash16/32--512 for "256-bit security", and
   (3) The ultra-ultra-slow (*) CubeHash16/1--512 for "512-bit security"
       against preimage attacks.

I'd like to point out that I don't agree with the split between (2) and (3). "512-bit
security" against preimage attacks isn't a "wacky notion", it is just another way to
describe the security margin.

The issue is the following:
```

* SHA-512 and most of the other SHA-3 semi-finalists actually offer the
  security level of (3) at competitive speed, while

* CubeHash only leaves the choice between an ultra-slow variant at the
  full security level and decent performance at the lower security levels
  (1) and (2).

Since we (i.e., most of the SHA-3 finalists) are able to provide the security level (3) at decent speed, why should users be forced to choose?

Or why should we even think of choosing a SHA-3, which, at the best of our current cryptanalytic knowledge, provides a much lesser security margin than the SHA-2-512 we already have?

One possible answer would be "because CubeHash16/32 is 50 times faster than SHA-2-512" or the like. But CubeHash isn't that stunningly fast.

```
> > The full "normal" cube hash is at b=32, where resistance against
> > preimage attacks is already poor. We compared the real thing with
> > the variant, where we tuned the security parameter to the worse by one: b=33.
>
> Here you are confusing two completely different security levels, [...]
```

I am not confusing anything!

I just stick with the rule that a good hash function has to provide about n bits of resistance against preimages for n bit hashes. A hash function which doesn't provide that security is poor. Unlike you, I don't make my own rules.

```
>    * 2010 Ferguson--Lucks--McKay, your new paper, presenting the same
>      generalized attack as if it were new.
```

Dan, please read the paper, before you accuse us of plagiarism! The previous paper claimed an attack which we showed would not work. We provided a similar attack which actually does work.

```
> The second-round CubeHash submission recommends b=32 for sensible
> users satisfied with 2^256 security, while preserving the b=1
> recommendation for anyone who wants 2^512 preimage security.
>
> You keep using loaded language
```

Dan, you call me (and anyone else who asks for more than 256 bits of secuity against preimage attacks for a 512-bit hash function) not sensible. This is, what I understand from the above paragraph of yours.

So why do you complain about any "loaded language" of mine?

So long

Stefan

(*) If we compare CubeHash16/32 with 12 rounds of Serpent, then
    CubeHash12/1 would provide the performance of 384 rounds of Serpent.

```
--
------ Stefan Lucks   --  Bauhaus-University Weimar  --   Germany  ------
               Stefan dot Lucks at uni minus weimar dot de
------  I  love  the  taste  of  Cryptanalysis  in  the  morning!  ------
```

From: hash-forum@nist.gov on behalf of Stefan.Lucks@uni-weimar.de
Sent: Thursday, August 12, 2010 7:37 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Thu, 12 Aug 2010, Stefan.Lucks@uni-weimar.de wrote:

> Since we (i.e., most of the SHA-3 finalists) are able to provide the
> security level (3) at decent speed, why should users be forced to choose?

Ooops! I meant to write "semi-finalists", not "finalists".

Sorry!

Stefan

```
--
------ Stefan Lucks   --  Bauhaus-University Weimar  --   Germany  ------
               Stefan dot Lucks at uni minus weimar dot de
------  I  love  the  taste  of  Cryptanalysis  in  the  morning!  ------
```

From: hash-forum@nist.gov on behalf of Christopher Drost [chris.drostie@gmail.com]
Sent: Thursday, August 12, 2010 3:25 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Thu, Aug 12, 2010 at 8:13 AM, Jean-Philippe Aumasson <jeanphilippe.aumasson@gmail.com> wrote:
>
> Will NIST make a statement to close the discussion?
>

I am of course not NIST, but I just don't see why this is taking the form of an argument. It strikes me that most of these statements are true:

a. Yes, at present it looks like the other candidates (and indeed, SHA2) are "more secure" than the fast version of CubeHash-512 in a tangible way.

b. Yes, that tangible way is functionally useless right now. Whether it increases the chance of a functional attack within the next fifty to a hundred years is probably impossible to evaluate and, at any rate, has not yet been discussed much. (After fifty to a hundred years, the robot wars will begin and our 2010 taste in hash functions will be a moot point.)

c. Yes, in most hash applications, the speed of the hash function is not the bottleneck.

d. Yes, I would rather have more implementations in more languages -- portable C, Java, optimized C, Javascript, Python, Ruby, and whatever else you've got.

e. No, I really don't care if their first drafts are slow relative to the machine-code-optimized C. On some platforms you simply don't have a choice, and on others you simply don't care about the speed.
Something is always better than nothing.

f. Yes, the random oracle model should probably assume something other than an O(1) query time to the random oracle, at the very least because there might be an easy side-channel distinguisher if it didn't.

g. Yes, NIST has a tough choice ahead.

So. Can't we all just get along? Why does this have to be an argument?

At least, let us focus on features.

I want to express first that the following remarks are premature, as I have not been able to implement or work with any of the following candidates: ECHO, Fugue, Grøstl, Hamsi, JH, Luffa, SHAvite-3, or SIMD.
The last one that I have been trying to implement and play around with was JH, and I find its specification very difficult for me personally to understand. I have spent the past several months mostly focused on my academic career and haven't had time to hack on my sha3-js project.

But among the ones that I /do/ have experience with, I would like to single out both Keccak and Skein as being (a) simple enough that I have programmed them in Javascript and (b) sophisticated enough to be usable as stream ciphers. What one traditionally has to do if one is developing an application which encrypts data under a password is something like this:

1. import the cipher, a hash function, and a random numbers source.
Pray that the cipher comes with chaining modes built in.

2. Define a padding function and its inverse, to pad a message to an integer multiple of 128 bits.
3. Define encrypt(password, data): obtain a random salt, set key = truncate(hash(salt, password)), obtain a random nonce, initialize the cipher with a chaining mode, the key, and the nonce as IV. Encrypt
pad(data) with this cipher. Return (salt, nonce, ciphertext).
4. Define decrypt(salt, nonce, ciphertext): initialize the cipher as above, decrypt the ciphertext, unpad it, and return the unpadded data.

With a hash-function-as-stream-cipher, this becomes:

1. Import the hash and a random numbers source.
2. Define encrypt(password, data): obtain a random nonce, then return (nonce, decrypt(nonce, password, data)).
3. Define decrypt(nonce, password, ciphertext): return hash(nonce, password, len(ciphertext)) XOR ciphertext.

I think this is obviously more simple and preferable. Stream ciphers like ChaCha and RadioGatun do not have good penetration into standard cryptography libraries, and it would be nice if sha3 could "carry them in". Ultimately, PySkein is easier to use to encrypt a set of data under a password than the standard PyCrypto library. Keccak would probably also be easier to use, if there were a Python API for it.
(See point "d" above.)

I would like to subtract a couple points from Skein because there exists no standard 32-bit variant at this time. In this vein, I feel that CubeHash and Shabal should get extra points for being very compact in their 32-bit implementations.

I would then like to add a couple points back to Skein for defining, as part of their SHA3 API, standard entrance points for both the password and the nonce in both examples. The more that the SHA3 library specifies such things, the less that us casual-programmers-who-want-crypto will have to get frustrated over the details. But: it is already good that we are shifting from inputs in $\{0, 1\}^n$ to inputs in $\{0, 1\}^*$. That already makes my life much easier.

I just hope that *these* considerations are not excluded from NIST's decision at this stage. Security and speed are of course high-priority concerns, but they aren't exclusive. I don't know precisely what has led to the persistence of MD5 and SHA1, but it surely doesn't help that SHA1 and SHA2 didn't offer any new functionality. People who want tree hashing can't use SHA2. People who want a stream cipher can't use SHA2. People who want a message authentication code need to use SHA2 with an HMAC mode. Perhaps others who follow this list might want to offer functions that they would like to see in the final SHA3 spec, above and beyond "hash this bit-string for me."

-- Chris Drost

(for the curious, some of my work so far: http://code.drostie.org/sha3/)

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Thursday, August 12, 2010 6:14 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

D. J. Bernstein wrote on 11 Aug 2010 22:39:21 +0200:

> The CubeHash specification tells you to use CubeHash16/32--512 if
> you're satisfied with 2^256 security against all attacks (as any
> sensible user will be).

Could you be a bit more precise about this "2^256 security against all attacks"? In particular, what is the expected security of CubeHash16/32--512 for a preimage attack with 2^64 targets? Is it 2^256 or 2^192?

--
Gaëtan Leurent

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Friday, August 13, 2010 3:53 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

Let me put my security claims more bluntly. I think it's clear that

* my CubeHash16/32--224 proposal is at least as secure as Skein-224,
* my CubeHash16/32--256 proposal is at least as secure as Skein-256,
* my CubeHash16/1--384 proposal is at least as secure as Skein-384, and
* my CubeHash16/1--512 proposal is at least as secure as Skein-512,

in every case with more confidence-inspiring security margins: CubeHash is much more conservatively designed than Skein. Furthermore, CubeHash provides an intermediate CubeHash16/32--512 proposal that I think hits a sweet spot, balancing high security and high all-around efficiency in a way that Skein can't match.

Note that CubeHash has a very small state size, 128 bytes. The original CubeHash submission carefully analyzes what one might call "capacity"
attacks based on this state size, and concludes that these attacks cost more than 2^512 simple operations for CubeHash.../1--512. Subsequent papers republished the same attacks as if they were new, and the SHA-3 Zoo incorrectly reported a preimage attack slightly below 2^512, based on nothing more than sloppy analysis of the attack costs. NIST, in its second-round selection report, correctly observed that this "attack"
wasn't news.

Stefan's recent claims of low CubeHash security are not based on any new attacks; in fact, they are not based on any attacks at all. What Stefan is doing is nothing more than mixing up the CubeHash proposals---taking the lower-security proposals, excitedly pointing out that they have lower security, and trying to deceive the reader into believing that this is some sort of security problem in CubeHash.

Stefan.Lucks@uni-weimar.de writes:
> Actually, in SHA-3, CubeHashX/32 is in a similar situation as a
> 12-rounds Serpent would have been: highly agressively optimised to be
> about as performant as the competitors.

False. An aggressively optimized version would be CubeHash8/64--512, running faster than 3.5 Core 2 cycles/byte---even faster than BMW!

But I don't think this is the right way to choose security parameters.
The actual CubeHash proposals are much more conservatively designed, with many more rounds and a much smaller block size (depending on the target security level), giving the attacker far less control over the CubeHash state. This increases hashing time, of course, but it's much more important to build confidence than to set speed records.

> "512-bit security" against preimage attacks isn't a "wacky notion", it
> is just another way to describe the security margin.

Again Stefan's use of the phrase "security margin" is completely out of whack with the community's standard use of the phrase for more than ten years. My last message discussed the Serpent example in detail, and I don't see any point in commenting further.

As for "wacky notion": Stefan's quote omits critical context, making it highly deceptive.
What I actually wrote was

> There is a wacky notion that 256-bit security isn't enough, and that
> hash functions should provide 512-bit security---but that it's okay
> for this extra security to be only for preimages, not for collisions,
> and that it's okay for this to be completely broken by quantum

computers, as if performing 2^256 operations were easier than
   building a quantum computer.

If Stefan wants to argue that this notion isn't wacky, he should be able to explain why he thinks that a 2^256 quantum preimage attack is less of a threat than a much more expensive non-quantum preimage attack, and he should be able to explain this without misusing the phrase "security margin."

> Dan, please read the paper, before you accuse us of plagiarism! The
> previous paper claimed an attack which we showed would not work. We
> provided a similar attack which actually does work.

The authors of the 2008 paper have already commented that the "new"
attack is essentially the second attack in Section 4.3 of the 2008 paper.

> Since we (i.e., most of the SHA-3 finalists) are able to provide the
> security level (3) at decent speed, why should users be forced to choose?

This is the only point where I think Stefan is asking a reasonable question. My answer is that CubeHash provides security levels #1 (2^128) and #2 (2^256) with much better all-around efficiency than other SHA-3 candidates. Achieving this efficiency meant sacrificing some performance at security level #3 (#2 plus 2^512 preimage until quantum computers are built), but efficiency for #1 and #2 is clearly much, much, much more important than efficiency for #3.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Stefan.Lucks@uni-weimar.de
Sent: Friday, August 13, 2010 9:11 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

On Fri, 13 Aug 2010, D. J. Bernstein wrote:

> > Since we (i.e., most of the SHA-3 finalists) are able to provide the
> > security level (3) at decent speed, why should users be forced to choose?
>
> This is the only point where I think Stefan is asking a reasonable
> question.

I also think my above question is reasonable. So, finally, we have found a point where the two of us agree on!

> My answer is that CubeHash provides security levels #1 (2^128) and #2
> (2^256) with much better all-around efficiency than other SHA-3
> candidates.

Now, this is a clever phrase: you compare CubeHash16/32 with "other SHA-3 candidates", with is logically the same as "for some SHA-3 candidates", but not equivalent to "all other SHA-3 candidates". ;-)

Looking at the eBASH tables <http://bench.cr.yp.to/results-hash.html>, it turns out that CubeHash does actually perfom better than *some* other
SHA-3 candidates.

Nevertheless, this is hardly a convincing answer to my question!

Observe that the same tables reveal that *some* other SHA-3 semi-finalists actually provide much better all-around efficiency than CubeHash16/32.
And, unlike CubeHash16/32, none of these other semi-finalists suffer from a certificational weakness against preimage attacks.

So long

Stefan

```
--
------ Stefan Lucks   --  Bauhaus-University Weimar  --   Germany  ------
               Stefan dot Lucks at uni minus weimar dot de
------  I  love  the  taste  of  Cryptanalysis  in  the  morning!  ------
```

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Friday, August 13, 2010 12:30 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
D. J. Bernstein wrote on 11 Aug 2010 20:50:57 +0200:

> There's no change in the rules. Everybody is providing the required
> range of security levels. But the rules _don't_ say which forms of
> efficiency will be prioritized! There are tradeoffs between (e.g.)
>
>    (1) efficiency for users who want a 128-bit security level,
>    (2) efficiency for users who want a 256-bit security level, and
>    (3) efficiency for users who also want 512-bit preimage security.
>
> Some SHA-3 candidates emphasize #1 while doing a bad job of #2 and #3.
> Some SHA-3 candidates emphasize #3 even if this means hurting #1 and #2.
> CubeHash is in the middle, sacrificing #3 for the sake of #2 and #1.
> This does _not_ mean that CubeHash is limited to 256-bit security.

The rules says that you have to design _one_ function for each size, and not one function
for each imaginable security level (that would be a nightmare to analyse).

As far as I can tell, the reference code of CubeHash with a 512 bit output implements
CubeHash16/32-512, and the test vectors in the submission package also correspond to
CubeHash16/32-512.  Therefore I was assuming that the official CubeHash submission was
CubeHash-normal and not CubeHash-formal.  Am I being mistaken?

--
Gaëtan Leurent
```

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Friday, August 13, 2010 10:31 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

I said this in the first SHA-3 workshop, and I'll say it again in the second: CubeHash is the _smallest_ high-security SHA-3 proposal.

For example, Bernet et al. reported a CubeHash ASIC implementation using just 7630 gate equivalents, which they called "particularly appealing for lightweight implementations." There's no cheating here: there's no "free external memory"; it's not "core functions only"; and there's no compromise of any sort in security. The same very small circuit supports 512-bit output for full 2^256 security (and even the CubeHash16/1--512 proposal for 512-bit preimage resistance, if anyone cares).

Can anyone show me another SHA-3 candidate that fits full functionality into this area? I've seen a few reports of ~10000 gate equivalents for 256-bit output for some candidates, but that's only 2^128 security.
What's the gate count when users want 512-bit output? What's the gate count when chip designers need to provide a flexible circuit that allows some users to select 256-bit output while other users select 512-bit?

Stefan Lucks writes:
> All in all Skein and some other semi-finalists (and perhaps even
> SHA-512) provide better security and better efficiency, compared to CubeHash16/32.

No. I agree that the quite reasonable software speed of CubeHash16/32 (e.g., 13 cycles/byte on a Core 2) is beaten by some aggressively designed candidates such as Skein, but I think that this is clearly outweighed by a much more important type of efficiency: namely, CubeHash's exceptionally small hardware profile.

How is Stefan going to convince a hardware designer to dedicate space to Skein-512? What's the gate count for Skein-512, compared to the 7630 gates for CubeHash16/1--512? Is Stefan going to tell hardware designers that they have to limit security, switching to Skein-256-256 (which is still quite a lot larger than CubeHash!), because of Skein-512's poor efficiency in this critical metric? How is this Skein-256-256 hardware supposed to interoperate with a software implementation using Skein-512?
Are protocol designers supposed to tell software to use Skein-256-256, compromising security for everybody?

> CubeHashX/32 is vulnerable to certificational attacks, unlike the
> other SHA-3 semi-finalists.

This is yet another indisputably false statement from Stefan regarding the CubeHash submission. Let's start with the basics: Is it true that the CubeHash second-round SHA-3 candidate is a simple "CubeHashX/32", as Stefan claims? No! Anyone who reads the submission can see that what it actually proposes is

```
   CubeHash16/32-224 for SHA-3-224,
   CubeHash16/32-256 for SHA-3-256,
   CubeHash16/32-384 for SHA-3-384-normal,
   CubeHash16/32-512 for SHA-3-512-normal,
   CubeHash16/1-384 for SHA-3-384-formal, and
   CubeHash16/1-512 for SHA-3-512-formal.
```

This proposal provides 1-byte blocks for the last two options, precisely to protect against the well-known attacks that Stefan is talking about and that were already discussed in detail in the first-round submission.
There are no "certificational attacks" against this proposal.

The submission also explains how silly it is to aim above 2^256 security for SHA-3, and says that "all real-world cryptographic applications" can simply use CubeHash16/32. This doesn't mean that the higher-security options aren't part of the proposal; what it means is that sensible users will be happy with 2^256 security.

  [1: use more rounds than necessary]
  [2: avoid "certificational" attacks]
> As much as I understand your mail, Dan, you only consider the first
> approach, and disregard the second. Is that correct, Dan?

This is not correct. CubeHash uses many more rounds than necessary, _and_ uses a much smaller block size than necessary, _and_ provides full protection against "certificational" attacks---although of course the user is free to reduce the security parameters for higher efficiency.
All of this is discussed in detail in the CubeHash submission.

Stefan's false accusations of "certificational" attacks against CubeHash are based entirely on his pretense that the CubeHash proposal doesn't include 512-bit-preimage-security options. This is as absurd as

   * taking the undisputed 2^112 collision attack against Skein-224,
   * arguing that 2^112 is not an acceptable SHA-3 security level,
   * concluding that Skein-224 doesn't provide acceptable security, and
   * concluding that Skein is subject to a "certificational" attack.

I think that we can have a much more reasonable discussion without this ridiculous sort of exaggeration. The simple fact is that all the SHA-3 candidates, including CubeHash and Skein, support the entire required range of security levels. If a new attack shows that a function actually fails to achieve the promised security, then the function will be eliminated---but there is no such attack against CubeHash, and the solid security margins in the CubeHash proposals make me much more confident in CubeHash's continued safety than in the safety of other candidates.

Of course, efficiency varies in different ways across security levels. I have no problem with people pointing out different aspects of efficiency that they find important. In particular, I would welcome an honest discussion of the extent to which current and future users will benefit from efficiency at various security levels, such as

   (1) security against all attacks costing below 2^128,
   (2) security against all attacks costing below 2^256, and
   (3) security against pre-quantum preimage attacks costing below 2^512.

Prioritizing these different aspects of efficiency leads to quite different views of the candidates.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Stefan.Lucks@uni-weimar.de
Sent: Saturday, August 14, 2010 9:00 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Fri, 13 Aug 2010, D. J. Bernstein wrote:

> Stefan Lucks writes:
> > CubeHashX/32 is vulnerable to certificational attacks, unlike the
> > other SHA-3 semi-finalists.
>
> This is yet another indisputably false statement from Stefan regarding
> the CubeHash submission. Let's start with the basics: Is it true that
> the CubeHash second-round SHA-3 candidate is a simple "CubeHashX/32",
> as Stefan claims? No!

Let me see:

I have pointed out the certificational attacks on CubeHashX/32, and Dan claims this an
"indisputably false statement" because there is another proposed variant (CubeHashX/1)
that doesn't suffer from the same weakness.
I am curious if anyone on this list can understand Dan's logic.

In any case, I wonder why Dan assumes me claiming "the CubeHash second-round SHA-3
candidate is a simple 'CubeHashX/32'". I had been carefully writing "CubeHashX/32", just
to make sure that I was talking about a certain variant of CubeHash and not about all
variants.

I won't argue any further, since I am sure that the readers on this list can make their
own judgment about who's statements are false.

So long

Stefan

P.S.:
  Dan, as far as I am concerned, I am fed up with the attitude you have
  shown in this discussion. I won't waste more of my time with responding
  to your mails.
  *plonk*




> Anyone who reads the submission can see that what it actually proposes
> is
>
>    CubeHash16/32-224 for SHA-3-224,
>    CubeHash16/32-256 for SHA-3-256,
>    CubeHash16/32-384 for SHA-3-384-normal,
>    CubeHash16/32-512 for SHA-3-512-normal,
>    CubeHash16/1-384 for SHA-3-384-formal, and
>    CubeHash16/1-512 for SHA-3-512-formal.
>
> This proposal provides 1-byte blocks for the last two options,
> precisely to protect against the well-known attacks that Stefan is
> talking about and that were already discussed in detail in the first-round submission.
> There are no "certificational attacks" against this proposal.
>
> The submission also explains how silly it is to aim above 2^256
> security for SHA-3, and says that "all real-world cryptographic
> applications" can simply use CubeHash16/32. This doesn't mean that the
> higher-security options aren't part of the proposal; what it means is
> that sensible users will be happy with 2^256 security.
>
>   [ 1: use more rounds than necessary ]
>   [ 2: avoid "certificational" attacks ]
> > As much as I understand your mail, Dan, you only consider the first
> > approach, and disregard the second. Is that correct, Dan?
>
> This is not correct. CubeHash uses many more rounds than necessary,
> _and_ uses a much smaller block size than necessary, _and_ provides
> full protection against "certificational" attacks---although of course
> the user is free to reduce the security parameters for higher efficiency.
> All of this is discussed in detail in the CubeHash submission.
>
> Stefan's false accusations of "certificational" attacks against
> CubeHash are based entirely on his pretense that the CubeHash proposal
> doesn't include 512-bit-preimage-security options. This is as absurd
> as
>
>    * taking the undisputed 2^112 collision attack against Skein-224,
>    * arguing that 2^112 is not an acceptable SHA-3 security level,
>    * concluding that Skein-224 doesn't provide acceptable security, and
>    * concluding that Skein is subject to a "certificational" attack.
>
> I think that we can have a much more reasonable discussion without
> this ridiculous sort of exaggeration. The simple fact is that all the
> SHA-3 candidates, including CubeHash and Skein, support the entire
> required range of security levels. If a new attack shows that a
> function actually fails to achieve the promised security, then the
> function will be eliminated---but there is no such attack against
> CubeHash, and the solid security margins in the CubeHash proposals
> make me much more confident in CubeHash's continued safety than in the safety of other
candidates.
>
> Of course, efficiency varies in different ways across security levels.
> I have no problem with people pointing out different aspects of
> efficiency that they find important. In particular, I would welcome an
> honest discussion of the extent to which current and future users will
> benefit from efficiency at various security levels, such as
>
>    (1) security against all attacks costing below 2^128,
>    (2) security against all attacks costing below 2^256, and
>    (3) security against pre-quantum preimage attacks costing below 2^512.
>
> Prioritizing these different aspects of efficiency leads to quite
> different views of the candidates.
>
> ---D. J. Bernstein
>    Research Professor, Computer Science, University of Illinois at
> Chicago
>
>

--
------ Stefan Lucks   --  Bauhaus-University Weimar  --   Germany  ------
               Stefan dot Lucks at uni minus weimar dot de
------  I  love  the  taste  of  Cryptanalysis  in  the  morning!  ------
```

From: hash-forum@nist.gov on behalf of Paulo S. L. M. Barreto [pbarreto@larc.usp.br]
Sent: Saturday, August 14, 2010 11:45 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

I wasn't going to reply to this boring discussion, but, oh my, let's go...

On Sat, August 14, 2010 10:00, Stefan.Lucks@uni-weimar.de said:
>
> On Fri, 13 Aug 2010, D. J. Bernstein wrote:
>
>> Stefan Lucks writes:
>> > CubeHashX/32 is vulnerable to certificational attacks, unlike the
>> > other SHA-3 semi-finalists.
>>
>> This is yet another indisputably false statement from Stefan
>> regarding the CubeHash submission. Let's start with the basics: Is it
>> true that the CubeHash second-round SHA-3 candidate is a simple
>> "CubeHashX/32", as Stefan claims? No!
>
> Let me see:
>
> I have pointed out the certificational attacks on CubeHashX/32, and
> Dan claims this an "indisputably false statement" because there is
> another proposed variant (CubeHashX/1) that doesn't suffer from the same weakness.
> I am curious if anyone on this list can understand Dan's logic.

Yes, it turns out that I for one can understand it. The subset of CubeHashX/32 that is vulnerable to a certificational attack is *not* a SHA-3 finalist as Dan repeatedly pointed out, yet you (incorrectly) imply it is.

> In any case, I wonder why Dan assumes me claiming "the CubeHash
> second-round SHA-3 candidate is a simple 'CubeHashX/32'". I had been
> carefully writing "CubeHashX/32", just to make sure that I was talking
> about a certain variant of CubeHash and not about all variants.

Sorry to say, it's regrettable that what you've been carefully writing is, rather, a mix of subsets of CubeHash that were proposed as a SHA-3 candidate with other subsets that weren't. CubeHashX/32 includes reduced variants that are susceptible to a certificational attack (and were not proposed for SHA-3) and variants that are not susceptible (some of these were proposed).

Curiously, this lengthy discussion gave me a much more positive view of CubeHash by making it clear what exactly Dan proposes, what he doesn't, and what is really known about CubeHash's security properties. CubeHash was not among my favorite SHA-3 candidates, but now, well...

> I won't argue any further, since I am sure that the readers on this
> list can make their own judgment about who's statements are false.

Not arguing any further is a sensible decision. I'll share it.

Paulo.

From: hash-forum@nist.gov on behalf of Paul Hoffman [paul.hoffman@vpnc.org]
Sent: Saturday, August 14, 2010 12:16 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

At 11:44 AM -0400 8/14/10, Paulo S. L. M. Barreto wrote:
>Curiously, this lengthy discussion gave me a much more positive view of
>CubeHash by making it clear what exactly Dan proposes, what he doesn't,
>and what is really known about CubeHash's security properties.

I am someone who has gotten quite confused about what is and is not being proposed by Dan here, and I'm glad to hear that I'm not alone. Paulo: could you do a brief summary here on the list of your view of what is in and out of scope? Dan's earlier attempts may have been, er, clouded in the presentation style.

--Paul Hoffman, Director
--VPN Consortium

From: hash-forum@nist.gov on behalf of Chen Ke-Fei Lin [chen.ke.fei.lin@gmail.com]
Sent: Saturday, August 14, 2010 1:16 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Sat, Aug 14, 2010 at 5:44 PM, Paulo S. L. M. Barreto <pbarreto@larc.usp.br> wrote:
>
> > I have pointed out the certificational attacks on CubeHashX/32, and
> > Dan claims this an "indisputably false statement" because there is
> > another proposed variant (CubeHashX/1) that doesn't suffer from the same weakness.
> > I am curious if anyone on this list can understand Dan's logic.
>
> Yes, it turns out that I for one can understand it. The subset of
> CubeHashX/32 that is vulnerable to a certificational attack is *not* a
> SHA-3 finalist as Dan repeatedly pointed out, yet you (incorrectly) imply it is.


I like you logic.
Proposals under the word "formal" do not suffer of any weaknesses.


>
> Curiously, this lengthy discussion gave me a much more positive view
> of CubeHash by making it clear what exactly Dan proposes, what he
> doesn't, and what is really known about CubeHash's security
> properties. CubeHash was not among my favorite SHA-3 candidates, but now, well...


I don't like you logic.
There is nothing to like in "formal" CubeHash, since it takes centuries to hash a message
and obtain 512-bit digest.

C.K.F.Lin
```

From:	hash-forum@nist.gov on behalf of Stefan.Lucks@uni-weimar.de
Sent:	Saturday, August 14, 2010 1:32 PM
To:	Multiple recipients of list
Subject:	Re: OFFICIAL COMMENT: CubeHash

```
On Sat, 14 Aug 2010, Paulo S. L. M. Barreto wrote:

>
> I wasn't going to reply to this boring discussion, but, oh my, let's go...
>
> On Sat, August 14, 2010 10:00, Stefan.Lucks@uni-weimar.de said:
> >
> > On Fri, 13 Aug 2010, D. J. Bernstein wrote:
> >
> >> Stefan Lucks writes:
> >> > CubeHashX/32 is vulnerable to certificational attacks, unlike the
> >> > other SHA-3 semi-finalists.
> >>
> >> This is yet another indisputably false statement from Stefan
> >> regarding the CubeHash submission. Let's start with the basics: Is
> >> it true that the CubeHash second-round SHA-3 candidate is a simple
> >> "CubeHashX/32", as Stefan claims? No!
> >
> > Let me see:
> >
> > I have pointed out the certificational attacks on CubeHashX/32, and
> > Dan claims this an "indisputably false statement" because there is
> > another proposed variant (CubeHashX/1) that doesn't suffer from the same weakness.
> > I am curious if anyone on this list can understand Dan's logic.
>
> Yes, it turns out that I for one can understand it. The subset of
> CubeHashX/32 that is vulnerable to a certificational attack is *not* a
> SHA-3 finalist as Dan repeatedly pointed out, yet you (incorrectly) imply it is.

Dan proposed the following six variants for SHA-3 (quoting from his most recent mail):

Dan>    CubeHash16/32-224 for SHA-3-224,
Dan>    CubeHash16/32-256 for SHA-3-256,
Dan>    CubeHash16/32-384 for SHA-3-384-normal,
Dan>    CubeHash16/32-512 for SHA-3-512-normal,
Dan>    CubeHash16/1-384 for SHA-3-384-formal, and
Dan>    CubeHash16/1-512 for SHA-3-512-formal.

> CubeHashX/32 includes reduced variants that are susceptible to a
> certificational attack (and were not proposed for SHA-3)

Not for round 1, but the current specification (for round 2) explicitely specify the six
variants from above.

So long

Stefan

--
------ Stefan Lucks   --  Bauhaus-University Weimar  --   Germany  ------
               Stefan dot Lucks at uni minus weimar dot de
------  I  love  the  taste  of  Cryptanalysis  in  the  morning!  ------
```

From: hash-forum@nist.gov on behalf of Chen Ke-Fei Lin [chen.ke.fei.lin@gmail.com]
Sent: Saturday, August 14, 2010 4:32 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

CubeHash16/32-224 for SHA-3-224, ------------------ Speed ~ SHA2, Security ~ SHA2
CubeHash16/32-256 for SHA-3-256, ------------------ Speed ~ SHA2, Security ~ SHA2
CubeHash16/32-384 for SHA-3-384-normal, --------- Speed ~ SHA2, Security Less Less Less than SHA2
CubeHash16/32-512 for SHA-3-512-normal, --------- Speed ~ SHA2, Security Less Less Less than SHA2
CubeHash16/1-384 for SHA-3-384-formal, and ------ Speed less less less than SHA2, Security ~ SHA2
CubeHash16/1-512 for SHA-3-512-formal.------------ Speed less less less than SHA2, Security ~ SHA2

Less, less, less, ...

Rivest made noble move, withdraw MD6 because low speed. (maybe withdraw was mistake) NIST made mistake accepted Round 2 formal candidate with slower speed than MD6.

C.K.F.Lin

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Sunday, August 15, 2010 10:28 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Sat, Aug 14, 2010 at 01:31:33PM -0400, Stefan.Lucks@uni-weimar.de wrote:
> Dan proposed the following six variants for SHA-3 (quoting from his
> most recent mail):
>
> Dan>    CubeHash16/32-224 for SHA-3-224,
> Dan>    CubeHash16/32-256 for SHA-3-256,
> Dan>    CubeHash16/32-384 for SHA-3-384-normal,
> Dan>    CubeHash16/32-512 for SHA-3-512-normal,
> Dan>    CubeHash16/1-384 for SHA-3-384-formal, and
> Dan>    CubeHash16/1-512 for SHA-3-512-formal.
```

For completeness: the "submission requirement" from NIST called for one set of parameters for each of the digest output sizes; this is reflected by the format of the test vectors (for a given output size and a given input length, there is one input message and one corresponding output).
In the case of CubeHash, the reference implementation and the test vectors are for CubeHash16/32 for all output sizes.

Hence it seems safe to assume that "CubeHash as submitted" is
CubeHash16/32 for all four output sizes, because that's what the submitter himself chose when it came to generating the test vectors. The (round 2) specification even includes the sentence: "the tweak amounts to a proposal of CubeHash16/32 as SHA–3". Seems fairly clear to me. The "formal" parameters are just a distraction, to be ignored. The CubeHash specification quite strongly suggests that you would have to be kinda stupid to use the "formal" parameters anyway.

--Thomas Pornin

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Sunday, August 22, 2010 5:09 AM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: CubeHash

In a talk on Cell/GPU performance a few days ago at CHES 2010, Joppe Bos quoted some of the speeds reported by supercop-20100509 on thoth, an old 900MHz Athlon (622): in particular, 72.37 cycles/byte for CubeHash16/32.

The point of this note is that the current CubeHash software (unrolled3 with gcc -march=i386 -O -fomit-frame-pointer) runs at just 43.59 cycles/byte on the same CPU. For comparison, here is the performance on the same CPU of other functions providing 2^256 collision security:

```
      25%    50%    75%
    10.15  10.17  10.19 shabal512
    23.81  23.81  23.81 bmw512
    40.30  40.35  40.35 keccak
    51.65  51.88  52.37 shavite3512
    52.05  52.13  52.22 skein512
    64.08  64.09  64.11 luffa512
    67.06  67.09  67.14 blake64
    69.80  69.81  69.92 sha512
    76.05  76.08  76.14 echo512
    94.22  95.14  99.01 simd512
   114.13 114.20 114.96 groestl512
   128.53 128.55 128.59 jh512
```

In the meantime, Thomas Pornin has confirmed that my current CubeHash software is considerably faster than his "sphlib" implementation on every one of his selected CPUs:

* 1.5x faster on a PowerPC 750;
* 1.4x faster on a Broadcom MIPS;
* 1.2x faster on his HP calculator; and
* 1.5x faster on a Marvell ARMv5.

CubeHash is naturally very highly parallelizable, giving the programmer tremendous flexibility to reorder operations. The current CubeHash implementations take advantage of this to improve locality of reference.
The previous Athlon data used the "cubehash1632/simple" implementation, which didn't include any optimizations---it was meant to show in the simplest possible way how CubeHash works.

Side note #1: Pornin makes the very strange claim that in C the order of operations is "_irrelevant_". In fact, as any experienced C programmer knows, the opposite is true: the order of operations often has a very heavy effect on performance.

Perhaps Pornin is misunderstanding the "as-if rule" in the C standard.
What the rule actually says is that compilers are _allowed_ to change the order of operations, within certain limits. The rule does not say, and does not even suggest, that the compiler output will be independent of the original order of operations in the C code.

Side note #2: Pornin describes gcc's instruction scheduling and register allocation as "suboptimal" and claims that "there is very little that can be done about it with C code." In fact, the opposite is true:
competent programmers have a huge amount of power over gcc's output, even without any use of inline assembly. The speedup from "simple" to "unrolled3" on an Athlon, nearly 1.7x faster, is a good illustration of this; note that "unrolled3" is a highly portable C

implementation.

I don't mean to suggest that gcc is the best programming tool. Quite often it is _easier_ to write high-speed code in assembly language than to write high-speed code using gcc. There are more options; part of my research---currently with some SHA-3 candidates as case studies---is aimed at further reducing the amount of programmer time required for writing high-speed code. In any event, it is clear that the overall cost of producing good SHA-3 code, even for the most complicated SHA-3 candidate, is negligible compared to the massive amount of CPU power that society is going to spend on SHA-3 computations.

Side note #3: As I've mentioned before, I've peeked at the sphlib code for some other SHA-3 candidates, and I find myself quite unimpressed with the level of optimization. CubeHash isn't the only function where other implementations have already been demonstrated to solidly beat sphlib on many different CPUs.

Pornin claims that the sphlib slowdowns are "similar between the compared functions." In fact, the slowdown factors vary heavily from one candidate to another. As an extreme example, on a Q6600 in 64-bit mode, CubeHash is more than 3 times faster than the sphlib implementation of CubeHash. If Pornin seriously believes that Shabal can be made anywhere near 3 times faster than the sphlib implementation of Shabal, why is he unable to show us an implementation demonstrating this?

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Thomas Pornin [thomas.pornin@cryptolog.com]
Sent: Sunday, August 22, 2010 8:56 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash

```
On Sun, Aug 22, 2010 at 05:07:16AM -0400, D. J. Bernstein wrote:
> In the meantime, Thomas Pornin has confirmed that my current CubeHash
> software is considerably faster than his "sphlib" implementation on
> every one of his selected CPUs:
>
>    * 1.5x faster on a PowerPC 750;
>    * 1.4x faster on a Broadcom MIPS;
>    * 1.2x faster on his HP calculator; and
>    * 1.5x faster on a Marvell ARMv5.

I confirm the figures on the three first processors. The Marvell ARMv5 is not one of my
"selected CPUs", and I do not have access to it, so I cannot confirm anything about it.
Also, the word "considerably" is highly subjective and I do not think it can possibly be
"confirmed"
(and, in any way, I do not confirm it; 1.5x is not what I would call "considerable").


> The speedup from "simple" to "unrolled3" on an Athlon, nearly 1.7x
> faster, is a good illustration of this

If "unrolled3" consists in some kind of loop unrolling (as the name
suggests) then this is not what we were talking about. We were talking about a branchless
sequence of operations, the kind which can be modelled as an acyclic circuit.

Theoretically, the compilers can also unroll loops when the loop count is known at
compile-time, but compiler writers are a bit wary about it because it increases memory
footprint, thus making it harder on the instruction cache -- how much harder requires
program-wide analysis that the compiler is unable or unwilling to perform. The gcc
documentation specifically says that it will not unroll loops at optimization levels
-O1 and -O2. The unrolling decision (and how much to unroll) is to be taken by the
programmer.


> Pornin claims that the sphlib slowdowns are "similar between the
> compared functions." In fact, the slowdown factors vary heavily from
> one candidate to another. As an extreme example, on a Q6600 in 64-bit
> mode, CubeHash is more than 3 times faster than the sphlib
> implementation of CubeHash. If Pornin seriously believes that Shabal
> can be made anywhere near 3 times faster than the sphlib
> implementation of Shabal, why is he unable to show us an
> implementation demonstrating this?

You already tried that weaseling suggestion three months ago, it did not work at that time
either. Since you were demonstrating some uncanny ability not to understand, I assumed
that I did not express myself properly and I added a paragraph in my article, which I
reproduce here:

<< It shall be noted that whenever applicable, sphlib performance was
   compared with the submitted optimized code and the performance
   figures reported in the submission packages; sphlib was always on par
   with those figures. More precisely, if the submitted code kept to
   ``portable C'' (no MMX/SSE2), then we compiled and benchmarked it on
   our x86 PC, in both 32-bit and 64-bit mode, using the same compiler
   and optimization flags than for sphlib. The sphlib code always
   offered comparable performance, or better performance in a few cases
   (e.g. with SHAvite-3 and a 512-bit output, sphlib is twice faster,
   because the submitted optimized code has an unrolled loop which
   exceeds the 32 kB of level-1 instruction cache of our Intel Q6600 --
   the SHAvite-3 submitters optimized for an AMD CPU which has a 64 kB
   level-1 cache). This validates the effectiveness of the optimization
   strategy of sphlib. >>
```

I invite you to read again the expressions "portable C" and "same compiler and optimization flags".

--Thomas Pornin

From: Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Sunday, October 03, 2010 3:44 PM
To: hash-function@nist.gov
Cc: hash-forum@nist.gov
Subject: OFFICIAL COMMENT: CubeHash (Round 2)

Attachments: CubeHash-quantum.pdf

CubeHash-quantum .pdf (238 KB)

Dear all,

Here is a short note discussing quantum attacks on CubeHash.

The result described in the note is a straightforward extension of previous symmetry-based attacks, using a quantum computer. We show that Grover's algorithm can be used to reach a symmetric state efficiently, and this yield a quantum preimage attack for CubeHash-16/32-512 (aka CubeHash-normal-512) in time 2^192 (as opposed to 2^256 for SHA-512).

This seems to contradict previous security statements of the designer.

Regards,

--
Gaëtan Leurent

Quantum Preimage and Collision Attacks on CubeHash

Gaëtan Leurent

Abstract. In this short note we show a quantum preimage attack on CubeHash-normal-512 with complexity 2^{192}. This kind of attack is expected to cost 2^{256} for a good 512-bit hash function, and we argue that this violates the expected security of CubeHash. The preimage attack can also be used as a collision attack, given that a generic quantum collision attack on a 512-bit hash function require 2^{256} operations, as explained in the CubeHash submission document.
This attack only use very simple techniques: we use the symmetry properties of CubeHash which were already described in the submission document and have been analyzed in detail in [1,5], together with Gover's algorithm which is also discussed in the submission document.

1 Introduction

CubeHash is a hash function designed by Bernstein and submitted to the SHA-3 competition [2]. It has been accepted for the second round of the competition.

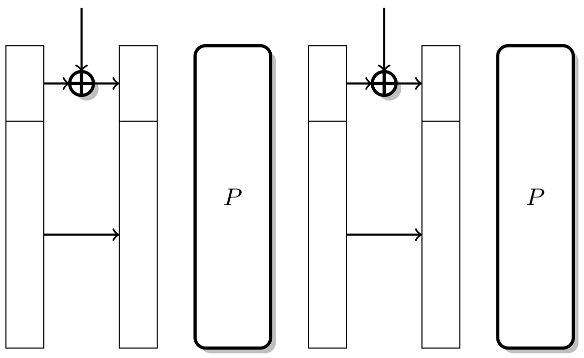

Fig. 1. CubeHash follows the sponge construction

1.1 CubeHash Versions

CubeHash is built following the sponge construction with a 1024-bit permutation. The small size of the state allows for compact hardware implementations, but it is too small to build a fast 512-bit hash function satisfying NIST security requirements. The submission document defines two versions of CubeHash with 512 bits of output: CubeHash-normal (*aka* CubeHash-16/32-512) and CubeHash-formal (*aka* CubeHash-16/1-512), but does not explicitly state which one is to be considered as a SHA-3 candidate. On the one hand, CubeHash-normal reaches a reasonable speed at the cost of security by using a capacity of 768 bits: this implies that preimage attacks only cost 2^{384}. On the other hand, CubeHash-formal reaches a reasonable security at the cost of speed by using a capacity of 1016 bits, and is much slower than the other second round candidates.

Interestingly, the reference code, the test vectors, and the benchmarks (section 2.B.2) are given for CubeHash-normal[1], but the security claims (section 2.B.4) are made for CubeHash-formal.

In this note we study CubeHash-normal-512, *i.e.* CubeHash-16/32-512.

1.2 Expected Security of CubeHash-normal

Strangely enough, the submission document of CubeHash does not make any formal security claims for CubeHash-normal-512, which is obviously the version that will be targeted by cryptanalysts. Moreover, the expected security of CubeHash-normal does not seem to follow one of the standard security notions.

The submission document only acknowledge that the best known preimage attack against CubeHash-normal-512 has complexity 2^{384}, and argue that it is not sensible to consider attacks with complexity higher than 2^{256}. This led many cryptographers to believe that CubeHash-normal had an expected security against preimage attack of 384 bits, but the designer stated on the NIST mailing list that CubeHash-normal-512 was actually only offering 256 bits of security, and declined to give a more formal statement similar to the claims for CubeHash-formal in the submission document.

In the absence of a clear security claim from the designer, one can either assume that the function does not offer any security, or extrapolate from the pieces of information available. Given that the main motivation for not offering 512 bits of security against *classical* preimage attacks is the availability of a *quantum* preimage attack with complexity 2^{256}, we believe that CubeHash-normal has been designed with quantum attacks in mind. Therefore, in the absence of any specific claim regarding quantum attacks, we assume that it should offer the same level of security as a good hash function against such attacks.

More explicitly, when discussing attacks on CubeHash, the submission document states (in section 2.B.5):

> Of course, these attack strategies need to be compared to attacks that apply to *all* h-bit hash functions:
> - Parallel collision search (1994 van Oorschot–Wiener), finding h-bit collisions in time roughly $2^{h/2}/A$ on circuits of total area A.
> - Parallel quantum preimage search (1996 Grover), finding h-bit preimages in time roughly $2^{h/2}/A^{1/2}$ on quantum circuits of total area A.

Our new observation precisely leads to an attack more efficient than the parallel quantum preimage search of Grover.

[1]Additionally, the title of the tweak document “16 times faster”, should actually be “twice as slow” if CubeHash-formal is the main candidate.

When discussing the expected security of CubeHash-normal and CubeHash-formal, Bernstein explained on the hash forum that he was interested in three different security notion [4]:

1. security against all attacks costing below 2^{128},
2. security against all attacks costing below 2^{256}, and
3. security against pre-quantum preimage attacks costing below 2^{512}.

It seems that the three version of CubeHash submitted as second-round candidates target those three security levels (see also [3]): CubeHash-16/32-256 for level (1), CubeHash-16/32-512 for level (2), and CubeHash-16/1-512 for level (3). This would imply that CubeHash-16/32-512 is supposed to resist to quantum attacks up to 2^{256}.

1.3 CubeHash Symmetries

The design of CubeHash is very symmetric and does not use any constants. Therefore, there exists many symmetry classes for the permutation. This was stated in the submission document, and later work have provided an explicit description and analysis of the symmetry classes [1,5].

The most efficient way to use those symmetries is to use a symmetry class such that the message expansion can produce symmetric messages. For instance, we can use the symmetry class called C_2 in [1]. A state is symmetric if:

$$\forall i, j, k, l \quad x_{ijkl0} = x_{ijkl1}$$

When b is 32, as is the case for CubeHash-normal, the message injection gives control over $x_{00klm}, \forall k, l, m$. Therefore, in order to reach a symmetric state, one just has to reach a state satisfying the following 384-bit equation:

$$x_{01kl0} = x_{01kl1} \qquad x_{10kl0} = x_{10kl1} \qquad x_{11kl0} = x_{11kl1} \qquad \forall\, k, l \tag{1}$$

and the message injection can be used to make the state fully symmetric. This is expected to cost 2^{384} on average.

This can be used to mount a preimage attack with the following steps:

1. Find a message A reaching a symmetric state from the IV.
2. Find a message D reaching a symmetric state backwards from the target value (you should first extend the target value into a full state, and compute the finalisation backwards).
3. Build 2^{192} symmetric messages B_i. Compute the states reached after processing $A \| B_i$.
4. Build 2^{192} symmetric messages C_j. Compute the states reached after processing $C_j \| D$ backwards.
5. With a good probability, there will be a pair of values that match on the last 768 bits (384 of those bits come from free because of the symmetry). Then, use a message bloc X to match the first 256 bits. This yields a preimage $A \| B_{i_0} \| X \| C_{j_0} \| D$

Steps 1 and 2 cost 2^{384}, while step 3 and 4 cost 2^{192}. Note that the meet in the middle technique can actually be done without memory. This attack has essentially the same complexity as a capacity-based attack when b is a power of two, but it becomes more efficient when b is not a power of two[2].

2 New Observation

The most expensive part of the symmetry based attack is to reach a symmetric state. However, it turns out that it is actually relatively easy to reach a symmetric state using Grover's algorithm on a quantum computer. Indeed, reaching a state satisfying equation (1) is equivalent to finding a preimage of zero for a hash function that would iterate the round function as CubeHash, and whose output would be (without any blank rounds):

$$\begin{array}{llll}
x_{01000} \oplus x_{01001} & x_{01010} \oplus x_{01011} & x_{01100} \oplus x_{01101} & x_{01110} \oplus x_{01111} \\
x_{10000} \oplus x_{10001} & x_{10010} \oplus x_{10011} & x_{10100} \oplus x_{10101} & x_{10110} \oplus x_{10111} \\
x_{11000} \oplus x_{11001} & x_{11010} \oplus x_{11011} & x_{11100} \oplus x_{11101} & x_{11110} \oplus x_{11111}
\end{array}$$

This is a 384-bit hash function, therefore Grover's algorithm requires time 2^{192} to find a preimage of zero on a small quantum computer.

Then we can use the same meet-in-the-middle technique as in the previous symmetry based attack, which requires time 2^{192} on a classical computer. This gives a preimage attack on CubeHash-normal with complexity 2^{192}, assuming that quantum computers are available.

2.1 Alternative attack

Instead of using a meet-in-the-middle technique with complexity 2^{192} on a *classical* computer, one can reach any given symmetric state (for instance, the all-zero state) for a cost of 2^{192} on a *quantum* computer using another call to Grover's algorithm.

3 Conclusion

Our work shows that CubeHash-normal can only provide the level of preimage resistance of a 384-bit hash function, even if you believe that *classical* preimage attacks are irrelevant because of more efficient *quantum* preimage attacks. Additionally, we show that the symmetry properties of the round function of CubeHash do actually lead to cryptographic weaknesses of the hash function.

[2]The proposed versions of CubeHash use powers of two for b, but the designer occasionally discussed versions of CubeHash with other values of b

References

1. Aumasson, J.P., Brier, E., Meier, W., Naya-Plasencia, M., Peyrin, T.: Inside the hypercube. In Boyd, C., Nieto, J.M.G., eds.: ACISP. Volume 5594 of Lecture Notes in Computer Science., Springer (2009) 202–213
2. Bernstein, D.J.: Cubehash specification. Submission to NIST (Round 2) (2009)
3. Bernstein, D.J.: Are options prohibited for SHA-3? NIST Hash Forum (23 Aug 2010) Message id: `<20100822232203.27042.qmail@cr.yp.to>`.
4. Bernstein, D.J.: OFFICIAL COMMENT: CubeHash. NIST Hash Forum (14 Aug 2010) Message id: `<20100814021923.7676.qmail@cr.yp.to>`.
5. Ferguson, N., Lucks, S., McKay, K.A.: Symmetric states and their structure: Improved analysis of cubehash. Cryptology ePrint Archive, Report 2010/273 (2010) `http://eprint.iacr.org/`.

From: D. J. Bernstein [djb@cr.yp.to]
Sent: Sunday, October 03, 2010 10:23 PM
To: hash-function@nist.gov
Cc: hash-forum@nist.gov
Subject: OFFICIAL COMMENT: CubeHash (Round 2)

The security claims for CubeHash are in the "CubeHash expected strength (2.B.4)" document. I stand by those claims, and by every other claim made in the CubeHash submission.

Starting with the second-round submission, that document has explicitly covered the complexity of quantum attacks, and has been careful to consistently reduce all preimage-resistance exponents by a factor of 2, exactly as anyone familiar with Grover's algorithm would expect.

I have never claimed 2^512 preimage security, or 2^256 quantum preimage security, for CubeHash16/32-512. I'm sorry if the "512" confuses people who think that output size dictates security level, but I really don't think that artificially tying these notions together is the best way to design a hash function.

When Gaetan wrote to me about this obvious 2^192-quantum-operation application of Grover's algorithm, I wrote back to him saying

> Please read the CubeHash documentation. The security claims are in the "CubeHash expected strength (2.B.4)" document.

He now claims to have contradicted "previous security statements of the designer," but is unable to back this up with a quote of the security statement that he claims to have disproven. The reason that he's unable to do this is that his claim is incorrect. All of the CubeHash security statements are holding up just fine.

---D. J. Bernstein
Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Monday, October 04, 2010 5:30 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Follow Up Flag: Follow up
Flag Status: Purple

D. J. Bernstein wrote on 04 Oct 2010 05:35:11 +0200:

> I have never claimed 2^512 preimage security, or 2^256 quantum
> preimage security, for CubeHash16/32-512. I'm sorry if the "512"
> confuses people who think that output size dictates security level,
> but I really don't think that artificially tying these notions
> together is the best way to design a hash function.

Please read section 1.2 of my note.

I won't say anything more until you make a formal security claim for CubeHash-16/32-512, as was requested by NIST.

--
Gaëtan

From: hash-forum@nist.gov on behalf of Christopher Drost [chris.drostie@gmail.com]
Sent: Monday, October 04, 2010 5:31 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Follow Up Flag: Follow up
Flag Status: Purple

On Mon, Oct 4, 2010 at 5:34 AM, D. J. Bernstein <djb@cr.yp.to> wrote:

> I have never claimed 2^512 preimage security, or 2^256 quantum
> preimage security, for CubeHash16/32-512.
> [...]
> He now claims to have contradicted "previous security statements of
> the designer," but is unable to back this up with a quote of the
> security statement that he claims to have disproven.

Dan,

I believe the security claim is resident in these statements:

2.B.4 'CubeHash-512 is expected to provide preimage resistance of approximately 512 bits, but quantum computers are expected to reduce preimage resistance to approximately 256 bits.'

2.B.4 'See the CubeHash specification for recommended parameters r, b.'

2.B.1 'For all real-world cryptographic applications, the "formal" versions here can be ignored, and the tweak amounts to a proposal of CubeHash16/32 as SHA-3.'

I see only three possible ways that we can take all of these statements as reasonable, and none of them seem particularly pleasant:

(1) You are misremembering the claims that you actually made, so that you think you "have never" made claims which you certainly did make explicitly. Certainly, you are chiding people for thinking "that output size dictates security level" when your own submission seems to indicate that your preimage security level /is/ the output size.

(2) You made all of those claims for CubeHash8/1, and in fact have *not* to date made *any* security claims for your SHA-3 proposal. In other words, all of your security claims are for "recommended parameters" which you recommended to be "ignored". It would be almost as if 2.B.4 began with a statement "please ignore this section, too"
or so.

(3) You think that a "resistance of 512 bits" is not the same as "2^512 security," for some reason or another. I as a cryptanalytic outsider would have assumed that these all should be the same, but perhaps when you /said/ "resistance of 512 bits" you meant "2^256 security" or some such.

Of these, I think (3) would need the most explanation and (1) is the most reasonable. On the other hand, Gaetan's paper accuses you explicitly of attempting to perform (2). Part of the problem is that section 2.B.4 does not make the same distinction between "SHA-3-formal" and "SHA-3-normal" that the paper has made.

So, I would like to ask for an official clarification of the CubeHash spec: Are there any security claims for the CubeHash SHA-3-normal proposal? If so, were they the generic claims provided in section 2.B.4?

-- Chris Drost

From: hash-forum@nist.gov on behalf of Thomas Bonik [bonik1977@googlemail.com]
Sent: Tuesday, October 05, 2010 1:37 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On Mon, Oct 4, 2010 at 9:30 AM, Christopher Drost <chris.drostie@gmail.com> wrote:
> Of these, I think (3) would need the most explanation and (1) is the
> most reasonable. On the other hand, Gaetan's paper accuses you
> explicitly of attempting to perform (2). Part of the problem is that
> section 2.B.4 does not make the same distinction between
> "SHA-3-formal" and "SHA-3-normal" that the paper has made.
>

Ahh, if NIST were a little bit more careful when they decided which SHA-3 candidate will go from Round 1 to Round 2 they would stayed on their own principles described in the "Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA–3) Family". In that case we would avoid this situation and CubeHash would not be in Round 2. The request has a broad list of evaluation criteria, requirements and expectations that must be followed during this competition. For this post I am specifically talking about the section 4.A.iii "Additional Security Requirements of the Hash Functions" where there is a clear statement:

"In addition to the specific requirements mentioned above, NIST expects the SHA–3 algorithm of message digest size n to meet the following security requirements at a minimum. These requirements are believed to be satisfiable by fairly standard hash algorithm constructions; any result that shows that the candidate algorithm does not meet these requirements will be considered to be a serious attack.

• Collision resistance of approximately n/2 bits, • Preimage resistance of approximately n bits, • Second-preimage resistance of approximately n-k bits for any message shorter than 2^k bits, • Resistance to length-extension attacks, and ..."

Pay attention to the part: "any result that shows that the candidate algorithm does not meet these requirements will be considered to be a serious attack" and compare it with the second-preimage security of CubeHash16/32-512. FULL STOP!

====
Thomas Bonik

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Tuesday, October 05, 2010 7:28 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Christopher Drost writes:
> 2.B.4 'CubeHash-512 is expected to provide preimage resistance of
> approximately 512 bits, but quantum computers are expected to reduce
> preimage resistance to approximately 256 bits.'

Exactly. You have to continue reading to see which parameters r,b are recommended for this insane security level. Those recommendations have been r=16 and b=1 since the start of the second round, and those recommendations continue to be r=16 and b=1.

Let me emphasize that quantum algorithms do _not_ break these parameter recommendations in time 2^192; they need time 2^256. All of the CubeHash security claims are holding up just fine, contrary to recent claims. The first-round recommendations are also holding up just fine.

> 2.B.4 'See the CubeHash specification for recommended parameters r, b.'

Exactly. The "Parameter recommendations" section of the specification tells you that there are two 512-bit options---

* CubeHash16/32512 for SHA3512normal,
* CubeHash16/1512 for SHA3512formal

---and then tells you that the "formal" proposal

> is aimed at users who are (1) concerned with attacks using 2^384
> operations, (2) unconcerned with quantum attacks that cost far less,
> and (3) unaware that attackers able to carry out 2^256 operations
> would wreak havoc on the entire SHA3 landscape, forcing SHA3 to be
> replaced no matter which function is selected as SHA3

while the "normal" proposal "is aimed at sensible users." If you want the insane security level described in 2.B.4 for CubeHash-512 then this text clearly tells you to use r=16 and b=1.

> 2.B.1 'For all real-world cryptographic applications, the "formal"
> versions here can be ignored, and the tweak amounts to a proposal of
> CubeHash16/32 as SHA-3.'

Exactly. Real-world cryptographic applications don't need, and will never need, the insane security level that we've been discussing here.
CubeHash16/32 deliberately reduces capacity to 768 bits so that it can increase the block size to 256 bits, dramatically improving hash speed without allowing any attacks within the lifetime of the universe.

Perhaps I should emphasize that users can push this tradeoff further.
Users comfortable with the limited 576-bit capacity (2^288 preimage security) hidden inside all of the Keccak speed advertisements would probably also be comfortable with 512-bit capacity and 512-bit blocks in CubeHash, _doubling_ the CubeHash speed. This can be supported as a run-time option by CubeHash hardware, with negligible area compromise, and moves CubeHash into the #1 position in hardware throughput per area (using a sensible 2-cycle-per-round design, not a Tillich design).

The only way to allow a large block size _and_ a 1024-bit capacity is to change the state size to something far above 1024 bits. When I designed CubeHash I started out with a 2048-bit state size for exactly this reason---but then I realized that having a hash function fit everywhere is much more important than having it be fast at insane security levels.

Of course, I also considered bundling two incompatible hash functions together, like SHA-256 and SHA-512, but the reality is that hardware implements only the smaller function, making the larger function unusable in any protocol important enough to talk to hardware. See

   http://cr.yp.to/papers.html#interop

for a much more detailed discussion of this issue. When I see the Skein team advertising that Skein-256-256 "can be implemented using about 100 bytes of RAM," and the Luffa team advertising 3220 Mbps at 90nm for
Luffa-256 in 13980 GEs, I find myself imagining how horrifying it would be to try to upgrade to higher security levels in 15 years in a world full of hardware using Skein-256-256 or Luffa-256. I'm glad that we have some SHA-3 candidates (CubeHash, ECHO, JH, Keccak, Shabal) that weren't designed that way.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Christopher Drost [chris.drostie@gmail.com]
Sent: Tuesday, October 05, 2010 7:45 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Dan --

I would like to ask for an official clarification of the CubeHash
spec: Are there *any* security claims for the CubeHash SHA-3-normal proposal? If so, were they the generic claims provided in section 2.B.4?

I shouldn't have to ask that twice, I guess, but you only meticulously quoted the parts of my email which you yourself wrote, and didn't respond to any of the parts that I wrote. (With that much, it's no small wonder that each of your paragraphs starts with the word 'Exactly'.)

-- Chris

From: hash-forum@nist.gov on behalf of Thomas Bonik [bonik1977@googlemail.com]
Sent: Tuesday, October 05, 2010 9:03 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
On Tue, Oct 5, 2010 at 11:28 AM, D. J. Bernstein <djb@cr.yp.to> wrote:
>
> Christopher Drost writes:
>> 2.B.4 'CubeHash-512 is expected to provide preimage resistance of
>> approximately 512 bits, but quantum computers are expected to reduce
>> preimage resistance to approximately 256 bits.'
>
> Exactly. You have to continue reading to see which parameters r,b are
> recommended for this ***insane security level*** .
> <snip>
>
> If you want the ***insane security level*** described in 2.B.4 for
> CubeHash-512 then this text clearly tells you to use r=16 and b=1.
>
> Real-world cryptographic applications don't need, and will never need,
> the ***insane security level*** that we've been discussing here.


NIST asked for 512 bits of (second)preimage security for SHA-3 with
512 bits of digest and according to your post it is ***insane security level***.
Why NIST asked for ***insane security level*** ?
Are you implying that NIST is insane asking for ***insane security
level*** or what?
What about designer teams that followed the "insane" NIST requirements and submitted
functions that are offering that ***insane security level***? Are you implying that all of
them are insane?

====
Thomas Bonik
```

From:	hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent:	Wednesday, October 06, 2010 10:20 AM
To:	Multiple recipients of list
Subject:	OFFICIAL COMMENT: CubeHash (Round 2)

Christopher Drost writes:
> Are there *any* security claims for the CubeHash SHA-3-normal proposal?

The 2.B.5 documentation states a preimage attack on CubeHash16/32-512 using roughly 2^384 bit operations. The formal security claims in 2.B.4 consistently state that quantum computers chop the preimage security exponent in half: 224->112, 256->128, 384->192, 512->256.

Given this background I don't understand how anyone can think that a 2^192 quantum attack against CubeHash16/32-512 is breaking anything.
I've made various other security statements about CubeHash, and I stand by all of them.
Gaetan claims that the attack contradicts "previous security statements of the designer"; this claim is simply not true.

I understand your complaint that CubeHash16/32-512 isn't explicitly covered in 2.B.4. I think it's reasonable to ask for one central source of all security statements. I've already been working on a "CubeHash parameter tweak: 10x smaller MAC overhead" document that has additional security claims (for 32-round finalization and a prefix MAC in place of HMAC), and that's a natural spot to include a comprehensive security table covering every specified CubeHash option.

Thomas Bonik writes:
> Why NIST asked for ***insane security level*** ?

NIST already had a SHA-512 standard reaching this 2^512 security level (with limitations: preimage, pre-quantum, short-message, single-target) and naturally required that SHA-3 also have an option reaching this security level. All of the SHA-3 candidates have such options.

But should we prioritize speed at this insane security level if that means punishing normal users? There's a fundamental tradeoff between

* making SHA-3 _fast_ at an insane 2^512 security level,
* allowing _small_ SHA-3 hardware, microcontrollers, etc., and
* avoiding SHA-3 interoperability problems.

Here are three examples of how design teams made different decisions regarding this tradeoff:

* Keccak avoids interoperability problems and loses only about a factor of 2 in speed at an insane 2^512 security level, but it can't fit into small hardware.

* Luffa-512 isn't even twice as slow as Luffa-256, and Luffa-256 can fit into relatively small hardware, but this hardware won't be able to understand Luffa-512 and will be a nightmare to upgrade. See http://cr.yp.to/papers.html#interop for further discussion.

* All CubeHash options fit into one very small circuit and avoid interoperability problems, but CubeHash loses a factor of 32 in speed if it's pushed to an insane 2^512 security level.

NIST hasn't said how these tradeoffs will be evaluated. Naturally, my view is that CubeHash makes the best tradeoff, because I think that having SHA-3 be small is important to many, many, many more users than having SHA-3 be fast at an insane security level.

> Pay attention to the part: "any result that shows that the candidate
> algorithm does not meet these requirements will be considered to be a
> serious attack" and compare it with the second-preimage security of
> CubeHash16/32-512. FULL STOP!

CubeHash16/1-512 meets those requirements. What you're missing is that the call for submissions also allowed a "tunable security parameter" to "allow the selection of a range of possible security/performance tradeoffs." Here are two examples of these tradeoffs:

* Keccak has two different 512-bit options, Keccak[]-512 (limited to 2^288 preimage security) and Keccak[c=1024] (slower).

* CubeHash has two different 512-bit options, CubeHash16/32-512 (limited to 2^384 preimage security) and CubeHash16/1-512 (much slower, although still very small).

Complaining that Keccak[]-512 and CubeHash16/32-512 don't reach 2^512 security is like complaining that Skein-256 and Shabal-256 don't reach
2^512 security: completely missing the point. Every SHA-3 submission has different options at different security levels. The real question is which collection of options will best serve the needs of the users.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Marsh Ray [marsh@extendedsubset.com]
Sent: Wednesday, October 06, 2010 1:15 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On 10/06/2010 09:20 AM, D. J. Bernstein wrote:
>
> Given this background I don't understand how anyone can think that a
> 2^192 quantum attack against CubeHash16/32-512 is breaking anything.

It probably has to do with:

    192 < 256 < 512

To get past that, one has to consider the background carefully and often involve a lot of discussion. This is certainly reasonable to expect in a deep analysis of hash functions. But the amount of discussion involved in sorting it out in this forum, it raises the question of users being expected to learn the subtleties.

It may be that an engineer proposing a new design chooses the faster 512-bit function. Then the customer rejects it on the basis that it "only provides 384 bits of security". They may have to go through this whole conversation over again, except with a customer who is just checking off requirements mechanically and cannot consider the subtleties involved.

Presenting multiple options often makes things harder.

I suspect the choice between "more expensive/more secure" and "cheaper/less secure" would end up getting made for all kinds of reasons other than any actual benefit gained by an additional 128 bits of preimage resistance.

For example http://en.wikipedia.org/wiki/SHA-2 says that
"SHA-512 is part of a system to authenticate archival video from the International Criminal Tribunal of the Rwandan genocide."
Not to say that genocide should be taken lightly, but it's unlikely
SHA-512 was chosen because it was considered plausible that the system's adversary would have 256-bit preimage capabilities.

> HMAC), and that's a natural spot to include a comprehensive security
> table covering every specified CubeHash option.

That might be helpful. Tables are nice.

> But should we prioritize speed at this insane security level if that
> means punishing normal users? There's a fundamental tradeoff between
>
>   * making SHA-3 _fast_ at an insane 2^512 security level,
>   * allowing _small_ SHA-3 hardware, microcontrollers, etc., and
>   * avoiding SHA-3 interoperability problems.

What kind of applications are there for which 512 bits is not an inherently "insane" length to use for a hash?

Would it not be even more inefficient to transmit and store such long values?

Perhaps if one were making a PRNG or stream cipher, you might end up using as many bits as were produced. But obviously it couldn't expect unbounded preimage resistance out of the fixed-size primitives from which it was constructed.

Perhaps the market for 512 bit hashes at a non-insane security level is very small indeed.

> having SHA-3 be small is important to many, many, many more users than
> having SHA-3 be fast at an insane security level.

+1 from this user.

- Marsh

From: hash-forum@nist.gov on behalf of Thomas Bonik [bonik1977@googlemail.com]
Sent: Wednesday, October 06, 2010 6:13 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
D. J. Bernstein <djb@cr.yp.to> wrote:
> But should we prioritize speed at this insane security level if that
> means punishing normal users? There's a fundamental tradeoff between
>
>   * making SHA-3 _fast_ at an insane 2^512 security level,
>   * allowing _small_ SHA-3 hardware, microcontrollers, etc., and
>   * avoiding SHA-3 interoperability problems.
>
> <snip>
>
> NIST hasn't said how these tradeoffs will be evaluated. Naturally, my
> view is that CubeHash makes the best tradeoff, because *** I think
> that having SHA-3 be small is important to many, many, many more users
> than having SHA-3 be fast at an insane security level.***

Wrong!

The best demand of your claims came from John Kelsey's slides from which we learned that
NIST is planning to define SHA-512 choped to 256 bits. There is a BIG REASON for that: the
main and heavy use of hash functions is in more than 230 million public web severs in the
world where SPEED MATTERS, and in modern 64-bit servers with at least 45nm production
technology, SHA-512 is really fast. That is recognized and valued very much from NIST and
probably by SHA-2 designers. They are not worried about the "issue" of putting SHA-512 (or
SHA-512-256) in small hardware or about interoperability problems simply because it is a
non-issue nowadays, and will be even less important with the future semiconductor
manufacturing processes:

# 90 nm in 2002

# 65 nm in 2006

# 45 nm in 2008

# 32 nm in 2010

# 22 nm approx. 2011

# 16 nm approx. 2013

# 11 nm approx. 2015

that will be capable to squeeze SHA-256 and SHA-512 and SHA-512-256 and SHA-3-256 and
SHA-3-512 and many other hash functions ALL TOGETHER in microns. But being safe, fast and
energy efficient is an issue and will always be an issue.


>> Thomas Bonik wrote:
>> Pay attention to the part: "any result that shows that the candidate
>> algorithm does not meet these requirements will be considered to be a
>> serious attack" and compare it with the second-preimage security of
>> CubeHash16/32-512. FULL STOP!
>
```

> CubeHash16/1-512 meets those requirements. What you're missing is that
> the call for submissions also allowed a "tunable security parameter"
> to "allow the selection of a range of possible security/performance
> tradeoffs."

CubeHash16/1-512 - meets security requirements.
CubeHash16/32-512 - NO!

CubeHash16/1-512 is your formal proposal. IT IS SLOW and does not meet efficiency requirements.

Moreover, following Gaetan's previous post, I figured out that your Second Round submission package is not complete since you did not provided test values *** for the formal proposal.*** In the package you say - /* For "formal" instead of "normal" change BLOCKBYTES to 1. */ - admitting that test values for the formal proposal are missing in the official submission package. This is ***INCOMPLETE submission***, and I understand why you are doing this:
you are trying to ride on the security of the "formal" proposal, in an effort to compete with other competitors on the ground of efficiency with informal proposal.
I have already made my mind about your play in this competition, but what is worrying me is that NIST decided to put on their web server that incomplete package.

And, NO, I am not missing the "tunable security parameter" part, but I am referring to the order of the importance in the evaluation criteria that NIST has put in their call: 4. A Security, 4. B Cost, ,and in part 4. C there is a part talking about tunable security parameters.
So, your "formal" proposal drops out of the competition in part 4. B Cost (i. Efficiency), your informal attempt drops out of the competition in part 4. A Security. Nothing has left out of your CubeHashr/b for the part 4. C.

====
Thomas Bonik

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Friday, October 08, 2010 6:51 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
D. J. Bernstein wrote on 06 Oct 2010 16:20:29 +0200:

>> Are there *any* security claims for the CubeHash SHA-3-normal proposal?
>
> The 2.B.5 documentation states a preimage attack on CubeHash16/32-512
> using roughly 2^384 bit operations.

I'm sorry, Dan, I'm afraid I can't let you do this.

Three people are asking for a formal security claim, and you are just referring to a part
of the documentation that you explicitly rejected as not being a security claim a few
months ago:

  The "Symmetric states" paper takes the 384-bit example completely out
  of context, and makes the reader falsely believe that (e.g.) 2^320
  operations or 2^256 operations would qualify, when in fact the
  CubeHash submission document says that they _can't_ qualify.

  -- message <20100725232008.7268.qmail@cr.yp.to>, 25 Jul 2010

Can you please stop wasting everybody's valuable time with your misleading statements and
make a formal security claim?

Seriously, how hard can it be to make a formal security claim similar to the one in 2.B.4?
I asked you for such a claim more than two months ago...

I can think a few reasons why you would refuse to make a claim, but none of them seem to
be worthy of an honest cryptographer like you.  You wouldn't submit a function without a
clear idea of its security, and wait until the end of the round to make a security claim
that could include any attacks found in between, would you?



However, I like the new way of reading the security claims that you now
promote:

> The formal security claims in 2.B.4 consistently state that quantum
> computers chop the preimage security exponent in half: 224->112,
> 256->128, 384->192, 512->256.
> Given this background I don't understand how anyone can think that a
> 2^192 quantum attack against CubeHash16/32-512 is breaking anything.

The formal claims in 2.B.4 consistently state that collision resistance is half of
preimage resistance: 224->112, 256->128, 384->192, 512->256.
Therefore, I guess that CubeHash-normal-512 has a collision resistance of less than 2^192
-- or maybe 2^128.  In fact, if I follow this logic, you are claiming that CubeHash-
normal-512 offers no more security than a 384-bit hash function -- or maybe a 256-bit hash
function.

Is this your new security claim?  It's a pretty stupid claim, and it contradicts most of
your previous statements, but at least it would be a claim, and we could start discussing
CubeHash-normal security.

And please don't tell me I misunderstand you when you deliberately avoid to give a clear
```

statement.

> I've made various other security statements about CubeHash, and I
> stand by all of them. Gaetan claims that the attack contradicts
> "previous security statements of the designer"; this claim is simply not true.

If you do read section 1.2 of my note, you will see that I don't just claim that this property contradicts some statements; I give precise references to such statements, both in the submission package, and on the hash forum. For instance, I would really love to know how the following statement does not cover quantum attacks:

  CubeHash16/32--512 is my main proposal, providing 2^256 security
  against all attacks.

  -- message <20100822232203.27042.qmail@cr.yp.to>, 22 Aug 2010

I shall just recall the context surrounding this message: most of your posts in this thread discuss quantum attacks (including this very one), and you oppose "all attacks" to "pre-quantum" a few messages earlier (message <20100814021923.7676.qmail@cr.yp.to>, 13 Aug 2010).

In fact your various statements made a quite reasonable picture until last week: CubeHash-normal-512 was supposed to be as good as SHA-512 in a post-quantum world (but of course you never formalized it into a security claim). Some people did not like that picture, because it does not fulfill NIST's requirements, but at least it was coherent and made sense.

> Complaining that Keccak[]-512 and CubeHash16/32-512 don't reach 2^512
> security is like complaining that Skein-256 and Shabal-256 don't reach
> 2^512 security: completely missing the point.

Please don't mix apples and oranges. Keccak[]-512 has a very precise security claim, and is not a SHA-3 candidate.

Regarding the security of CubeHash16/32-512, as long as you don't make a formal security claim, we cannot miss the point because there is no point. Without a formal security claim, and given that your security statements are contradictory, misleading, and/or very easy to misunderstand, CubeHash16/32-512 can only be used in systems where
CRC-32 would be an acceptable hash function.

--
Gaëtan Leurent

From: hash-forum@nist.gov on behalf of Charles Bouillaguet [charles.bouillaguet@ens.fr]
Sent: Friday, October 08, 2010 8:11 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

After having read Dan and the rest of the world arguing, I decided that I also had to have an opinion on this matter. The first step was to read
(again) the CubeHash submission document. In this message I summarize what I have found.

On the one hand, section 2.B.4 of the Cubehash submission document (that I read at : http://cubehash.cr.yp.to/submission/strength.pdf) tells use that "Cubehash-512 is expected to provide preimage resistance of approximately 512 bits.". Seems reasonable.

On the other hand, section 2.B.5 of the Cubehash submission document (that I read at http://cubehash.cr.yp.to/submission/attacks.pdf) tells use that :

"However, the attacker loses control over the state as b decreases. Generic attacks have success probability dropping exponentially with the pipe size 1024 - 8b. For example, CubeHash r/32 has a 768-bit pipe, putting generic attacks (including advanced attacks such as herding) far out of reach."

So, I am lead to think that "CubeHash16/32-512" is susceptible to a
2^384 meet-in-the-middle preimage attack, because of its invertible round function. This is indeed confirmed by Dan's document on generic attacks
(http://cubehash.cr.yp.to/submission/generic.pdf). Therefore, I am inclined to say that CubeHash-normal (which is apparently the "normal" submission for "normal" people) fails to meet the requirement of section 2.B.4 of the documentation (which does not specify which CubeHash version it is addressing -- what Gaëtan is complaining about). This seems a bit weird.

But then (and even worse !), on this forum, I could read:

CubeHash16/32--512 is my main proposal, providing 2^256 security against all attacks.
 -- message <20100822232203.27042.qmail@cr.yp.to>, 22 Aug 2010

I am now, apparently just like everybody, at a complete loss ! I first thought that CubeHash-512 was supposed to provide 512 bits of preimage resistance [256 with quantum computers], just to learn a bit after that it in fact offers 384 bits of security [192 with quantum computers]. Then you claim that it offers 256 bits of security, but since this is incompatible with the 2^192 quantum preimage attack, it must mean that CubeHash-512 offers 256 bits of classical security.

There are, as far as I understand, (at least) 3 competing statements.

Please Dan, help me ! I'm going crazy figuring this damn thing out !

Hopefully, I found a way out. I recently borrowed this philosophy book at the local library, and I learned about Hegel's law of contradictions:
the law that everything is contradictory.

So, OK, life itself is contradictory. But Hash functions security statements don't have to....

--
Charles Bouillaguet
http://www.di.ens.fr/~bouillaguet/

PS : My apologies for the bad irony, but the situations really calls for it

```
D. J. Bernstein wrote on 06 Oct 2010 16:20:29 +0200:

> Christopher Drost writes:
>> Are there *any* security claims for the CubeHash SHA-3-normal proposal?
>
> The 2.B.5 documentation states a preimage attack on CubeHash16/32-512
> using roughly 2^384 bit operations. The formal security claims in 2.B.4
> consistently state that quantum computers chop the preimage security
> exponent in half: 224->112, 256->128, 384->192, 512->256.
>
> Given this background I don't understand how anyone can think that a
> 2^192 quantum attack against CubeHash16/32-512 is breaking anything.
> I've made various other security statements about CubeHash, and I stand
> by all of them. Gaetan claims that the attack contradicts "previous
> security statements of the designer"; this claim is simply not true.
>
> I understand your complaint that CubeHash16/32-512 isn't explicitly
> covered in 2.B.4. I think it's reasonable to ask for one central source
> of all security statements. I've already been working on a "CubeHash
> parameter tweak: 10x smaller MAC overhead" document that has additional
> security claims (for 32-round finalization and a prefix MAC in place of
> HMAC), and that's a natural spot to include a comprehensive security
> table covering every specified CubeHash option.
>
> Thomas Bonik writes:
>> Why NIST asked for ***insane security level*** ?
>
> NIST already had a SHA-512 standard reaching this 2^512 security level
> (with limitations: preimage, pre-quantum, short-message, single-target)
> and naturally required that SHA-3 also have an option reaching this
> security level. All of the SHA-3 candidates have such options.
>
> But should we prioritize speed at this insane security level if that
> means punishing normal users? There's a fundamental tradeoff between
>
>     * making SHA-3 _fast_ at an insane 2^512 security level,
>     * allowing _small_ SHA-3 hardware, microcontrollers, etc., and
>     * avoiding SHA-3 interoperability problems.
>
> Here are three examples of how design teams made different decisions
> regarding this tradeoff:
>
>     * Keccak avoids interoperability problems and loses only about a
>       factor of 2 in speed at an insane 2^512 security level, but it
>       can't fit into small hardware.
>
>     * Luffa-512 isn't even twice as slow as Luffa-256, and Luffa-256 can
>       fit into relatively small hardware, but this hardware won't be able
>       to understand Luffa-512 and will be a nightmare to upgrade. See
>       http://cr.yp.to/papers.html#interop for further discussion.
>
>     * All CubeHash options fit into one very small circuit and avoid
>       interoperability problems, but CubeHash loses a factor of 32 in
>       speed if it's pushed to an insane 2^512 security level.
>
> NIST hasn't said how these tradeoffs will be evaluated. Naturally, my
> view is that CubeHash makes the best tradeoff, because I think that
> having SHA-3 be small is important to many, many, many more users than
> having SHA-3 be fast at an insane security level.
>
>> Pay attention to the part: "any result that shows that the candidate
>> algorithm does not meet these requirements will be considered to be a
>> serious attack" and compare it with the second-preimage security of
>> CubeHash16/32-512. FULL STOP!
>
```

```
> CubeHash16/1-512 meets those requirements. What you're missing is that
> the call for submissions also allowed a "tunable security parameter" to
> "allow the selection of a range of possible security/performance
> tradeoffs." Here are two examples of these tradeoffs:
>
>    * Keccak has two different 512-bit options, Keccak[]-512 (limited to
>      2^288 preimage security) and Keccak[c=1024] (slower).
>
>    * CubeHash has two different 512-bit options, CubeHash16/32-512
>      (limited to 2^384 preimage security) and CubeHash16/1-512 (much
>      slower, although still very small).
>
> Complaining that Keccak[]-512 and CubeHash16/32-512 don't reach 2^512
> security is like complaining that Skein-256 and Shabal-256 don't reach
> 2^512 security: completely missing the point. Every SHA-3 submission has
> different options at different security levels. The real question is
> which collection of options will best serve the needs of the users.
>
> ---D. J. Bernstein
>    Research Professor, Computer Science, University of Illinois at Chicago
>
```

From: hash-forum@nist.gov on behalf of Thomas Bonik [bonik1977@googlemail.com]
Sent: Friday, October 08, 2010 9:16 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Yess.

I bought two more big popcorn boxes here at the theater gallery of the
SHA-3 competition ;-) ,
and I am carefully watching the interesting developments and discussions.

My two favor highlights from recent posts are:

2010/10/8 Gaëtan Leurent <gaetan.leurent@ens.fr> wrote:

> Without a formal security claim, and given that your security
> statements are contradictory, misleading, and/or very easy to
> misunderstand, CubeHash16/32-512 can only be used in systems where
> CRC-32 would be an acceptable hash function.

and

On Fri, Oct 8, 2010 at 12:09 PM, Charles Bouillaguet <charles.bouillaguet@ens.fr> wrote:

> Hopefully, I found a way out. I recently borrowed this philosophy book
> at the local library, and I learned about Hegel's law of contradictions:
> the law that everything is contradictory.
>

There is another simpler way out of this endless debate about SHA-3 candidate CubeHash:
NIST will have opportunity to fix their mistake letting CubeHash in the Second Round and
in Round 3, there will be no such a contradictory candidate as CubeHash.

====
Thomas Bonik

From: hash-forum@nist.gov on behalf of Shawn Collenburg [greynite@gmail.com]
Sent: Friday, October 08, 2010 10:36 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

> Can you please stop wasting everybody's valuable time with your
> misleading statements and make a formal security claim?

NIST created the soapbox, and while Dan has taken every opportunity to step upon it, the decision to spend your time to listen is wholly upon the individual listener.

In the beginning I myself found Dan's bickering aggravating, however at this point I agree with Thomas Bonik -- grab some popcorn & enjoy the show.

Dan plays a dangerous game. His inclination to bicker and incite has motivated many to attack CubeHash with all their might, and in doing so gracing CubeHash with the honor of being the most [or close to it] analyzed of the submissions, and therefore earning CubeHash high marks in one of NIST's core criteria:

> v. Other Consideration Factors
> In addition to the evaluation factors
> mentioned above, the quality of the
> security arguments/proofs

While simultaneously risking the confidence of the cryptographic community. Again from FR Nov07:

> v. Other Consideration Factors
> [...] and the confidence of NIST and the
> cryptographic community in the
> algorithm’s long-term security

It boils down to this then:

* How confident are you in CubeHash at this point, judged on
* it's merits, leaving aside any personal feelings toward Dan?

The rub being that the "merits" include not just cryptanalysis, but also the sizable ego of the submitter, and the valid concern that such an ego has blinded the submitter to flaws in their own submission.

Regards,
Shawn

From: hash-forum@nist.gov on behalf of Marsh Ray [marsh@extendedsubset.com]
Sent: Friday, October 08, 2010 1:30 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
On 10/08/2010 09:35 AM, Shawn Collenburg wrote:
>
>  grab some popcorn &
> enjoy the show.

Mmm popcorn.

> Dan plays a dangerous game. His inclination to bicker and incite has
> motivated many to attack CubeHash with all their might, and in doing
> so gracing CubeHash with the honor of being the most [or close to it]
> analyzed of the submissions

There may be another factor at work as well.

There are multiple submissions based on this sponge design, which is relatively new. I
haven't yet dug into all of them at this point, but CubeHash looks like it may be the
purest embodiment of the design. It has an exceedingly simple description.

While simplicity is usually a good thing, it ends up having a lot of invertibility and
symmetry. For example, there appears to be no particular resistance to meet-in-the-middle
attacks (it lacks even a block counter). It seems to derive all of its security from
having a generous amount of internal state, 768 or 1024 bits depending on how you look at
it.

I'm not saying that this is a good or bad way to build a hash function, just that it looks
to me like it approximates the Platonic ideal sponge closer than others.

So when "observations" are made concerning sponge functions in general, they're likely to
affect CubeHash directly. Also, CubeHash makes an obvious first choice when you want to
think about how such an observation affects a concrete design.

If, say, it turns out that "meet-in-the-middle" generalizes into some kind of "meet-in-
the-(2j + 1)/(2^i)", we might expect it to hit CubeHash at least as hard as any of the
other sponges (hmm, can you hit a sponge hard?), relative to the basic size parameters.

>, and therefore
> earning CubeHash high marks in one of NIST's core criteria:
>
>     v. Other Consideration Factors
>     In addition to the evaluation factors
>     mentioned above, the quality of the
>     security arguments/proofs

How many security arguments are likely to apply in support of CubeHash that wouldn't also
support the other sponges. Conversely, how many weaknesses wouldn't also affect CubeHash?

> * How confident are you in CubeHash at this point, judged on
> * it's merits, leaving aside any personal feelings toward Dan?
>
> The rub being that the "merits" include not just cryptanalysis, but
> also the sizable ego of the submitter, and the valid concern that such
> an ego has blinded the submitter to flaws in their own submission.

I don't think we need to invoke any deep psychology here, but it's maybe a little unfair
to expect submitters to be completely objective and unbiased. It's not often that the Math
```

club gets to be cheerleaders (hmm wait a minute...)

Whether he's optimally diplomatic is another question, but DJB is mostly talking about his design and arguing for claims made about it. My guess is that the popcorn will eventually run out on that discussion without anyone feeling satisfied.

But the winner probably won't be the submitter who argued the most that their function isn't broken. If anyone gets tired of talking about CubeHash claims, all they need to do to change the subject is to come up with an attack on anything having a lower exponent.

- Marsh

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]

Sent: Friday, October 08, 2010 2:50 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
> Re: OFFICIAL COMMENT: CubeHash (Round 2)
>
> Marsh Ray
>
> to:
>
> hash-forum@nist.gov
>
> On 10/08/2010 09:35 AM, Shawn Collenburg wrote:

>
> There are multiple submissions based on this sponge design, which is
> relatively new. I haven't yet dug into all of them at this point, but
> CubeHash looks like it may be the purest embodiment of the design. It
> has an exceedingly simple description.
```

Let me add some input on this other aspect about "sponge designs", which is that they are new creatures, and hence less understood.
Sponge designs, or sponge-like designs such as Fugue, should be viewed as an input stage and a final compression stage. If the input stage is proven collision resistant, and the final compression stage is a random oracle, one should not care if it is a sponge or not -- the sponge part is all in the input stage.

This methodology is much easier to design and understand, than building a Merkle Damgard style design, for which you need additional Merkle Damgard mode theorem to convince yourself that you got a good hash function (a definition of a good hash function is that it is a variable input length random oracle). Of course, the extra burden in the new methodology is to build a sponge based collision resistant function.

So, to summarize:
1) Convince yourself that the sponge part is just collision resistant.
2) Convince yourself that the final compression is a Random oracle, which you anyway do in all methodologies, but here you do not need to worry about arbitrary long inputs, but just this one, let's say, 1024 bit to 256 bit compression function.

Charanjit Jutla

```
>
> While simplicity is usually a good thing, it ends up having a lot of
```

From: hash-forum@nist.gov on behalf of Marsh Ray [marsh@extendedsubset.com]
Sent: Friday, October 08, 2010 3:54 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On 10/08/2010 01:50 PM, Charanjit Jutla wrote:
>
> So, to summarize:
> 1) Convince yourself that the sponge part is just collision resistant.

I'll have to get back to you on that. :-)

> 2) Convince yourself that the final compression is a Random oracle,
> which you anyway do in all methodologies, but here you do not need to
> worry about arbitrary long inputs, but just this one, let's say, 1024
> bit to 256 bit compression function.

Please correct me if I'm wrong, but isn't the final compression in CubeHash (and it looks like in Fugue too) trivially invertible?
Moreover, the final truncation appears to enable one to find a nearly-arbitrary number of penultimate states (pseudopreimages?) which result in the target hash value.

This doesn't sound very random oracleish to me.

Thanks for helping me understand this fascinating topic.

- Marsh

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]

Sent: Friday, October 08, 2010 4:59 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Marsh Ray <marsh@extendedsubset.com> wrote on 10/08/2010 03:41:23 PM:

> On 10/08/2010 01:50 PM, Charanjit Jutla wrote:
> >
> > So, to summarize:
> > 1) Convince yourself that the sponge part is just collision resistant.
>
> I'll have to get back to you on that. :-)
>
> > 2) Convince yourself that the final compression is a Random oracle,
> > which you anyway do in all methodologies, but here you do not need to
> > worry about arbitrary long inputs, but just this one, let's say, 1024
> > bit to 256 bit compression function.
>
> Please correct me if I'm wrong, but isn't the final compression in
> CubeHash (and it looks like in Fugue too) trivially invertible?
> Moreover, the final truncation appears to enable one to find a
> nearly-arbitrary number of penultimate states (pseudopreimages?) which
> result in the target hash value.

No the final compression is not invertible! The final compression is same
as all compression functions. It takes 1024 bits and produces 256 bits.
You do not get to see the other 1024-256 bits after the application of the
final round.

This is same as in SHA-1 compression function. It takes 512 bits and produces
160 bits. That function is also invertible if you do not have the SHA-1 final feed
forward. The SHA-1 feed forward is there because it prevents meet in the middle
attacks in its Merkler-Damgard mode. If you follow the methodology I stated,
you do not have to worry about those meet-in the middle attacks. There is no
meet in the middle attack on constant input length Random Oracles...i.e. nuance
only holds for variable length Random oracles which you try to build using Merkle
Damgard.

Hope this helps.

Charanjit Jutla

>
> This doesn't sound very random oracleish to me.
>
> Thanks for helping me understand this fascinating topic.
>
> - Marsh

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]

Sent: Friday, October 08, 2010 5:00 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Ah, ok, I forgot to add that the random oracle must be fed with inputs generated by the compression function...yes you are right there.

Charanjit

```
Marsh Ray <marsh@extendedsubset.com> wrote on 10/08/2010 03:41:23 PM:

> [image removed]
>
> Re: OFFICIAL COMMENT: CubeHash (Round 2)
>
> Marsh Ray
>
> to:
>
> hash-forum
>
> 10/08/2010 03:42 PM
>
> Cc:
>
> Charanjit Jutla
>
> On 10/08/2010 01:50 PM, Charanjit Jutla wrote:
> >
> > So, to summarize:
> > 1) Convince yourself that the sponge part is just collision resistant.
>
> I'll have to get back to you on that. :-)
>
> > 2) Convince yourself that the final compression is a Random oracle,
> > which you anyway do in all methodologies, but here you do not need to
> > worry about arbitrary long inputs, but just this one, let's say, 1024
> > bit to 256 bit compression function.
>
> Please correct me if I'm wrong, but isn't the final compression in
> CubeHash (and it looks like in Fugue too) trivially invertible?
> Moreover, the final truncation appears to enable one to find a
> nearly-arbitrary number of penultimate states (pseudopreimages?) which
> result in the target hash value.
>
> This doesn't sound very random oracleish to me.
>
> Thanks for helping me understand this fascinating topic.
>
> - Marsh
```

From: hash-forum@nist.gov on behalf of Charanjit Jutla [csjutla@us.ibm.com]

Sent: Friday, October 08, 2010 10:28 PM

To: Multiple recipients of list

Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Marsh, you brought up a good point there about final round being a permutation.
However, as you well know most random oracles are built out of components
which are permutations (and then applying a feed forward or as in the case of
sponge not even a feed forward).

so, let me rephrase my earlier claim, this time using a random permutation
as the final round. This time I claim that you
1) Convince yourself that the sponge part (by this I mean the variable length input absorption part) is collision resistant, AS WELL AS is a one-way function,
2) convince yourself that the final transformation is a random permutation.

then you have a variable input length Random oracle.

So, how does one formally prove this?

Before we get into formalizing this, let me remark that
the indifferentiability result of Bertoni et al about Sponge constructions
claims that if all input rounds are random permutations , and the final round is a
random permutation, and the input rounds do not yield internal collisions then this gives you a variable input length random oracle.

What I am claiming above is a much stronger result in some way, as it requires the input round (though as a monolithic variable input length entity) to be just collision resistant and oneway. These are much easier to attain (or even prove under reasonable assumptions) than proving each input round to be random (unless you relinquish efficiency and just keep churning in each input round).

Ok, coming back to the claim, let's define some ideal functionalilties.

A Random Oracle (RO) ideal functionality keeps a table T.
On any input, it checks table T to see if it has been previously called upon,
and if so returns the previously returned value. Otherwise, it returns a random
value of 256 bits, and saves this entry in T.

The RO can be defined as VIL-RO or FIL-RO. The latter takes inputs of only 1024 bits or less, and the former takes arbitrarily long inputs.

A compression function (VIL-CF) ideal functionality keeps a table T.
On any input, it calls the adversary with the input, and let's it decide the output
as long as it is not colliding with any output it has previously made.

The one-way function ideal functionality is tricky, as the usually understood definition is that on a randomly chosen 256 bit output value, no adversary
can come up with an input which would yield that value. Defining ideal functionality with this "randomly chosen"

notion is a bummer.

However, define the following ideal functionality called
(VIL-OW-FIL-RO) where it has two functions which can be called.
The first function is called "Generate Random", and the second
is called "Generate Oneway". So, it maintains two tables: R and T.
- On the first type of input of length only 1024 bit or less,
it behaves like a FIL-RO, that is returns a random value on new inputs,
and saves the value output in R.
- On the second type of input of variable input length, it calls the
adversary and let's the adversary decide the output as long as it
is not in table R.

Thus, the first function is just a FIL-RO, and the second is linked with the
first, and thats how it models a oneway function. Essentially, imagine how
the oneway function security game is played: the adversary is given a random
value and asked to invert it. Where did that random value come from? It must
have come from some random oracle, so we built it into the functionality.

So, now we are ready to define a functionality which models the "new world sponge". This ideal functionality is called (VIL-CROW-FIL-RP) ...RP stands
for random permutation. This has three functions called
invert-final, input-sponge, and finalize. It also has three table, S, R, O.

Invert-final:
- on the first type of input x of length only 1024 bit or less it behaves like a FIL-RO
but this time generates 1024 bit random value s (s for internal state) and saves it as R[x] = s, as well as O[s] =x...this models the invertion function of the final round random permutation.

finalize:
- a finalize function takes 1024 bit input s (s for internal state), and ignores
the input if s was not set as internal state before, i.e. it is not in table S.
Else, if O[s] is defined it outputs that, else it picks a new 1024 bit random value x and outputs that. It saves O[s] = x, as well as R[x] =s.

input-sponge:
-on the third type of input P of variable input length, it calls the adversary
and lets it decide a value s (for internal state), as long as it is not already in S or in R. If not it saves S[P]=s. This makes it model both collision resistant and
oneway.

Now, it is easy to see that this is a random oracle when calls are made as follows: With input P, call input-sponge which returns, say s. Then call finalize
with s, and get back r.

It is easy to see that this is a VIL-RO: On the first call adversary gets to set s,
but it can never be set to something it set before as internal state (collision resistance) nor to some value one obtained by calling invert-final (onewayness).
Now, before it calls finalize on s, an adversary can make several invert-final
queries, but those will be random and unrelated to s. When the finalize call
is made on s, it will not be in S or R, and hence a random value will be output.

QED.

Hope this was productive :)

Charanjit Jutla

Now, notice its behaviour

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Saturday, October 09, 2010 8:23 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

To reduce the number of messages I'll send this one response to the parallel messages from two SIMD people. Let me again emphasize that all of the CubeHash security claims, both pre-quantum and post-quantum, are holding up just fine.

Charles Bouillaguet writes:
> I am now, apparently just like everybody, at a complete loss ! I first
> thought that CubeHash-512 was supposed to provide 512 bits of preimage
> resistance [256 with quantum computers], just to learn a bit after
> that it in fact offers 384 bits of security [192 with quantum computers].

There are two different 512-bit options: CubeHash16/1-512 and CubeHash16/32-512. Is this really so hard to understand? Are you really "at a complete loss"? Are you also perplexed by the presence of

* two 512-bit ECHO options,
* two 512-bit Fugue options (announced here as an upcoming tweak),
* two 512-bit Keccak options, and
* two 512-bit Skein options,

in each case with measurably different security levels? For example, are you confused by the fact that Keccak provides Keccak[]-512 with "only"
2^288 preimage security, and Keccak[c=1024]-512 with more? Do you seriously believe that everybody else has trouble understanding this?

There have been two incidents of other submission teams misrepresenting the CubeHash16/1-512 security claims as CubeHash16/32-512 security claims, and on this basis misrepresenting non-attacks as attacks. I think that this is as ridiculous as taking, e.g., your SIMD-512 security claims, misrepresenting them as SIMD-256 security claims, and on this basis claiming to have broken SIMD-256.

GaÃ«tan Leurent writes:
> For instance, I would really love to know how the following statement
> does not cover quantum attacks:
>   CubeHash16/32--512 is my main proposal, providing 2^256 security
>   against all attacks.

Gaetan, do you deny that your submission claims "complexity" 2^256 for every "preimage or second preimage attack" on SIMD-256? Do you deny that this is a claim of 2^256 security against all preimage attacks?

NEWS FLASH: I've discovered that I can find preimages in SIMD-256 using Grover's quantum algorithm with complexity only 2^128, far below your claimed 2^256. Do you deny that I've found a successful preimage attack against SIMD-256? Do you deny that your SIMD-256 security claim has been massively violated? When will we see your formal withdrawal of SIMD?

I haven't checked all the other submissions but there are certainly several that have claims of 2^256 preimage security for 256-bit output.
All of these claims are violated by quantum computers. Do you think we should run around claiming to have "attacked" those candidates too?

There's one submission whose security claims _do_ resist quantum
computers: namely, CubeHash. For the second round I added several explicit preimage-security claims against quantum computers, and for those claims I was careful to chop all exponents in half, precisely so that I _wouldn't_ be exaggerating the security of CubeHash

against quantum attacks. See http://cr.yp.to/papers.html#collisioncost to understand why this isn't necessary for collision attacks.

> Keccak[]-512 has a very precise security claim, and is not a SHA-3
> candidate.

Don't be obtuse. The Keccak SHA-3 submission defines Keccak[]-512, proposes it (along with other Keccak[] output lengths) as a "fifth candidate," and says that it is the Keccak "default." Practically all of the Keccak speed advertisements are for Keccak[]-512, not the higher-security Keccak[c=1024]-512.

> Seriously, how hard can it be to make a formal security claim similar
> to the one in 2.B.4? I asked you for such a claim more than two
> months ago...

Offhand I don't recall your request, but in any event I'm adding a comprehensive security table to my upcoming "CubeHash parameter tweak:
10x smaller MAC overhead" document.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of Charles Bouillaguet [charles.bouillaguet@ens.fr]
Sent: Monday, October 11, 2010 8:15 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
Dear Dan (and others),

D. J. Bernstein wrote on 10 Oct 2010 02:23:02 +0200:
> To reduce the number of messages I'll send this one response to the
> parallel messages from two SIMD people. Let me again emphasize that
> all of the CubeHash security claims, both pre-quantum and
> post-quantum, are holding up just fine.
>
> Charles Bouillaguet writes:
>> I am now, apparently just like everybody, at a complete loss ! I
>> first thought that CubeHash-512 was supposed to provide 512 bits of
>> preimage resistance [256 with quantum computers], just to learn a bit
>> after that it in fact offers 384 bits of security [192 with quantum computers].
>
> There are two different 512-bit options: CubeHash16/1-512 and
> CubeHash16/32-512. Is this really so hard to understand? Are you
> really "at a complete loss"? Are you also perplexed by the presence of
>
>     * two 512-bit ECHO options,
>     * two 512-bit Fugue options (announced here as an upcoming tweak),
>     * two 512-bit Keccak options, and
>     * two 512-bit Skein options,
>
> in each case with measurably different security levels? For example,
> are you confused by the fact that Keccak provides Keccak[]-512 with "only"
> 2^288 preimage security, and Keccak[c=1024]-512 with more? Do you
> seriously believe that everybody else has trouble understanding this?

I am not confused by that. I am confused by the fact that section 2.B.4 of your submission
document tells us about the security of "CubeHash-512", without any mention of the tunable
parameters. It would be great if you could be a teeny tiny bit more precise : you pointed
out that only a complete moron could think that the security claim of section 2.B.4
applies to CubeHash 1/127. I certainly agree with you. You seem to be promoting r=16, b=32
as your preferred compromise between speed and security. So why don't you tell us about
the security of this tradeoff?  Why don't you slightly update your formal security claims
to cover your favorite choice of r and b?

This is, I believe, a bit confusing. Some people would say it is also a bit dishonest, as
one could understand that section 2.B.4 discuss the security of the *DEFAULT* value of the
tunable parameters, or maybe of
*ALL* values, and it is in fact not the case. To summarize, my concern is that r and b do
not appear in section 2.B.4 of your documentation, except in the sentence : "See the
CubeHash specification for recommended parameters r, b", which leaves the obvious
question: to which (ranges
of) values of r and b does this apply?

Seriously, Dan, this is a honnest (i.e., non-rhetorical) question, the answer of which is
certainly not trivial. Some (weak) choices of (r,b) have been broken, and your choices are
supposedly in the safe zone. But where is the safe zone?

Please observe that the other team you are mentioning all discuss the security of their
preferred tunable parameter value in their submission document, either by giving explicit
bounds (e.g. Keccak[] reaches a security level of 2^288) or by claiming a security level
that depends on the tunable parameters (i.e., Keccak[c,r,d] is assumed to resist your
favorite attack up to f(c,r,d) evaluations of the permutation for some explicit function
```

f). If I read correctly, they all also explicitly say something like "you may change the tunable parameters according to your liking, but my own choice is ....". And they certainly discuss the security of their own choice.

*) while Keccak has tunable security parameters, there is a *NOMINAL* Keccak for each output size. The keccak team also suggests using Keccak[] and then truncating the output in all cases. This version is called "Keccak" on the ebash, if I'm not mistaken. The security of Keccak[], and of all the candidates (one for each output size) is explicitly stated. The submission documents discuss the security of the proposed parameters.

*) The Echo team has proposed a single-pipe version, but they consider the double-pipe to be their main submission (please correct me if I'm wrong). In the "Overwiew" section of their submission document they
say: "While we recommend the use of the double-pipe construction, ECHO easily supports a single-pipe construction while giving a significant performance improvement at the same time." In the "Security Claim"
section of their submission document, they only discuss the security of the double-pipe version.

*) The Skein team also promotes Skein-512 as their main sumbission :
their submisison documents tells us : "Skein-512—our primary proposal—....". They nevertheless explicitly provide distinct security claims for the 3 possible sizes of the internal state.

So far, section 2.B.4 of the CubeHash specification document
--seemingly-- does not cover CubeHash-512-normal.... It wouldn't hurt your submission to write down its actual security level. I believe this is what Gaëtan was also complaining about.

Best regards,
--
Charles Bouillaguet
http://www.di.ens.fr/~bouillaguet/

From: hash-forum@nist.gov on behalf of Elias Yarrkov [yarrkov@gmail.com]
Sent: Monday, October 11, 2010 4:47 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
On Wed, Oct 6, 2010 at 17:20, D. J. Bernstein <djb@cr.yp.to> wrote:
>
> Christopher Drost writes:
>> Are there *any* security claims for the CubeHash SHA-3-normal proposal?
>
> The 2.B.5 documentation states a preimage attack on CubeHash16/32-512
> using roughly 2^384 bit operations. The formal security claims in
> 2.B.4 consistently state that quantum computers chop the preimage
> security exponent in half: 224->112, 256->128, 384->192, 512->256.
>
> Given this background I don't understand how anyone can think that a
> 2^192 quantum attack against CubeHash16/32-512 is breaking anything.
> I've made various other security statements about CubeHash, and I
> stand by all of them. Gaetan claims that the attack contradicts
> "previous security statements of the designer"; this claim is simply not true.
>
> I understand your complaint that CubeHash16/32-512 isn't explicitly
> covered in 2.B.4. I think it's reasonable to ask for one central
> source of all security statements. I've already been working on a
> "CubeHash parameter tweak: 10x smaller MAC overhead" document that has
> additional security claims (for 32-round finalization and a prefix MAC
> in place of HMAC), and that's a natural spot to include a
> comprehensive security table covering every specified CubeHash option.
>
> Thomas Bonik writes:
>> Why NIST asked for ***insane security level*** ?
>
> NIST already had a SHA-512 standard reaching this 2^512 security level
> (with limitations: preimage, pre-quantum, short-message,
> single-target) and naturally required that SHA-3 also have an option
> reaching this security level. All of the SHA-3 candidates have such options.
>
> But should we prioritize speed at this insane security level if that
> means punishing normal users? There's a fundamental tradeoff between
>
>   * making SHA-3 _fast_ at an insane 2^512 security level,
>   * allowing _small_ SHA-3 hardware, microcontrollers, etc., and
>   * avoiding SHA-3 interoperability problems.
>
> Here are three examples of how design teams made different decisions
> regarding this tradeoff:
>
>   * Keccak avoids interoperability problems and loses only about a
>     factor of 2 in speed at an insane 2^512 security level, but it
>     can't fit into small hardware.
>
>   * Luffa-512 isn't even twice as slow as Luffa-256, and Luffa-256 can
>     fit into relatively small hardware, but this hardware won't be
> able
>     to understand Luffa-512 and will be a nightmare to upgrade. See
>     http://cr.yp.to/papers.html#interop for further discussion.
>
>   * All CubeHash options fit into one very small circuit and avoid
>     interoperability problems, but CubeHash loses a factor of 32 in
>     speed if it's pushed to an insane 2^512 security level.
```

>
> NIST hasn't said how these tradeoffs will be evaluated. Naturally, my
> view is that CubeHash makes the best tradeoff, because I think that
> having SHA-3 be small is important to many, many, many more users than
> having SHA-3 be fast at an insane security level.
>
>> Pay attention to the part: "any result that shows that the candidate
>> algorithm does not meet these requirements will be considered to be a
>> serious attack" and compare it with the second-preimage security of
>> CubeHash16/32-512. FULL STOP!
>
> CubeHash16/1-512 meets those requirements. What you're missing is that
> the call for submissions also allowed a "tunable security parameter"
> to "allow the selection of a range of possible security/performance
> tradeoffs." Here are two examples of these tradeoffs:
>
>   * Keccak has two different 512-bit options, Keccak[]-512 (limited to
>   2^288 preimage security) and Keccak[c=1024] (slower).
>
>   * CubeHash has two different 512-bit options, CubeHash16/32-512
>   (limited to 2^384 preimage security) and CubeHash16/1-512 (much
>   slower, although still very small).
>
> Complaining that Keccak[]-512 and CubeHash16/32-512 don't reach 2^512
> security is like complaining that Skein-256 and Shabal-256 don't reach
> 2^512 security: completely missing the point. Every SHA-3 submission
> has different options at different security levels. The real question
> is which collection of options will best serve the needs of the users.
>
> ---D. J. Bernstein
>   Research Professor, Computer Science, University of Illinois at
> Chicago
>
>

Alright. So, with current knowledge, CubeHash-normal-512 appears to provide
  * about 2^256 conventional collision resistance
  * about 2^384 conventional preimage resistance
  * about 2^192 quantum preimage resistance

For comparison, a secure (by normal standards) 480-bit hash is expected to provide
  * about 2^240 conventional collision resistance
  * about 2^480 conventional preimage resistance
  * about 2^240 quantum preimage resistance

(You've previously written that the generic 2^(n/3) quantum collision attack is costlier
than a conventional 2^(n/2) generic collision attack, which I can accept, so we'll ignore
that.)

The conventional preimage attacks are clearly less plausible than the quantum ones, but
this makes no difference in the comparison as
CubeHash-normal-512 loses to the 480-bit hash in both. The only advantage CubeHash has
seems to be conventional collision resistance.

Does this justify calling CubeHash-normal-512 a strong 512-bit hash?
I don't think so. Not from a theoretical standpoint, or in practice.

In the long term, it's a good bet the 2^192 quantum preimage attack will be cheaper to
perform than the 2^256 generic collision attack.
Previously you in fact have emphasized considering the potentially much lower cost of
quantum attacks, when compared to conventional attacks.

In the real world, it's not entirely implausible that a 2^240 conventional collision
attack on the 480-bit hash will always be easier to perform than the 2^192 quantum
preimage attack on CubeHash-normal-512, but I seriously would not count on this. If I were
to pick a hash based on real world security, I would certainly prefer the secure 480-bit

hash over CubeHash-normal-512.

Further, it seems one can in fact truncate even CubeHash-normal-512 output to some extent while retaining the same security against the most plausible attack. As such, part of the 512-bit output is simply a waste of space.

CubeHash-formal-512 fixes these problems. It's also 32 times slower. No other candidate has a secondary mode for actually performing as a standard cryptographic primitive that does something like 32 times the normal number of rounds. The "ultra-conservative" mode of Skein increases the number of rounds from 72 to 80, not to 2304.

Neither set of proposed parameters can replace SHA-2 without significant drawbacks. In some cases, neither can work as an acceptable replacement for SHA-512.

From: hash-forum@nist.gov on behalf of John Wilkinson [wilkjohn@gmail.com]
Sent: Monday, October 11, 2010 10:14 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On Mon, Oct 11, 2010 at 4:46 PM, Elias Yarrkov <yarrkov@gmail.com> wrote:
>
> Alright. So, with current knowledge, CubeHash-normal-512 appears to
> provide
> * about 2^256 conventional collision resistance
> * about 2^384 conventional preimage resistance
> * about 2^192 quantum preimage resistance
>

[snip]

>
> In the long term, it's a good bet the 2^192 quantum preimage attack
> will be cheaper to perform than the 2^256 generic collision attack.
> Previously you in fact have emphasized considering the potentially
> much lower cost of quantum attacks, when compared to conventional attacks.
>

[snip]

>
> CubeHash-formal-512 fixes these problems. It's also 32 times slower.
> No other candidate has a secondary mode for actually performing as a
> standard cryptographic primitive that does something like 32 times the
> normal number of rounds. The "ultra-conservative" mode of Skein
> increases the number of rounds from 72 to 80, not to 2304.
>
> Neither set of proposed parameters can replace SHA-2 without
> significant drawbacks. In some cases, neither can work as an
> acceptable replacement for SHA-512.
>

Suppose someone offered you a good 384-bit hash function, for which the least expensive attack required 2^192 effort, and for which the most effective cryptanalytic attack broke 4 of 16 rounds. Would you continue to use SHA-512 for its claimed 2^256 security, even if attackers had broken 40 of its 80 rounds with negligible (<< 2^64) effort? Even if the 384-bit hash was more efficient in software and hardware and had other practical advantages, such as resistance to length extensions and the ability to be used directly as a MAC? How about if it could be used to encrypt *and* MAC at the same time, with negligible overhead?

Are you really more concerned that an attacker will someday be able to mount a 2^192 attack, than you are about advances in cryptanalysis?
About the practical advantages of the new hash?

Just when do you think that a 2^192 effort attack will be mounted? If one assumes that in 2010, a 2^80 effort is infeasible, then, even if computing power continues to double every 18 months, well past the point of single-atom transistors, and beyond the point where the laws of thermodynamics make irreversible computing energy-prohibitive, it will still be the year 2178 (!!) before such an attack even becomes interesting to discuss.

We have yet to have a hash function remain unbroken for more than about a decade. Do we really think we'll do so much better with SHA-3 that we need worry about attacks that won't be feasible for more than sixteen decades?

If any of the SHA-3 candidates last 70 years, by which time 2^128 attacks may be imaginable, it will be a minor miracle. If SHA-3 isn't replaced by an SHA-4 better suited to the computing technology of 2080, I'll be surprised.

Are we really being productive arguing about attacks like this?
(That's a serious question.)

V/r,

John

From: hash-forum@nist.gov on behalf of Shawn Collenburg [greynite@gmail.com]
Sent: Tuesday, October 12, 2010 12:04 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On Mon, Oct 11, 2010 at 9:13 PM, John Wilkinson <wilkjohn@gmail.com> wrote:

> Are we really being productive arguing about attacks like this?
> (That's a serious question.)

And my serious answer is yes, very productive. There is a recent thread discussing the changeover & interoperability costs of moving to a SHA-(n). Considering these costs, if we can extend the useful lifetime of SHA-(n) by 1 year, we add that time to the amortization schedule, making for a decrease in these costs when considered on a "$/year" basis. So going from 10 years to 11 means a 9.1% decrease in cost.

Peak efficiency will be even worse, since there will be a ramp-up time for those efficiencies to percolate out. Consider the additional overhead of supporting SHA-(n) along with SHA-(n-(1+)). Until every node in a particular network can inter-operate via SHA-(n), all nodes must support additional algorithms, incurring both R&D overhead as well as additional unit costs & operating costs. This process takes a significant and fixed amount of time. Assuming it takes 5 years, you are left with only 5 years of peak efficiency, thus a +1 improvement gets you a 20% increase.

Cheers,
Shawn

From: hash-forum@nist.gov on behalf of Elias Yarrkov [yarrkov@gmail.com]
Sent: Tuesday, October 12, 2010 12:35 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On Tue, Oct 12, 2010 at 05:13, John Wilkinson <wilkjohn@gmail.com> wrote:
>
> Suppose someone offered you a good 384-bit hash function, for which
> the least expensive attack required 2^192 effort, and for which the
> most effective cryptanalytic attack broke 4 of 16 rounds. Would you
> continue to use SHA-512 for its claimed 2^256 security, even if
> attackers had broken 40 of its 80 rounds with negligible (<< 2^64)
> effort? Even if the 384-bit hash was more efficient in software and
> hardware and had other practical advantages, such as resistance to
> length extensions and the ability to be used directly as a MAC? How
> about if it could be used to encrypt *and* MAC at the same time, with
> negligible overhead?
>
> Are you really more concerned that an attacker will someday be able to
> mount a 2^192 attack, than you are about advances in cryptanalysis?
> About the practical advantages of the new hash?

Given sufficient additional analysis, I would certainly prefer such a function, and many SHA-3 candidates including CubeHash do provide such additional features. Many parties, though, need or "need" stronger security, including NIST. 512-bit hashes that do try to get the full 512-bit security (with limitations) are common at this point. Moving to an algorithm that definitely does not achieve it is not a pleasant move to make.

Are such 512-bit hashes useful, compared to shorter ones? Probably not.
Some old folk wisdom says that one should use a 2n-bit hash to get "n-bit security", but taking quantum attacks into consideration, the length equivalent to an n-bit symmetric key is more like n+c. 384 bits is in fact probably a choice pretty well in line with the intended security of algorithms like AES-256.

But CubeHash-normal-512 is not a 384-bit hash. One of the things NIST is looking for is a strong 512-bit hash. The performance penalty for using
CubeHash-formal-512 is not something I would consider acceptable.

> Just when do you think that a 2^192 effort attack will be mounted? If
> one assumes that in 2010, a 2^80 effort is infeasible, then, even if
> computing power continues to double every 18 months, well past the
> point of single-atom transistors, and beyond the point where the laws
> of thermodynamics make irreversible computing energy-prohibitive, it
> will still be the year 2178 (!!) before such an attack even becomes
> interesting to discuss.

2^192 is indeed not going to happen in the foreseeable future, quantum or conventional. 2^256 is even more unforeseeable. Neither is worth worrying about, but the advantage of CubeHash-normal-512 over
CubeHash-normal-384 is less than that.

> We have yet to have a hash function remain unbroken for more than
> about a decade. Do we really think we'll do so much better with SHA-3
> that we need worry about attacks that won't be feasible for more than
> sixteen decades?

Whichever algorithm becomes SHA-3, it will indeed probably be broken in practice by cryptanalytic techniques within the century. AES-256 will probably be broken in practice before a 2^128 quantum attack becomes feasible. I don't think that means we shouldn't even try to provide all the security the box we're designing the algorithm in allows.

> If any of the SHA-3 candidates last 70 years, by which time 2^128
> attacks may be imaginable, it will be a minor miracle. If SHA-3 isn't
> replaced by an SHA-4 better suited to the computing technology of
> 2080, I'll be surprised.
>
> Are we really being productive arguing about attacks like this?
> (That's a serious question.)
>

I think so.

From: hash-forum@nist.gov on behalf of Thomas Bonik [bonik1977@googlemail.com]
Sent: Tuesday, October 12, 2010 2:24 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On Tue, Oct 12, 2010 at 2:13 AM, John Wilkinson <wilkjohn@gmail.com> wrote:
>
> Suppose someone offered you a good 384-bit hash function, for which
> the least expensive attack required 2^192 effort, and for which the
> most effective cryptanalytic attack broke 4 of 16 rounds. Would you
> continue to use SHA-512 for its claimed 2^256 security, even if
> attackers had broken 40 of its 80 rounds with negligible (<< 2^64)
> effort?

You are discussing here as we are in a situation to chose between
SHA-512 and only one other alternative that "someone" (in this case Dan Bernstein) has
offered to us to use. But we (the info-security and crypto folks) are very lucky, because
we have 13 other offers that do not manifest weaknesses that CubeHash (that you are not
explicitly mentioning but you are defending it in your discussion) has.

> How
> about if it could be used to encrypt *and* MAC at the same time, with
> negligible overhead?
>

Ha, are you trying to make us "blind"? What basically you are advocating with this
statement is: "There are confirmed theoretical weaknesses in CubeHash but forget about
them because they are just theoretical, and look what CubeHash is offering you: it is
promising you that you can get some other crypto primitives with it." This is not
encryption and MAC competition. In the context of the hash competition, the value of this
"added values" for me is ZERO and I am very comfortable to use in all my software projects
well trusted and established encryption and MAC primitives and standards.

> Are you really more concerned that an attacker will someday be able to
> mount a 2^192 attack, than you are about advances in cryptanalysis?
> About the practical advantages of the new hash?
>

Here again you are discussing as CubeHas (which you are not explicitly mentioning but you
are defending it in your discussion) was our only alternative to SHA-512. Ohh, we are
really lucky that it is not the case.

> Just when do you think that a 2^192 effort attack will be mounted? If
> one assumes that in 2010, a 2^80 effort is infeasible, then, even if
> computing power continues to double every 18 months, well past the
> point of single-atom transistors, <snip>

Well, you have again forgot that other SHA-3 candidates are offering stronger security
without controversies, so your statement is perfectly true with 192 replaced by 256, and
CubeHash (that you are not explicitly mentioning but you are defending it in your
discussion) replaced by other 13 candidates.

> Are we really being productive arguing about attacks like this?
> (That's a serious question.)
>

NIST is running SHA-3 competition where we have 14 candidates in second round. Every discussion about every candidate and its properties is productive. On the other hand, since we have 14 candidates that are offering security more than 2^128, discussions that attacks that cost more than 2^128 or more than 2^192 or more than
2^256 computations of the compression function are non-productive since we all know the facts about physical infeasibility of such big computations. However, it is documented with the messages send to this forum, the problems, tensions, offensive language, mocking and ridiculing raise when someone presents attack or observation about one specific candidate: CubeHash.

====
Thomas Bonik

From: hash-forum@nist.gov on behalf of Gaëtan Leurent [gaetan.leurent@ens.fr]
Sent: Friday, October 15, 2010 3:48 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

D. J. Bernstein wrote on 10 Oct 2010 02:23:02 +0200:

> Let me again emphasize that all of the CubeHash security claims, both
> pre-quantum and post-quantum, are holding up just fine.

Of course! You made no security claim for CubeHash-normal, so all of them all holding up fine.

And your security statements are vague enough that any security level can be shoehorned into them as long as there is no practical attack.
I'm not impressed by prediction after the fact.

> There have been two incidents of other submission teams
> misrepresenting the CubeHash16/1-512 security claims as
> CubeHash16/32-512 security claims, and on this basis misrepresenting non-attacks as attacks.

No Dan, that is not what happened.

What happened is that two teams tried to interpret the security statements about CubeHash16/32-512, and that you then gave another interpretation of these statements. It has absolutely nothing to do with CubeHash16/1-512.

A sensible person would promptly make a clear security claim so as to avoid the repetition of such situations. For some reason, you are still avoiding to make one. I wonder what will happen when someone comes up with a classical collision attack with complexity 2^192...

>> For instance, I would really love to know how the following statement
>> does not cover quantum attacks:
>>   CubeHash16/32--512 is my main proposal, providing 2^256 security
>>   against all attacks.
>
> Gaetan, do you deny that your submission claims "complexity" 2^256 for
> every "preimage or second preimage attack" on SIMD-256? Do you deny
> that this is a claim of 2^256 security against all preimage attacks?

Well, that's funny. You complain when you're quoted out of context, but when I do quote you with a bit of context you just remove it...

So, yes, I deny that my claim has the same meaning as yours, because the context is vastly different. If you want a claim for the security of SIMD against quantum attacks, just ask for it, and I will gladly make one.

But let me set some things straight.

I'm *not* saying that a 2^k/2 quantum preimage on Skein would be more interesting than a 2^k classical preimage on Skein. For a function that aims to behave like a random

function, the properties have to be compared to generic attack, and either k<n and they are both attacks, or
k>n and none of them are attacks.

However, *you* made it pretty explicit that it was not the correct way to study CubeHash, and that studying classical preimage attacks above
2^256 was "stupid", "wacky", "silly", or "insane". But you didn't say the same about quantum attacks above 2^128. On the contrary, you said quite vehemently that quantum preimage attacks are far more relevant than classical preimage attacks, thus justifying that classical attacks over 2^256 are not covered. That's why I'm studying quantum attacks against CubeHash: because *you* said those are the attacks that matters.

> There's one submission whose security claims _do_ resist quantum
> computers: namely, CubeHash. For the second round I added several
> explicit preimage-security claims against quantum computers, and for
> those claims I was careful to chop all exponents in half, precisely so
> that I _wouldn't_ be exaggerating the security of CubeHash against
> quantum attacks. See http://cr.yp.to/papers.html#collisioncost to
> understand why this isn't necessary for collision attacks.

More precisely you said in the submission package:

  CubeHash expected strength (2.B.4) (strength.pdf) has been modified to
  note the expected impact of quantum computers. Grover’s algorithm will
  find (e.g.) 224-bit preimages for any of the SHA–3 candidates in only
  about 2^112 quantum operations. This quantum computer

  - has a much higher success chance than a conventional computer
    performing 2^200 operations and
  - is much more likely to be available to future attackers than a
    conventional computer performing 2^200 operations,

  so considering the conventional threat while ignoring the quantum
  threat makes no sense from a risk-analysis perspective.

  -- CubeHash round-2 modifications (Round2Mods.pdf)

By your very words, speaking about a 2^384 classical preimage attack, and ignoring a 2^192 quantum preimage attack makes no sense (even a
2^215 quantum attack would be more interesting than a 2^384 classical attack).

Then why wasn't my obvious attack mentioned in the submission package?
Maybe it wasn't so obvious, and you actually expected a higher security level against quantum attack, as expressed by numerous other statements?

In any event, it seems pretty clear that my new attack is better than the previous ones, according to your own criteria (or did those criteria change opportunely?).

--
Gaëtan Leurent

From:	hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent:	Friday, October 29, 2010 9:38 PM
To:	Multiple recipients of list
Subject:	OFFICIAL COMMENT: CubeHash (Round 2)

This is a comment on the paper "Linear Analysis of Reduced-Round CubeHash" by Ashur and Dunkelman.

The paper reports, as its main result, "distinguishing 11-round CubeHash using about 2^470 queries." A casual reader might think that this target is a large fraction of CubeHash's official 16 rounds.

However, that simply isn't true! CubeHash actually has, beyond the number of rounds, three additional defenses against this type of attack:

* CubeHash exposes at most 512 bits of its state to the attacker, not the 1024 bits assumed by Ashur and Dunkelman. As an illustration of how strong this defense is, Ashur and Dunkelman say they are able to distinguish only 5 rounds, not 11 rounds, if they take this restriction into account. (Ashur and Dunkelman analyzed this on page 13 of the version of the paper that they sent to me, but then suppressed the analysis in their online version for some reason.)

* CubeHash provides the attacker only 256 bits of control over the CubeHash state, not the 1024 bits assumed by Ashur and Dunkelman. This is an important restriction on message modification.

* Most importantly, CubeHash has many extra rounds of finalization. In particular, the second-round CubeHash MAC proposal has 320 rounds of finalization. This is obviously massive overkill, and I've already mentioned that I'm reducing it to 32 rounds---still very far beyond the 5 rounds attacked here.

Ashur and Dunkelman say that they are attacking reduced-round CubeHash.
What they are actually doing is reducing the rounds _and_ giving the attacker 4 times as much control over the input _and_ eliminating the entire finalization _and_ eliminating the final truncation. Describing this series of huge changes as merely "reduced-round CubeHash" is not even marginally reasonable; CubeHash reduced to 11 rounds, or even 5 rounds, would still be completely resistant to this attack.

My overall assessment, based on extensive third-party differential cryptanalysis, is that a CubeHash round has similar strength to a Serpent round. The linear analysis from Ashur and Dunkelman makes a CubeHash round seem even stronger than a Serpent round:

* The latest Serpent analysis uses 128 bits of input and 128 bits of output to detect a 2^(-52) bias in 9 rounds.

* This CubeHash analysis uses far more input (1024 bits) and far more output (1024 bits) and detects only a 2^(-235) bias in 11 rounds.

Ashur and Dunkelman describe this as a "high" bias without giving any justification for their value judgment. I find the analysis interesting, but the bottom line is that CubeHash has a much larger security margin against this type of attack than the authors admit.

---D. J. Bernstein
Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of orr.dunkelman@weizmann.ac.il
Sent: Saturday, October 30, 2010 8:07 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
Dear all,

This is a comment on the comment...

> This is a comment on the paper "Linear Analysis of Reduced-Round
> CubeHash" by Ashur and Dunkelman.
>
> The paper reports, as its main result, "distinguishing 11-round
> CubeHash using about 2^470 queries." A casual reader might think that
> this target is a large fraction of CubeHash's official 16 rounds.
>
> However, that simply isn't true! CubeHash actually has, beyond the
> number of rounds, three additional defenses against this type of attack:
>
>    * CubeHash exposes at most 512 bits of its state to the attacker, not
>      the 1024 bits assumed by Ashur and Dunkelman. As an illustration of
>      how strong this defense is, Ashur and Dunkelman say they are able
>      to distinguish only 5 rounds, not 11 rounds, if they take this
>      restriction into account. (Ashur and Dunkelman analyzed this on
>      page 13 of the version of the paper that they sent to me, but then
>      suppressed the analysis in their online version for some reason.)
>
>    * CubeHash provides the attacker only 256 bits of control over the
>      CubeHash state, not the 1024 bits assumed by Ashur and Dunkelman.
>      This is an important restriction on message modification.

Both of these issues are clearly mentioned at the end of Section 5 (for whomever missed
this while reading the rest of the paper):

"We emphasize that as our results are linear in nature, they require that the adversary
has access both to the input to the nonlinear function as well as its output. To the best
of our knowledge, there is no way to use this directly in a hash function setting."

>    * Most importantly, CubeHash has many extra rounds of finalization.
>      In particular, the second-round CubeHash MAC proposal has 320
>      rounds of finalization. This is obviously massive overkill, and
>      I've already mentioned that I'm reducing it to 32 rounds---still
>      very far beyond the 5 rounds attacked here.

Indeed. Just as a general comment, if you happen to use prefix-MAC, then there are only
160 finalizations in the plain CubeHash.

> My overall assessment, based on extensive third-party differential
> cryptanalysis, is that a CubeHash round has similar strength to a
> Serpent round. The linear analysis from Ashur and Dunkelman makes a
> CubeHash round seem even stronger than a Serpent round:
>
>    * The latest Serpent analysis uses 128 bits of input and 128 bits of
>      output to detect a 2^(-52) bias in 9 rounds.
>
>    * This CubeHash analysis uses far more input (1024 bits) and far more
>      output (1024 bits) and detects only a 2^(-235) bias in 11 rounds.

Well, if you discuss 1024-bit to 1024-bit transformations, then CubeHash is susceptible to
a 14-round approximation.
```

Moreover, Serpent never tried (and cannot) offering a 256-bit security against linear cryptanalysis, CubeHash can. Do you claim that CubeHash offers only 128-bit security?

In any case, if I recall correctly (I can look up the numbers in a few days, if it is really of interest), for approximations of 6 rounds of CubeHash, the bias is roughly the same as 2^-52 (I may be off by a 10). So 128-bit security in linear cryptanalysis takes 6 rounds in CubeHash (out of 16), and 9 in Serpent (out of 32). The safety margins: 10/16 rounds in CubeHash (62.5%) vs. 22/32 in Serpent (71.88%).

> Ashur and Dunkelman describe this as a "high" bias without giving any
> justification for their value judgment. I find the analysis
> interesting, but the bottom line is that CubeHash has a much larger
> security margin against this type of attack than the authors admit.

A random function from 1024-bit to 1024-bit should not have biases which are more than 2^-510 or so (actually a bit higher, but it would not be more than 2^{-480} for a random function).

Anything which is more than that, is high.

We were simply surprised that 11-round CubeHash was not offering more security against linear cryptanalysis. Moreover, if you will look carefully, our approximations are based on some specific rotation constants. We were surprised (again, because we value CubeHash) that the longest approximations exist and satisfy the constraints on the rotations, as we expected them to be shorter.

Best regards,
Orr

--
Orr Dunkelman,
Faculty of Mathematics and Computer Science The Weizmann Institute of Science Tel. +972-8-934-3365

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Saturday, October 30, 2010 11:49 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

```
Orr Dunkelman writes:
> So 128-bit security in linear cryptanalysis takes 6 rounds in CubeHash
> (out of 16), and 9 in Serpent (out of 32).

No. "3 CubeHash rounds out of >=32" would be a much more accurate summary of your results.

The reason that you say 16, rather than >=32, is that you're ignoring the CubeHash
finalization. This finalization adds many more rounds before the state is exposed to the
attacker.

The reason that you say 6, rather than 3, is that you're ignoring the final CubeHash
truncation. This truncation hides a large part of the CubeHash state, making linear
cryptanalysis much more difficult.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago
```

From: hash-forum@nist.gov on behalf of Marsh Ray [marsh@extendedsubset.com]
Sent: Saturday, October 30, 2010 7:33 PM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

On 10/29/2010 08:37 PM, D. J. Bernstein wrote:
>
>   * CubeHash exposes at most 512 bits of its state to the attacker, not
>   the 1024 bits assumed by Ashur and Dunkelman.

I don't get it.

Doesn't the concept of 2nd preimage resistance represent a model in which the attacker knows an entire message and thus "expose" the entire state?

Even for 1st preimages, because the CH rounds are fully invertible the attacker has easy access to 2^512 distinct internal states which all result in the target value. Of course, finding a message prefix to reach one of these states is the challenge. At best this requires a brute force search for a 768 bit collision. This would be a three-block message, with the attacker computing 16 rounds in the forward and reverse directions to meet at the point of the middle 256-bit message text injection. This would appear to be 2^384 work, but any common bias that the attacker is able to engineer in those 768 middle non-message bits (or even expressions constructed from them) serves to reduce his work.

> As an illustration of
>   how strong this defense is, Ashur and Dunkelman say they are able
>   to distinguish only 5 rounds, not 11 rounds, if they take this
>   restriction into account. (Ashur and Dunkelman analyzed this on
>   page 13 of the version of the paper that they sent to me, but then
>   suppressed the analysis in their online version for some
> reason.)
>
>   * CubeHash provides the attacker only 256 bits of control over the
>   CubeHash state, not the 1024 bits assumed by Ashur and Dunkelman.
>   This is an important restriction on message modification.

Alternatively, the attacker could attempt to find an internal collision with a two-block message. He would trivially control 256 bits at each message data injection point for a total of 512 bits. Also, some small number of the 1536 non-message bit values could be fixed through simple brute force searching (with one additional message text block before and after). He then seeks to find a mid-block 1024-bit collision of the forward and reverse computations of only 8 rounds.

At first glance a 1024-bit collision seems like far more work than a 768-bit one (though both are cosmic scale). But its also seems likely that the 8 round transform is categorically weaker than the full 16 round one.

Each CH round involves 32 addition operations of 32 bits each. The 31 carry inputs of each addition are the only non-linear elements in CubeHash (and carries out of the lower bit positions will have significant bias). So the 8 rounds introduce at most 31*32*8 = 7936 non-linear bits, a noticeable percentage of which can probably be effectively marginalized through selection of message data.

It's probably not an apples-to-apples comparison, but as a point of reference, it looks like MD5 incorporates 30-40000 non-linear bits per
512 bits of message input.

For another data point, in
   http://research.microsoft.com/en-us/people/mironov/sat-hash.pdf
the authors say:

```
      we translate the rounds 12–19 of the compression function of
      MD4 into a CNF formula with 10364 variables and 39482 clauses.
      [...]
      SatELiteGTI finds 222 solutions to the resulting formula
      in less than one hour on a PC.

>     * This CubeHash analysis uses far more input (1024 bits) and far more
>       output (1024 bits) and detects only a 2^(-235) bias in 11 rounds.
>
> Ashur and Dunkelman describe this as a "high" bias without giving any
> justification for their value judgment.

Haha, I never thought I'd hear anyone debating whether or not 2^(-235) was a big number.
:-)

- Marsh
```

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Sunday, October 31, 2010 12:56 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash (Round 2)

Marsh Ray writes:
> Doesn't the concept of 2nd preimage resistance represent a model in
> which the attacker knows an entire message and thus "expose" the entire state?

Yes, but Ashur and Dunkelman don't have a 2nd-preimage attack! Here are the two attacks stated in their paper:

* A 2^396-query PRF attack if CubeHash is used as a prefix MAC and weakened in the following way: reduce the number of finalization rounds from 160 to 5.

* A 2^470-query PRF attack if CubeHash is used as an Even-Mansour cipher and weakened in the following four ways: expand the block size to 1024 bits; reduce the number of rounds from 16 to 11; reduce the number of finalization rounds from 160 to 0; and expand the output size to 1024 bits.

The authors suppressed the first attack from their online version, and quite unreasonably described the target of the second attack as merely "reduced-round CubeHash"---highlighting the 16->11 reduction while making most readers miss the 256->1024 input expansion, the 512->1024 output expansion, and the complete elimination of finalization.

I really think that the authors are obliged to provide a clear list of all four of the CubeHash defenses that they're suppressing to make their attack work. I also told them this before they posted their paper.

> This would appear to be 2^384 work, but any common bias that the
> attacker is able to engineer in those 768 middle non-message bits (or
> even expressions constructed from them) serves to reduce his work.

No, it doesn't. As Ashur and Dunkelman comment, linear cryptanalysis is not useful for finding collisions except when the bias is extremely high. If you don't believe us, try reducing CubeHash from 16 rounds to 2 rounds, and explain how the 2-round biases found by Ashur and Dunkelman (around 2^(-10)---let's charitably assume that these can be isolated in attacker-controlled bits) can be used to speed up finding collisions.

> At first glance a 1024-bit collision seems like far more work than a
> 768-bit one (though both are cosmic scale). But its also seems likely
> that the 8 round transform is categorically weaker than the full 16
> round one.

The largest bias reported by Ashur and Dunkelman is 2^(-123) in 8 rounds, using a flip of 10 bits, only 2 of which are actually controlled by the attacker. The attacker (1) doesn't have any way to find inputs with this relationship, (2) doesn't have any way to detect, let alone exploit, the very small output bias, and (3) is starting out with 256 fewer controlled bits than the generic attacks reported in the original CubeHash submission. I can understand why Ashur and Dunkelman didn't bother exploring this further!

---D. J. Bernstein
Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Sunday, October 31, 2010 7:26 PM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: CubeHash tweak

I have written a document "CubeHash parameter tweak: 10x smaller MAC overhead" issuing new higher-speed recommendations for the CubeHash parameters and for message authentication using CubeHash:

   http://cubehash.cr.yp.to/submission2/tweak2.pdf

The CubeHash algorithm itself is unchanged, and cryptanalysts are faced with the same spectrum of reduced-security targets as challenges. If CubeHash is allowed into the third round then I would like this tweak to be considered as part of the submission.

The tweak document also covers several related issues: the importance of large security margins; the impact of finalization; the trouble with tweaks; the trouble with options; flexibility; interoperability; bait and switch; and the complexity of cryptanalysis. These issues are illustrated with various observations, positive and negative, about many different SHA-3 candidates.

Benchmarks for cubehash512 (CubeHash16+16/32+32-512) on many software platforms are now online, showing that CubeHash is now between #2 and #6 in software performance on every modern CPU, for long messages and short messages. For example, on an i.MX515 (ARM Cortex A8, like the iPad), the only competitors are

```
   shabal;
   bmw---but only bmw256; bmw512 is slower than cubehash512;
   blake---but only blake32; blake64 is slower than cubehash512;
   luffa---but only luffa256, by a nose; luffa{384,512} are slower.
```

On a Core 2 Quad Q9550 (berlekamp)---in 64-bit mode, the best case for most SHA-3 candidates---the only competitors are

```
   bmw;
   shabal;
   skein;
   blake;
   simd---but only simd256; simd512 is slower.
```

Furthermore, I expect CubeHash to end up as #1, #2, or #3 in every sensible hardware benchmark. CubeHash was already near the top of most benchmark reports, and should now be even better, for three reasons:

* The tweak reduces finalization to 32 rounds, the equivalent of hashing just 64 bytes, drastically reducing per-message overhead.

* The tweak reduces initialization to 16 rounds, allowing many implementations to eliminate 1024 bits of ROM.

* CubeHash's heavy vectorizability allows a spectrum of low-area high-speed options that were missed in previous hardware reports.

I've posted cubehash512/hardware{2,4,8,16} implementations that can easily be translated into HDLs; I request that hardware benchmarking teams update their reports accordingly.

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Monday, November 01, 2010 3:56 AM
To: Multiple recipients of list
Subject: Re: OFFICIAL COMMENT: CubeHash tweak

Danilo Gligoroski writes:
> I expect that soon you will update the SUPERCOP package.

Already done last week. Measurements for your new BMW implementations are already online from 18 computers. I'm sure we'll have measurements online from most of the remaining computers long before NIST makes the final cut---and last-minute benchmark data is obviously much easier to assess than last-minute cryptanalysis.

> The new un-optimized NEON BMW512 implementation on ARM Cortex A8
> achieves 24cpb while cubehash512 achieves ~27cpb.

If so then that's 1.4x faster than your previous implementation; thanks for the data point!

---D. J. Bernstein
   Research Professor, Computer Science, University of Illinois at Chicago

From: hash-forum@nist.gov on behalf of D. J. Bernstein [djb@cr.yp.to]
Sent: Saturday, November 20, 2010 12:54 AM
To: Multiple recipients of list
Subject: OFFICIAL COMMENT: CubeHash: small fast algorithms

This week I've worked out details of three new algorithms to compute CubeHash. Here's software showing how the algorithms work:

```
    8 cycles per round: http://cubehash.cr.yp.to/hardware8/hash.c
   16 cycles per round: http://cubehash.cr.yp.to/hardware16/hash.c
   32 cycles per round: http://cubehash.cr.yp.to/hardware32/hash.c
```

The algorithms are much more hardware-friendly than previous methods of computing CubeHash, including my previous ASIC-oriented implementations.
The software is written in a combinational style that should be very easy to translate to various hardware-design languages, demonstrating that CubeHash can maintain high speeds while being squeezed into smaller and smaller areas. Here are the critical metrics:

* State size: These algorithms naturally squeeze the CubeHash state into 1024 bits of SRAM and just 192 extra flip-flops. The 8-cycle algorithm accesses only 256 bits of SRAM per cycle; the 16-cycle algorithm accesses only 128 bits of SRAM per cycle; the 32-cycle algorithm accesses only 64 bits of SRAM per cycle.

  The SRAM access is completely regular: each cycle reads once and writes once at a position linearly indexed by the cycle counter. For the same reason, the 32-cycle algorithm should also work very well with 16-bit shift registers or LUT RAMs on FPGAs.

  For comparison, the CubeHash ASIC reports in the literature have all used 1024 flip-flops. 1024 bits of SRAM, with only a fraction accessed per cycle, are far smaller than 1024 flip-flops.

* Computation size: My new 8-cycle algorithm uses far fewer bit operations per cycle than my earlier 4-cycle algorithm. The savings is almost a factor of 2. My new 16-cycle algorithm saves another factor of almost 2---it has fewer than 1000 bit operations. My new 32-cycle algorithm saves yet another factor of almost 2.

  For comparison, the 4-cycle algorithm saved a factor of almost 2 compared to the 2-cycle approach from Bernet et al., and the 2-cycle approach was already considerably smaller than the 1-cycle approach used in most CubeHash ASIC reports.

  I should emphasize that the sizes I'm talking about are for unified CubeHash256/CubeHash512/CubeHash512x units, with negligible penalty for run-time selection of the hash output size. CubeHash does not encourage future interoperability problems.

* Clock speed: The critical path in the 8-cycle algorithm is only slightly longer than an 8-bit adder. The 16-cycle and 32-cycle algorithms have even shorter critical paths.

  For comparison, the CubeHash ASIC reports in the literature use 32-bit adders. The shorter critical paths in the new algorithms allow higher clock speeds---depending on the underlying ASIC library, of course. Note that doubling the number of cycles can _improve_ the throughput-to-area ratio if the clock speed goes up a little bit and the area drops by nearly a factor of 2.

All of the new algorithms rely critically on the regular structure of CubeHash: vectorizability, reuse of rotation distances, etc. None of the other SHA-3 proposals offer anywhere near the same level of flexibility; perhaps this is why it didn't occur to the previous implementors to try such approaches for CubeHash.

For comparison, Luffa-256 (don't ask about Luffa-384!) has advertised being squeezable from 20000 GEs to 14000 GEs to 10340 GEs, but loses a factor of almost 5 in speed from the first squeezing and loses another factor of 6 from the second, according to the numbers presented by the designers in August. As another example, Keccak has advertised being squeezable from 40000 GEs to 9300 GEs, but loses a factor of more than 100 in performance from this. Neither Luffa nor Keccak can be squeezed much further.

There was one low-area CubeHash implementation report, by Bernet et al., but the new 32-cycle algorithm is obviously much better than the approach taken in that report. Specifically, the report squeezed CubeHash from 21540 GEs to 7630 GEs, but lost more than a factor of 100 in speed. The new 32-cycle algorithm does better in two ways:

* >20x throughput improvement: Much higher clock speed _and_ half as many cycles per round. Bernet et al. lost a factor of 10 from clock speed from squeezing (according to their own figures), while my algorithms _gain_ clock speed from squeezing.

* Close to 2x area improvement, primarily from replacing most of the flip-flops with SRAM.

Bernet et al. didn't document the ASIC library that they were using, so it's impossible for anyone else to produce concrete numbers for comparison, but it's clear that for any library the new approach will provide much higher CubeHash speeds in smaller area than the approach by Bernet et al.

CubeHash was already very close to the best throughput-to-area ratio among all SHA-3 candidates. CubeHash is unique in continuing to provide an excellent throughput-to-area ratio when it is squeezed into smaller and smaller areas. CubeHash now provides by far the smallest area among
SHA-3 candidates for a wide range of throughput targets, and is the only
SHA-3 candidate that can fit full security comfortably onto RFIDs.

---D. J. Bernstein
Research Professor, Computer Science, University of Illinois at Chicago