Single Block Attacks and Statistical Tests on CubeHash

Benjamin Bloom[*] Alan Kaminsky[†]

August 21, 2009

Abstract

This paper describes a second preimage attack on the CubeHash cryptographic one-way hash function. The attack finds a second preimage in less time than brute force search for these CubeHash variants: CubeHash r/b-224 for $b > 100$; CubeHashr/b-256 for $b > 96$; CubeHashr/b-384 for $b > 80$; and CubeHashr/b-512 for $b > 64$. However, the attack does not break the CubeHash variants recommended for SHA-3. The attack requires minimal memory and can be performed in a massively parallel fashion. This paper also describes several statistical randomness tests on CubeHash. The tests were unable to disprove the hypothesis that CubeHash behaves as a random mapping. These results support CubeHash's viability as a secure cryptographic hash function.

1 Introduction

Invented by Daniel Bernstein, CubeHash [3] is a cryptographic one-way hash function submitted as a candidate to the NIST SHA-3 hash function competition. CubeHash has three parameters:

- r, the number of mixing rounds applied to each message block ($r \geq 1$).
- b, the length of a message block in bytes ($1 \leq b \leq 128$).
- h, the size of the hash value in bits ($h = 8, 16, 24, \ldots, 512$).

"CubeHashr/b-h" denotes the CubeHash variant with certain parameter choices; for example, CubeHash1/128-512.

CubeHash is expected to be stronger (more resistant to attack) as the number of mixing rounds r is increased and as the message block size b is reduced. The CubeHash variants recommended in the original SHA-3 submission are CubeHash8/1-224, CubeHash8/1-256, CubeHash8/1-384, and CubeHash8/1-512. Later (after the work described in this paper was done), Bernstein revised

[*]Department of Computer Science, Rochester Institute of Technology, bwb1636@cs.rit.edu
[†]Department of Computer Science, Rochester Institute of Technology, ark@cs.rit.edu

the recommended variants for SHA-3 to be CubeHash16/32-224, CubeHash 16/32-256, CubeHash16/32-384, and CubeHash16/32-512 [5]. Several attacks and collisions on reduced-strength variants of CubeHash have been reported [1, 2, 8, 9, 10, 11, 12, 13, 18]. None of these break the CubeHash variants recommended for SHA-3 (neither the original nor the revised variants).

We describe a second preimage attack on CubeHash. It is called a "single block attack" because it finds a second preimage consisting of a single message block. The attack finds a second preimage in less time than brute force search (fewer than 2^h hash function evaluations for an h-bit hash) for these CubeHash variants: CubeHashr/b-224 for $b > 100$; CubeHashr/b-256 for $b > 96$; CubeHashr/b-384 for $b > 80$; and CubeHashr/b-512 for $b > 64$. Note that the attack works equally well for any number of mixing rounds r. However, the attack does not break the CubeHash variants recommended for SHA-3. The attack requires minimal memory and can be performed in a massively parallel fashion.

In addition, several statistical randomness tests can be performed on CubeHash using the single block attack methodology. To our knowledge, this paper is the first to report the results of statistical randomness tests on CubeHash. The tests failed to disprove the hypothesis that CubeHash behaves as a random mapping.

The paper is organized as follows. Section 2 describes the single block attack. Section 3 describes the implementation of the attack as a massively parallel program for a hybrid parallel computer and reports the program's performance. Section 4 gives examples of second preimages found using the parallel program. Section 5 describes the statistical tests and their results. Section 6 compares the single block attack to previously reported attacks.

2 Single Block Attack

2.1 Hash Computation

The CubeHash state consists of 128 bytes. The state bytes are organized into 32 words, with four bytes per word in little-endian order. The state words are designated x_{00000} through x_{11111}. Word x_{00000} contains byte 0 (least significant) through byte 3 (most significant), word x_{00001} contains bytes 4–7, and so on.

The hash of a b-byte message is computed as follows using CubeHashr/b-h (Figure 1). This is the procedure for hashing a message consisting of a single b-byte block. CubeHash can hash a message of any size; the procedure for doing so is described in [3].

1. Set the state to the initialization vector (IV). The initialization vector is described later. This is shown as $State_0$ in Figure 1.
2. Exclusive-or the b bytes of the message into the first b bytes of the state, yielding $State_1$. (The last $128 - b$ bytes of $State_1$ remain the same as the IV.)

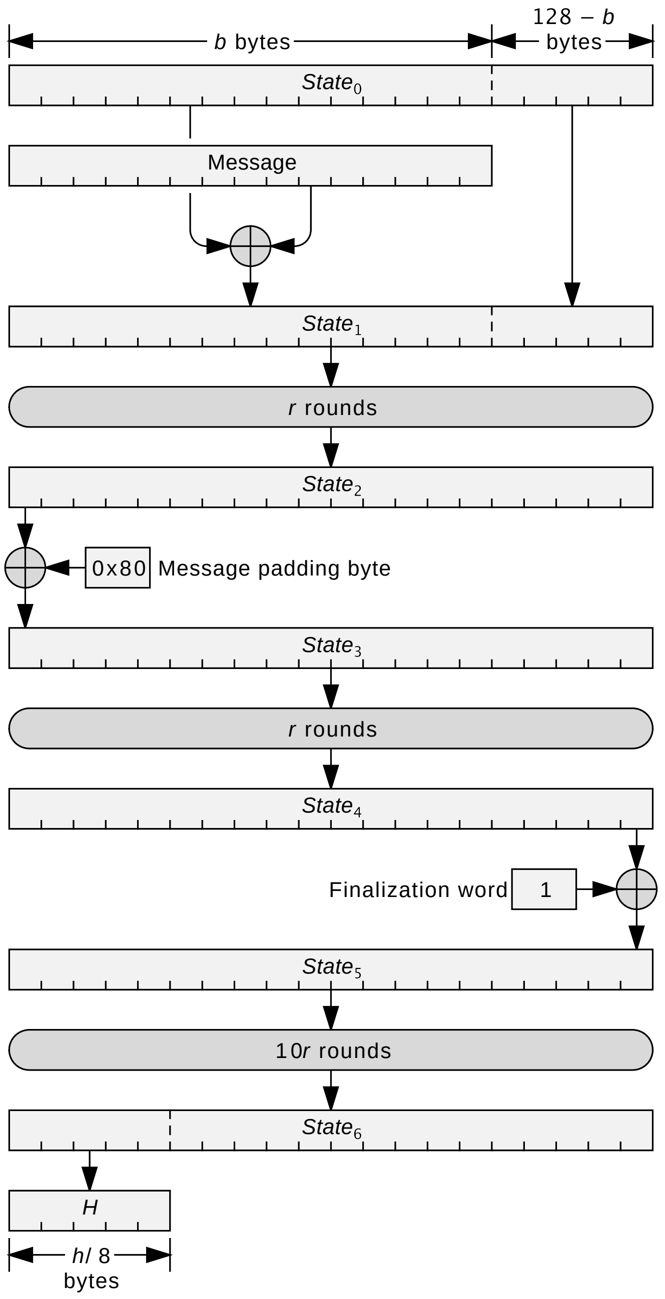

Figure 1: Computing the CubeHash hash of a single block message

3. Apply r mixing rounds to the state, yielding $State_2$. The round function is described later.
4. Exclusive-or a message padding byte of 0x80 into the first byte of the state, yielding $State_3$.
5. Apply r rounds to the state, yielding $State_4$.
6. Exclusive-or the number 1 into the last word of the state, yielding $State_5$.
7. Apply $10r$ rounds to the state, yielding $State_6$. (Steps 6–7 finalize the computation.)
8. The hash of the message, H, is the first $h/8$ bytes of the final state.

The round function does the following invertible transformation on the CubeHash state [3].

1. Add x_{0jklm} into x_{1jklm} modulo 2^{32}, for each (j, k, l, m).
2. Rotate x_{0jklm} upwards by 7 bits, for each (j, k, l, m).
3. Swap x_{00klm} with x_{01klm}, for each (k, l, m).
4. Exclusive-or x_{1jklm} into x_{0jklm}, for each (j, k, l, m).
5. Swap x_{1jk0m} with x_{1jk1m}, for each (j, k, m).
6. Add x_{0jklm} into x_{1jklm} modulo 2^{32}, for each (j, k, l, m).
7. Rotate x_{0jklm} upwards by 11 bits, for each (j, k, l, m).
8. Swap x_{0j0lm} with x_{0j1lm}, for each (j, l, m).
9. Exclusive-or x_{1jklm} into x_{0jklm}, for each (j, k, l, m).
10. Swap x_{1jkl0} with x_{1jkl1}, for each (j, k, l).

The IV is computed by setting state word x_{00000} to $h/8$, x_{00001} to b, x_{00010} to r, and x_{00011} through x_{11111} to 0, then applying $10r$ rounds to the state, yielding $State_0$ [3].

2.2 Attack Computation

Because the CubeHash round function is invertible, the initial state can be computed by working backwards from the final state. The single block attack, first reported in [7], is based on this ability.

Given the hash value H of some message, a b-byte second preimage message is computed as follows using CubeHashr/b-h (Figure 2).

1. Set the first $h/8$ bytes of the final state to the hash value H and set the last $128 - h/8$ bytes to a trial value T, yielding $State_6$. The manner of choosing the trial value is described later.
2. Apply $10r$ reverse rounds to the state, yielding $State_5$. The reverse round function does the steps of the round function in reverse order, substituting downward rotations for upward rotations and subtraction modulo 2^{32} for addition modulo 2^{32}.
3. Exclusive-or the number 1 into the last word of the state, yielding $State_4$.
4. Apply r reverse rounds to the state, yielding $State_3$.
5. Exclusive-or a message padding byte of 0x80 into the first byte of the state, yielding $State_2$.

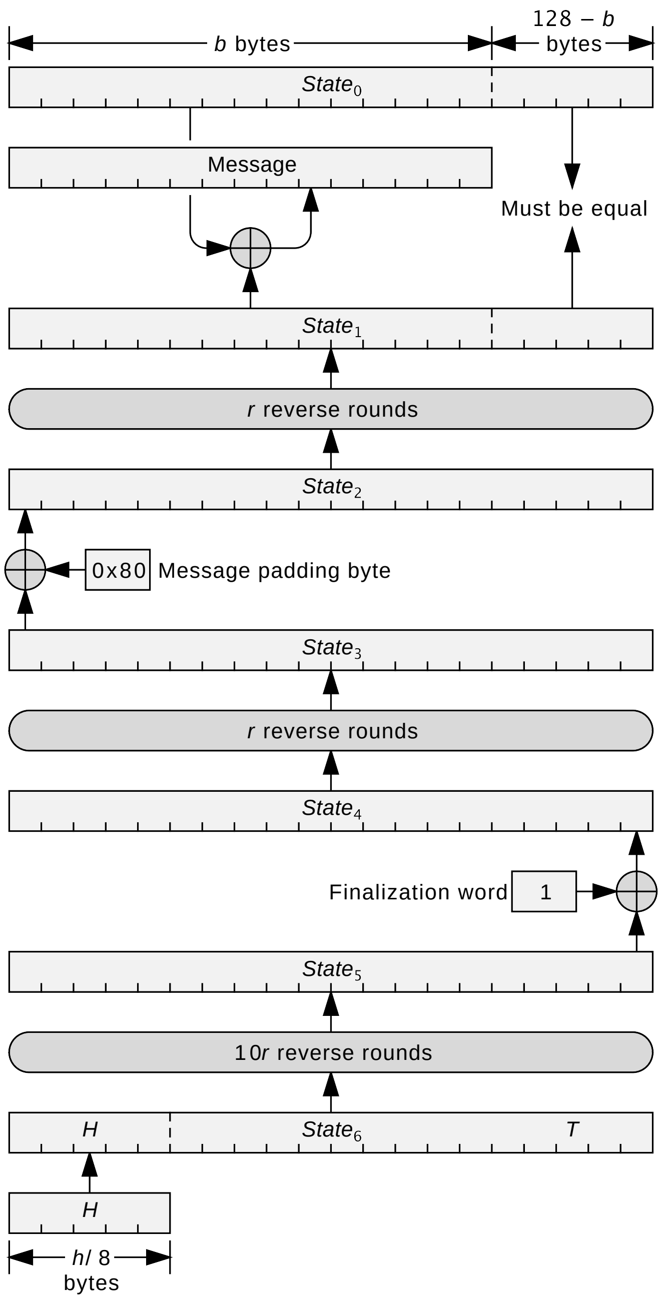

Figure 2: The single block attack on CubeHash

6. Apply r reverse rounds to the state, yielding $State_1$.
7. If the last $128 - b$ bytes of $State_1$ are not equal to the last $128 - b$ bytes of the IV ($State_0$), then the trial was unsuccessful.
8. Otherwise, the trial was successful. The second preimage is computed by exclusive-oring the first b bytes of $State_1$ with the first b bytes of the IV.

Repeating the above procedure with different values of T will eventually generate a second preimage. The manner of choosing T values is not critical. For example, a sequence of consecutive values could be used, or randomly-chosen values could be used.

Assuming that CubeHash (in either the forward or the reverse direction) behaves as a random mapping, for an arbitrary trial value T, the probability that the final $128-b$ bytes of $State_1$ are equal to the corresponding bytes of the IV is $2^{-8(128-b)}$. Therefore, the expected number of trials to achieve success is $2^{8(128-b)}$. If $2^{8(128-b)} < 2^h$, then the single block attack finds a second preimage with fewer expected trials than brute force search. In other words, the single block attack is more effective than brute force search for $h = 224$ and $b > 100$, for $h = 256$ and $b > 96$, for $h = 384$ and $b > 80$, and for $h = 512$ and $b > 64$.

Note that the expected number of trials to achieve success does not depend on the number of rounds r. Of course, the running time of the attack computation increases as r increases.

Also note that for $b = 128$, the probability of success is 1; in other words, *any* value of T immediately produces a second preimage.

3 Parallel Implementation

The single block attack was implemented as a Java program, using the Parallel Java Library [15, 16], to run on a hybrid SMP cluster parallel computer—that is, a cluster of compute nodes, each node being a shared memory multiprocessor (SMP), or multicore, computer. The program is available at [17].

When executed on one processor, the single block attack program does the following. The program generates a random b-byte message and computes its hash using CubeHashr/b-h. The program then does n trials in sequence. (r, b, h, n, and the pseudorandom number generator seed are specified on the command line.) Each trial carries out the attack computation with a different trial value T from 0 through $n-1$. Each trial's outcome is success (a second preimage was found) or failure. The program counts the number of successes and records the smallest value of T that yielded a second preimage. The program reports those results as well as the original message, the second preimage derived from the smallest successful T, and the hash value H.

When executed in parallel, the single block attack program runs with Kp parallel processes, one process per compute node, and with Kt parallel threads in each process, one thread per core on each node—a total of $K = Kp \cdot Kt$ threads altogether. The n trials are partitioned equally among the K threads, and the trials are performed in parallel. Each thread counts the number of successes

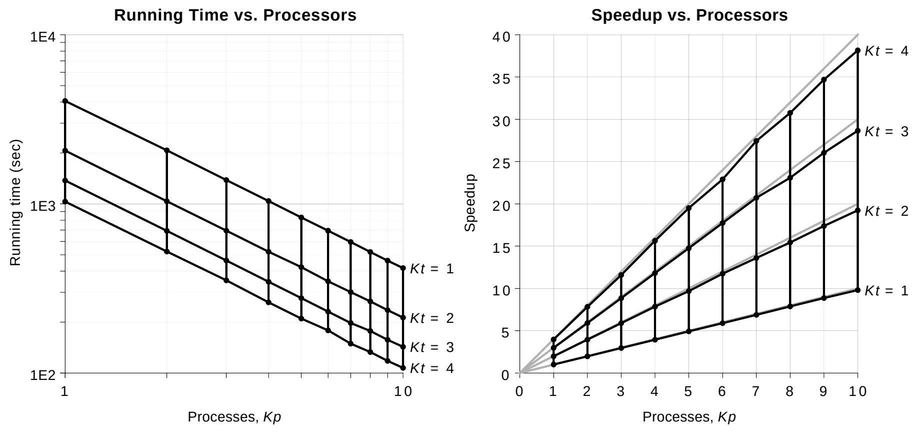

Figure 3: Program performance for CubeHash1/127-512, 2^{32} trials

and records the smallest successful T for that thread's portion of the trials. When all trials have finished, the threads on each node do a shared-memory parallel reduction to combine the per-thread results, yielding the total number of successes and the smallest successful T for that node. The processes on all the nodes then do a message-passing parallel reduction to combine the per-node results, yielding the total number of successes and the smallest successful T for the entire program run. One process reports the results. Apart from the final reductions, the program executes in a massively parallel fashion with no communication or coordination among the threads or processes.

The single block attack program was run on a hybrid SMP cluster parallel computer with ten nodes, each node a Sun Microsystems workstation with two AMD Opteron 2218 dual-core CPUs, a 2.6 GHz clock, and 8 GB of main memory. The nodes are connected by a dedicated 1 Gbps switched Ethernet network. On this computer, the program can scale up to as many as $Kp = 10$ processes and $Kt = 4$ threads per process; that is, 40 processors. The nodes use the SunOS 5.10 operating system and the Sun JDK 1.5.0_15 HotSpot Server Java Virtual Machine (JVM).

Figure 3 plots the running time and the speedup of the single block attack program for CubeHash1/127-512 with 2^{32} trials versus the number of parallel processors. Speedup is the running time of a sequential version of the program (with one process, no multithreading, and no message passing) relative to the running time of the parallel program. The program experiences speedups 95% of the ideal or better all the way out to 40 processors. Complete running time measurements are available at [17].

For CubeHash1/127-512, the single block attack program takes 4,059 seconds to do 2^{32} trials on one processor, or 945 nsec per trial. Since for $r = 1$ each trial consists mostly of 12 applications of the (reverse) round function, the program takes 78.8 nsec per round. At a clock speed of 2.6 GHz, the program takes 205 cycles per round. For comparison, the CubeHash speeds reported by the eBASH Project [14] for a C program on x86-architecture processors range from 21 to 535 cycles per round. (These are derived from the median speed of CubeHash8/1-512 in cycles per byte for long messages, divided by 8 rounds per byte.)

4 Attack Results

The single block attack program was executed with parameter settings $r = \{1, 2, 4, 8\}$ rounds per message block, $b = \{127, 126, 125\}$ bytes per message block, $h = 512$-bit hash value, $n = 2^{32}$ trials, and 100 different pseudorandom number generator seeds. The program was also executed with parameter settings $r = \{1, 2, 4, 8\}$ rounds per message block, $b = 124$ bytes per message block, $h = 512$-bit hash value, $n = 2^{40}$ trials, and 10 different pseudorandom number generator seeds. Some of the second preimages found are listed below, along with the decimal trial numbers T that generated the second preimages; complete results are available at [17]. The messages and hash values are given in hexadecimal, with each pair of digits representing one byte.

In the following examples, the original message consists of the first b bytes of this sequence of 128 bytes:

```
951fefce947916ccbaf6134ab2b377a54db9f0b7c1f4932b5bc68147dce57828
ba5b054f446fecc05c9086e96555fada9118b5598364d1b023f425bcdd094505
7c33e9fbbc20ff096ba9740f4d278f4922d5178ae650210fa9680512bf998ef1
2e9f246b5266e40c6240b7a681566d4a3817c19319bbcaede6cf93df7635ebdf
```

CubeHash1/127-512—second preimage for $T = 190$:

```
634a7948fcfdfc384e0e6f9bcff365aec19076d0629d9225ff717f9a6c0b832c
bfb27a6ef39927000b59695995efade283b695b4233f48147a0eba13b4e9e775
251f70b402da5b7e52b8306ba46aacc0655a5f0e73b7f9321f520ed656dbcb3f
5f4c8490bc02885ba622e9155703e0ec845d19a459b4563f686661af8c2320
```

Hash value of original message and second preimage:

```
9975b5bb8fb33005a6bc7414d13d470581474a69f6c2cb01e3e8fe8150996272
4d3633a9ab53cdbe8cbf8d61fa9650d0e16e0be832c607eaeaed2bb1f58a078e
```

CubeHash2/126-512—second preimage for $T = 92{,}677$:

```
cf9f6826607b68a01f1241715c38440c6b34c28fa356c3bbc0d7a6440870b8c5
02a8443c503630361feebe7a9638c77469250f39285ab31b035e28de492ecfce
ce4d446a36b40a90e90ab9e89424fc30aeaa25f97aa717a978735c9a2c028a77
b3e1fd5b96cedbe6902f4ae272e8a42d21b2993a266ebd0bfc9eb03c9147
```

Hash value of original message and second preimage:

```
58a1c388b6523a2cf2f7d75e58dc5f4b4d2f7715b12327ba32e4422149d554d1
70d735851cba47a2faf38f63bf23b326a3c09ba0d4887a655c6913d7f22446bd
```

CubeHash4/125-512—second preimage for T = 10,746,494:

```
985de06c0e2484aed791dafa9eeb4e28a3a635d43cd6e02c3712f4d150516602
7290996865b1e7248bc12f6bbc426553fcad2a666517c935a5dda79539fd3739
5739d14b83a03a862f0b12af35a4ad3584cf98818c5c99927822dad8a8b5d65a
6d1e047614a6c62c00607b470ae2937b6f8b11779d76e0f9bb5295305f
```

Hash value of original message and second preimage:

```
f3a508e80f39cf1ec5a079559a65f49eb8df6e86d67cd4455084dd06219af708
233b89c4b6443fc4ccd077c8358e59c6ffd6871bb5880d13291bf4dcaa778808
```

CubeHash8/124-512—second preimage for T = 2,860,087,247:

```
d901bc3da81f07c292d9d074825b0fddafb87304fde1fe54fd9cd7c88befbfbf
644e39d6d437a99ab9d19dc4f5c3fbf2a61a51533afa4f27c7fabc51c356bb1e
2b23d1252ca8e4c421a883c2c43d69adbf7a2adc257b219408717ad04ec13b21
6cf31959fed1e6450c1795280361003affbb2cfa6bc0aa786f434911
```

Hash value of original message and second preimage:

```
b130c28fbb1dc8aa1135c2a85e46826ab272247a61c246a041664b1e9bad2bd2
e14c0e0f19386c4838b2214140e6477d7b1b1804128fd9e13a039c8ad26f5ba6
```

As mentioned previously, the single block attack succeeds with probability 1 for $b = 128$ and any trial value T; no search is required. Here is the second preimage of the original message for CubeHash8/128-512 with $T = 0$:

```
bdcbaa75be2f003456bd95652dc85f04a0d2bd31947435600ca3ea077d884d79
da4d7cf62a490188d908d8722af8ee10b56fa22ba06873a83203c5b1d8c3ebfc
dfb180cc60bca3dbf086977ab41515749577f1aef8bb1af2b7f20f811aa54b86
a5a82b8d88d83a56aadb3d642128007dfaf964bb4dc2e1c3c6fe4e8b19362d65
```

Hash value of original message and second preimage:

```
2631bf00ee41369ce3a19ac4a91cf5df4fd9cffb18278d5f32b193b4bf82c58c
73bd83bdcd81a0729ee9a6b862948edbe2a160ea9b8403e710328711e66d9311
```

5 Statistical Tests

5.1 Number of Successful Trials

Under the null hypothesis that CubeHash behaves as a random mapping, when the single block attack program does n trials with CubeHashr/b-h, the expected number of successes s is $n{\cdot}2^{-8(128-b)}$ and the expected number of failures is $n{-}s$. The single block attack program performs a χ^2 test on the observed number of successes o to attempt to disprove the null hypothesis. The χ^2 statistic is

$$\chi^2 \;=\; \frac{(o-s)^2}{s} + \frac{((n-o)-(n-s))^2}{n-s} \;=\; \frac{(o-s)^2}{s} + \frac{(o-s)^2}{n-s} \qquad (1)$$

Table 1: χ^2 Tests of Number of Successful Trials

		$b = 127$		$b = 126$		$b = 125$		$b = 124$
		$n = 2^{32}$		$n = 2^{32}$		$n = 2^{32}$		$n = 2^{40}$
$r = 1$	s	16,777,216	s	65,536	s	256	s	256
	o	16,768,519	o	66,254	o	290	o	290
	χ^2	4.52604	χ^2	7.86639	χ^2	4.51563	χ^2	4.51563
	p	0.033383	p	0.005036	p	0.033587	p	0.033587
$r = 2$	s	16,777,216	s	65,536	s	256	s	256
	o	16,764,484	o	66,086	o	300	o	277
	χ^2	9.70003	χ^2	4.61585	χ^2	7.56250	χ^2	1.72266
	p	0.001843	p	0.031678	p	0.005960	p	0.189351
$r = 4$	s	16,777,216	s	65,536	s	256	s	256
	o	16,763,353	o	66,188	o	304	o	209
	χ^2	11.4999	χ^2	6.48667	χ^2	9.00000	χ^2	8.62891
	p	0.000696	p	0.010869	p	0.002700	p	0.003309
$r = 8$	s	16,777,216	s	65,536	s	256	s	256
	o	16,767,205	o	64,713	o	203	o	287
	χ^2	5.99701	χ^2	10.3354	χ^2	10.9727	χ^2	3.75391
	p	0.014330	p	0.001305	p	0.000925	p	0.052684

Under the null hypothesis, χ^2 obeys a chi-squared distribution with one degree of freedom. For each CubeHash variant tested, Table 1 shows the results of the program run with the largest value of the χ^2 statistic, including the expected number of successes s, the observed number of successes o, χ^2, and the significance (p-value) of χ^2.

In 1,238 of the 1,240 program runs the null hypothesis was not rejected, and in two of the program runs the null hypothesis was rejected, at a significance of 0.001. (These two runs are highlighted in Table 1.) However, we cannot conclude that CubeHash does not behave as a random mapping on the basis of these two runs, because at a significance of 0.001 we would expect the null hypothesis to be rejected about one in 1,000 times by chance even if the null hypothesis is true.

For each CubeHash variant, a Kolmogorov-Smirnov (K-S) test was performed to test the null hypothesis that the χ^2 values from that variant's program runs obey a chi-squared distribution with one degree of freedom. As one example, Figure 4 shows the expected and observed cumulative distributions for CubeHash1/127-512; plots for all variants are available at [17]. Table 2 lists the K-S statistic D and its significance for each variant. In all cases, the null hypothesis was not rejected.

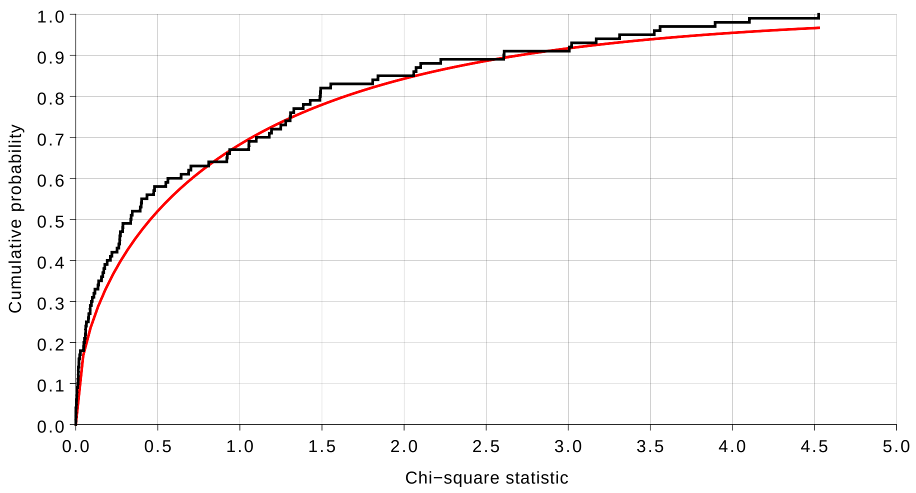

Figure 4: Expected (red) and observed (black) cumulative distributions of χ^2 for CubeHash1/127-512

Table 2: K-S Tests of χ^2 Distributions

		$b = 127$		$b = 126$		$b = 125$		$b = 124$
$r = 1$	D	0.081986	D	0.099036	D	0.091300	D	0.449424
	p	0.495247	p	0.266442	p	0.359208	p	0.023479
$r = 2$	D	0.093538	D	0.062532	D	0.051574	D	0.217311
	p	0.330395	p	0.817055	p	0.947710	p	0.676334
$r = 4$	D	0.126019	D	0.098237	D	0.068251	D	0.283495
	p	0.076778	p	0.275138	p	0.725416	p	0.339452
$r = 8$	D	0.092620	D	0.062538	D	0.071269	D	0.191768
	p	0.342026	p	0.816968	p	0.674332	p	0.813183

5.2 Number of Failures Before First Success

The single block attack program does a series of trials with $T = 0, 1, 2, \ldots$ and records the smallest T that yielded a second preimage. In other words, the series of trials yielded T failures in a row before the first success. Under the null hypothesis that CubeHash behaves as a random mapping, the probability of experiencing T failures followed by one success is

$$\Pr[T \text{ failures}] \;=\; (1-p)^T p \tag{2}$$

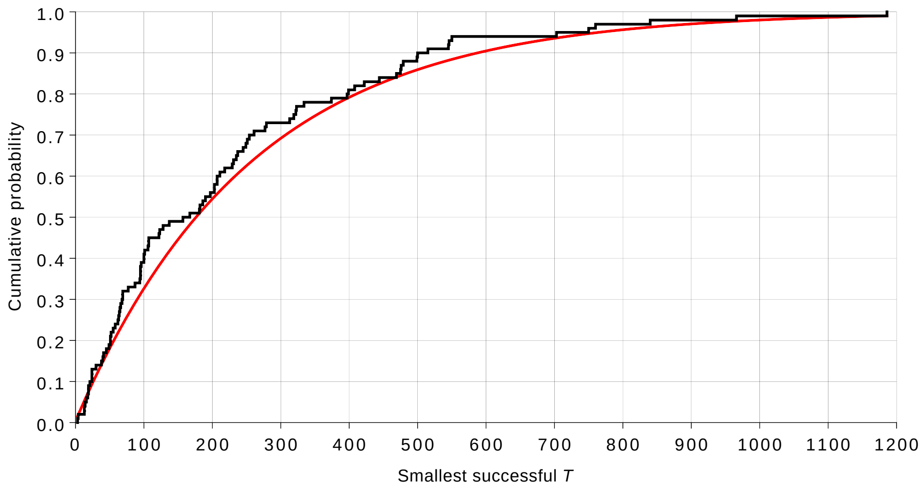

Figure 5: Expected (red) and observed (black) cumulative distributions of smallest successful T for CubeHash1/127-512

Table 3: K-S Tests of Smallest Successful T Distributions

		$b = 127$		$b = 126$		$b = 125$		$b = 124$
$r = 1$	D	0.105274	D	0.070612	D	0.080661	D	0.240533
	p	0.205371	p	0.685513	p	0.516467	p	0.547622
$r = 2$	D	0.066822	D	0.083771	D	0.083605	D	0.245708
	p	0.749153	p	0.467318	p	0.469881	p	0.519894
$r = 4$	D	0.047921	D	0.115872	D	0.111087	D	0.193748
	p	0.972475	p	0.127049	p	0.158740	p	0.803264
$r = 8$	D	0.089831	D	0.054503	D	0.067466	D	0.213803
	p	0.378969	p	0.920626	p	0.738502	p	0.695875

where $p = 2^{-8(128-b)}$ is the probability of success on each trial. The cumulative probability distribution is

$$\text{Pr}[\le T \text{ failures}] \;=\; \sum_{i=0}^{T}(1-p)^i p \;=\; 1-(1-p)^{T+1} \tag{3}$$

For each CubeHash variant, a K-S test was performed to test the null hypothesis that the smallest successful T values from that variant's program runs obey the above distribution. As one example, Figure 5 shows the expected and observed cumulative distributions for CubeHash1/127-512; plots for all variants are available at [17]. Table 3 lists the K-S statistic D and its significance for each variant. In all cases, the null hypothesis was not rejected.

6 Related Work

Most of the reported attacks on CubeHash rely on the specific structure of CubeHash's round function. These include attacks based on differential pathways in the round function [1, 2, 8, 9, 10, 11, 12, 13] and attacks based on symmetries in the internal state transformed by the round function [2].

In contrast, Bernstein (in the original CubeHash submission, [4]) reported *generic* collision and second preimage attacks that work for any hash function using an invertible round function and that do not rely on the hash function's specific structure. For CubeHash with a b-byte message block, the second preimage attack computes the initial state forward through the round function to an intermediate state S, computes the final state backward through the inverse round function to another intermediate state S', and looks for a collision in the last $128 - b$ bytes of S and S'. Khovratovich *et al.* [18] reported this same attack. Aumasson *et al.* [2] pointed out that this attack could look for collisions in the state after each step of the round function, not just between applications of the round function.

Like Bernstein's generic second preimage attack, our single block attack is a generic attack that works for any hash function using an invertible round function. However, the single block attack computes the final state backward through the inverse round function all the way to the initial state and looks for a match with the last $128 - b$ bytes of the IV, rather than meeting in the middle and looking for a collision. Because the single block attack looks for an exact match rather than a collision, the expected number of trials ($2^{8(128-b)}$) is large, but no memory is required (apart from storage for the CubeHash state itself). In contrast, a birthday-paradox collision search requires many fewer trials ($\sqrt{2^{8(128-b)}}$) but also requires copious memory. Furthermore, on a cluster parallel computer, the single block attack executes in a massively parallel fashion with almost no interprocessor communication (only one final reduction). A birthday-paradox collision search on a cluster parallel computer requires extensive interprocessor communication (see, for example, [6]).

7 Conclusion

We have reported an attack on the CubeHash one-way hash function that finds a second preimage consisting of a single message block. The attack requires minimal memory and was implemented as a massively parallel Java program running on a hybrid parallel computer. The attack requires less time than brute force search for reduced-strength CubeHash variants but does not break the variants recommended for SHA-3. From statistical tests based on the single block attack, we found no reason to disbelieve that CubeHash behaves as a random mapping. These results support CubeHash's viability as a secure cryptographic hash function.

References

[1] J.-P. Aumasson. Collision for CubeHash2/120-512. NIST hash-forum mailing list, December 4, 2008. http://ehash.iaik.tugraz.at/uploads/a/a9/Cubehash.txt

[2] J.-P. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia, and T. Peyrin. Inside the hypercube. In *14th Australasian Conference on Information Security and Privacy (ACISP 2009),* July 2009.

[3] D. Bernstein. CubeHash specification (2.B.1). Extracted from CubeHash submission to the NIST SHA-3 Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/CubeHash.zip

[4] D. Bernstein. CubeHash appendix: complexity of generic attacks. Extracted from CubeHash submission to the NIST SHA-3 Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/CubeHash.zip

[5] D. Bernstein. CubeHash parameter tweak: 16 times faster. July 15, 2009. http://cubehash.cr.yp.to/submission/tweak.pdf

[6] D. Bernstein, T. Lange, C. Peters, R. Niederhagen, and P. Schwabe. FSBday: Implementing Wagner's generalized birthday attack against the SHA-3 candidate FSB. Cryptology ePrint Archive Report 2009/292, June 17, 2009. http://eprint.iacr.org/2009/292

[7] B. Bloom and S. Janis. Inference attacks on CubeHash. Rochester Institute of Technology, Department of Computer Science, Parallel Computing II class project, May 13, 2009. http://www.cs.rit.edu/~ark/spring2009/736/team/1/

[8] E. Brier, S. Khazaei, W. Meier, and T. Peyrin. Attack for CubeHash-2/2 and collision for CubeHash-3/64. NIST hash-forum mailing list, February 3, 2009. http://ehash.iaik.tugraz.at/uploads/3/3a/Peyrin_ch22_ch364.txt

[9] E. Brier, S. Khazaei, W. Meier, and T. Peyrin. Real collisions for CubeHash-4/64. NIST hash-forum mailing list, July 6, 2009. http://ehash.iaik.tugraz.at/uploads/9/93/Bkmp_ch464.txt

[10] E. Brier, S. Khazaei, W. Meier, and T. Peyrin. Real collisions for CubeHash-4/48. NIST hash-forum mailing list, July 16, 2009.

[11] E. Brier and T. Peyrin. Cryptanalysis of CubeHash. In *International Conference on Applied Cryptography and Network Security (ACNS '09),* June 2009.

[12] W. Dai. Collisions for CubeHash1/45 and CubeHash2/89. December 26, 2008. http://www.cryptopp.com/sha3/cubehash.pdf

[13] W. Dai. Collision for CubeHash2/12. January 17, 2009. `http://www.cryptopp.com/sha3/cubehash2.pdf`

[14] ECRYPT Benchmarking of Cryptographic Systems: Measurements of hash functions. Version 2009.04.22. `http://bench.cr.yp.to/results-hash.html`

[15] A. Kaminsky. Parallel Java: A unified API for shared memory and cluster parallel programming in 100% Java. In *21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007),* March 2007.

[16] A. Kaminsky. Parallel Java Library. `http://www.cs.rit.edu/~ark/pj.shtml`

[17] A. Kaminsky. Single block attacks and statistical tests on CubeHash web site. `http://www.cs.rit.edu/~ark/parallelcrypto/cubehash01/`

[18] D. Khovratovich, I. Nikolić, and R.-P. Weinmann. Preimage attack on CubeHash512-$r/4$ and CubeHash512-$r/8$. 2008. `http://ehash.iaik.tugraz.at/uploads/6/6c/Cubehash.pdf`