Cube Test Analysis of the Statistical Behavior of CubeHash and Skein

Alan Kaminsky[*]

May 6, 2010

Abstract

This work analyzes the statistical properties of the SHA-3 candidate cryptographic hash algorithms CubeHash and Skein to try to find nonrandom behavior. Cube tests were used to probe each algorithm's internal polynomial structure for a large number of choices of the polynomial input variables. The cube test data were calculated on a 40-core hybrid SMP cluster parallel computer. The cube test data were subjected to three statistical tests: balance, independence, and off-by-one. Although isolated statistical test failures were observed, the balance and off-by-one tests did not find nonrandom behavior overall in either CubeHash or Skein. However, the independence test *did* find nonrandom behavior overall in both CubeHash and Skein.

1 Introduction

NIST inaugurated the Cryptographic Hash Algorithm Competition [10, 11] in November 2007 to select the next U.S. government standard hash function, SHA-3. One of the criteria NIST will use to evaluate the candidate hash algorithms is: "Hash algorithms will be evaluated against attacks or observations that may threaten existing or proposed applications, or demonstrate some fundamental flaw in the design, such as exhibiting nonrandom behavior and failing statistical tests." [11] While numerous cryptographic attacks against the candidate hash algorithms have been published, little attention has been paid to evaluating the candidate hash algorithms' statistical behavior; see [4] for one study.

Dinur and Shamir [5] reported the *cube attack* on a cryptographic primitive, which succeeds in a practical amount of time if the primitive can be represented as a low-degree polynomial in $GF(2)$. Specifically, the cube attack's complexity is $2^{d-1}n + n^2$, where d is the polynomial's degree and n is the number of secret bits to be recovered (such as a block cipher key).

Subsequently, Aumasson, Dinur, Meier, and Shamir [1] reported the *cube test* on a cryptographic primitive. Rather than recovering a secret key or otherwise attacking the primitive, the cube test probes the primitive's internal polynomial structure and can be used to analyze the primitive's statistical behavior. Furthermore, the cube test can be applied to any cryptographic primitive, not just one represented by a low-degree polynomial. Aumasson *et al.* applied cube tests to the Trivium stream cipher and the MD6 hash function, a SHA-3 Round 1 candidate.

Continuing the line of research suggested in [1], this paper reports the results of an analysis of the statistical behavior of two of the SHA-3 Round 2 candidates, CubeHash and Skein. Section 2 describes the cube test methodology used in the analysis and describes the massively parallel Java program that generated the cube test data. Section 3 introduces the CubeHash and Skein algorithms and describes how the cube test methodology was applied to them. Section 4 describes the statistical tests performed on the cube test data. Section 5 presents the statistical test results on CubeHash and Skein. Section 6 offers concluding remarks.

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

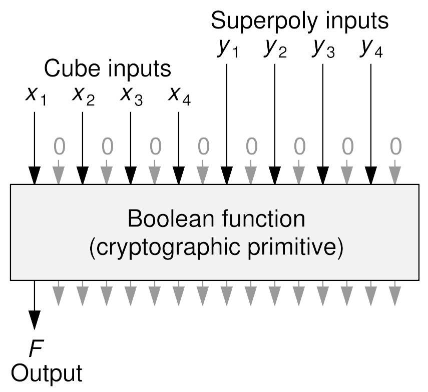


Figure 1: Cube test of one output bit of a cryptographic primitive

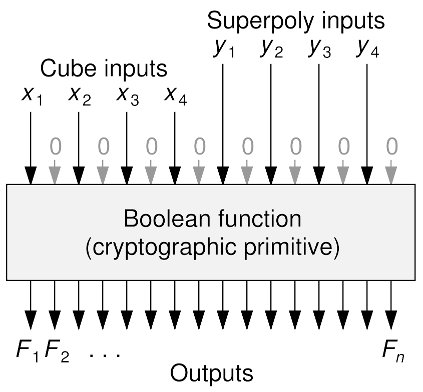


Figure 2: Cube test of all output bits of a cryptographic primitive

2 Cube Tests

2.1 Cube Test Methodology

Consider a cryptographic primitive, such as a hash function, to be a Boolean function with multiple input bits and output bits (Figure 1). Following the terminology of [1], some number c of the input bits are designated as a vector of *cube inputs* $\mathbf{x} = (x_1, x_2, \ldots, x_c)$, and some number s of the input bits are designated as a vector of *superpoly inputs* $\mathbf{y} = (y_1, y_2, \ldots, y_s)$. All input bits other than the cube inputs and superpoly inputs are set to 0. Then a particular output bit F can be treated as a $GF(2)$ polynomial function of the cube inputs and superpoly inputs: $F(\mathbf{x}, \mathbf{y})$. Now, express F as follows:

$$F(\mathbf{x}, \mathbf{y}) = x_1 x_2 \cdots x_c Q(\mathbf{y}) + R(\mathbf{x}, \mathbf{y}) \tag{1}$$

(Note that, in $GF(2)$, multiplication is the same as Boolean "and," and addition is the same as Boolean "exclusive-or.") The first part of the right hand side of (1) consists of the terms in the polynomial F that include *all* the cube inputs plus one or more superpoly inputs. The cube inputs are factored out, leaving a polynomial Q in just the superpoly inputs. The second part of the right hand side of (1) consists of the remaining terms in the polynomial F, which is another polynomial R in the cube and superpoly inputs. The polynomial Q is called the *superpoly* of F with respect to the cube inputs $\mathbf{x}$.

The superpoly $Q(\mathbf{y})$ can be calculated by the following *summation procedure* [5], without even knowing the polynomial formula for Q, as long as the overall Boolean function F can be evaluated:

```
Set unused inputs of $F$ to 0
Set superpoly inputs of $F$ to $\mathbf{y}$
$Q \leftarrow 0$
For each $\mathbf{x}$ from $00\ldots00$ to $11\ldots11$:
    $Q \leftarrow Q + F(\mathbf{x}, \mathbf{y})$
Return $Q$
```

Proof. In the summation of F over the 2^c values of $\mathbf{x}$, each term in R is added in an even number of times, since no term in R contains all of x_1 through x_c. Therefore, in $GF(2)$ arithmetic, the terms in R sum up to 0. The terms in Q, however, are added in only once, when $\mathbf{x} = 11\ldots11$. Therefore, the summation yields just Q. □

The cube test is based on the above summation procedure. The null hypothesis is that the cryptographic primitive is a random polynomial. Therefore, for any particular choice of cube and superpoly inputs, the superpoly Q is also a random polynomial. Evaluate Q for some number of randomly chosen values for the superpoly inputs, and apply a statistical test to the resulting series of superpoly output values. If the statistical test fails at a designated significance level, then the null hypothesis is disproved, Q is not a random polynomial, and the cryptographic primitive exhibits nonrandom behavior. Testing one or more superpolys might reveal nonrandom behavior where testing the cryptographic primitive as a whole might not reveal nonrandom behavior.

Calculating the superpoly Q requires calculating the whole cryptographic primitive, which yields n output bits, not just one. Each output bit is a different polynomial function of the cube and superpoly inputs: $F_1, F_2, \ldots, F_n$ (Figure 2). Thus, the cube test actually tests multiple superpolys $Q_1, Q_2, \ldots, Q_n$ for nonrandomness.

2.2 Parallel Cube Test Program

The summation procedure performs 2^c evaluations of F. To speed up the calculation, the evaluations of F can be performed in a massively parallel fashion, followed by a parallel reduction to exclusive-or the F values, yielding Q.

Figure 3 depicts the design of the parallel program that calculates the superpoly values for the cube test. The program is written in Java using the author's Parallel Java Library [7, 8]. The program is designed to run on a hybrid parallel computer with multiple compute nodes connected in a cluster via a high-speed network, each node a shared memory multiprocessor (SMP) machine with multiple CPUs. The program consists of multiple processes, one process running on each node. Each process in turn consists of multiple threads, one thread running on each CPU and sharing memory with the other threads in the process.

Figure 4 describes the program's operation. The program computes the superpolys for m randomly chosen superpoly inputs $\mathbf{y}$ in parallel, partitioning the superpoly input samples among the processes. For each superpoly input sample, the program performs the evaluations of $F(\mathbf{x}, \mathbf{y})$ in parallel, partitioning the 2^c cube inputs $\mathbf{x}$ among the threads in the process. The results are written to a cube test data file for later analysis.

The cube test program takes as an input the name of the Java class for the *target,* the cryptographic primitive being tested. Instances of the target class are created using Java reflection to do the calculations of F. By defining an appropriate subclass of the base class Target, the program can perform cube tests on any cryptographic primitive (or any Boolean function).

Besides the m random samples of the superpoly input values, the program chooses additional superpoly input values if necessary such that every sample that differs from any original sample in one bit position is included. For example, if 101110 is one of the m original superpoly input samples ($s = 6$), the program ensures that superpoly input samples 001110, 111110, 100110, 101010, 101100, and 101111 are also evaluated. This is needed to perform the off-by-one statistical test (see Section 4.4).

3 Hash Algorithms

3.1 CubeHash

Invented by Bernstein [2], the CubeHash algorithm 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 ($1 \leq b \leq 128$); and h, the size of the hash value in bits ($h = 8, 16, 24, \ldots, 512$). "CubeHash$r/b$-$h$" denotes the CubeHash variant with parameter choices r, b, and h. Bernstein recommends the following settings for SHA-3: CubeHash16/32-224, CubeHash16/32-256, CubeHash16/32-384, and CubeHash16/32-512 [3]. This paper analyzes just the last variant, CubeHash16/32-512.

The CubeHash16/32-512 algorithm was implemented in Java as a subclass of the base class Target for the cube test program. The input is a single 32-byte message block, consisting of 31 bytes of data followed by one required padding byte of 0x80; thus, the CubeHash target has 248 input bits from which to choose the cube inputs and superpoly inputs. The CubeHash target has 512 output bits, yielding superpolys Q_1 through Q_{512}.

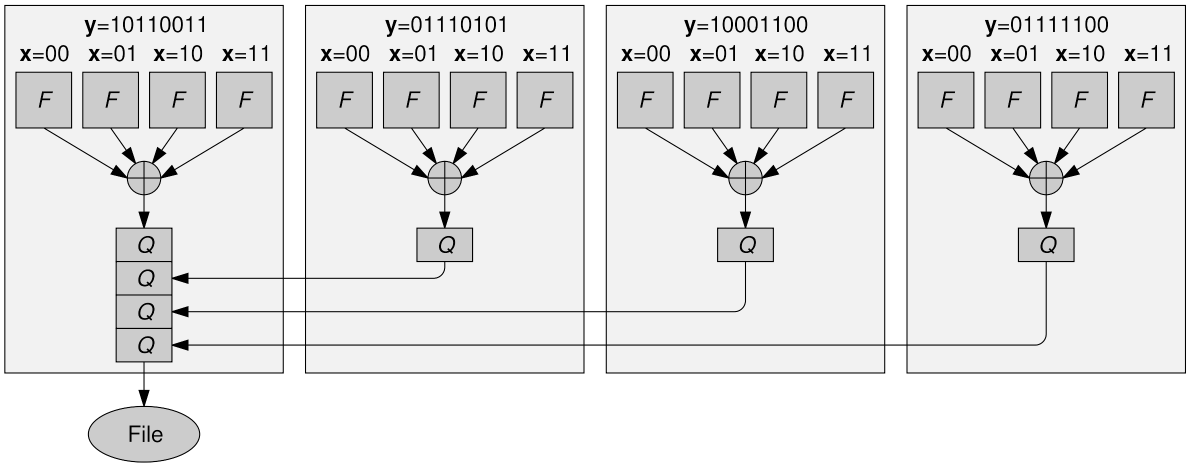


Figure 3: Parallel cube test program design

Inputs: Target class name for computing cryptographic primitive F with n output bits
Number of cube inputs, c
Number of superpoly inputs, s
Number of random superpoly samples, m

Output: Cube test data file

```
Choose $c$ input bits at random to be cube inputs $\mathbf{x}$
Choose $s$ input bits at random to be superpoly inputs $\mathbf{y}$
**parallel for** $i = 0$ to $m - 1$        — partitioned equally among the processes
    $\mathbf{y} \leftarrow \mathrm{random}(0, 2^s - 1)$
    **parallel for** $\mathbf{x} = 0$ to $2^c - 1$        — partitioned equally among the threads in the process
        Compute $F_1(\mathbf{x}, \mathbf{y}), \ldots, F_n(\mathbf{x}, \mathbf{y})$
    **end parallel for**
    $Q_1(\mathbf{y}), \ldots, Q_n(\mathbf{y}) \leftarrow \sum_{\mathbf{x}} F_1(\mathbf{x}, \mathbf{y}), \ldots, \sum_{\mathbf{x}} F_n(\mathbf{x}, \mathbf{y})$        — shared memory parallel reduction
    Send $(\mathbf{y}, Q_1(\mathbf{y}), \ldots, Q_n(\mathbf{y}))$ to one process        — message passing
**end parallel for**
Write all $(\mathbf{y}, Q_1(\mathbf{y}), \ldots, Q_n(\mathbf{y}))$ data to cube test data file
```

Figure 4: Parallel cube test program operation

3.2 Skein

Invented by Ferguson, Lucks, Schneier, Whiting, Bellare, Kohno, Callas, and Walker [6], the Skein algorithm has three variants, with internal state sizes of 256, 512, and 1024 bits. The Skein algorithm can produce a hash value of any size. (The hash size can even be larger than the internal state size, although the hash value has no more entropy than the internal state.) "Skein-s-h" denotes the Skein variant with an s-bit internal state and an h-bit hash value. Ferguson *et al.* recommend the following variants of Skein for SHA-3: Skein-256-224, Skein-512-224, Skein-256-256, Skein-512-256, Skein-512-384, Skein-1024-384, Skein-512-512, and Skein-1024-512 [6]. This paper analyzes just the Skein-512-512 variant.

The Skein-512-512 algorithm was implemented in Java as a subclass of the base class Target for the cube test program. The input is a single 64-byte message block, consisting of 31 bytes of data followed by 33 padding bytes of 0. Thus, the Skein target has 248 input bits and 512 output bits, just like the CubeHash target.

4 Statistical Tests

Each run of the cube test program samples a group of randomly chosen superpolys of the target hash function (CubeHash or Skein), Q_1 through Q_{512}, one superpoly for each bit of the output hash value. The superpolys are defined by choosing c cube inputs and s superpoly inputs at random. m samples of the superpoly input values are chosen at random and the superpolys are calculated, yielding a series of M samples for each superpoly Q_1 through Q_{512}. (The results in Section 5 use $M = \min(100, 2^S)$.) These superpoly samples were subjected to three statistical tests—balance test, independence test, and off-by-one test—to attempt to disprove the null hypothesis that the target hash function is a random polynomial.

4.1 Chi-Square Test

Each of the three statistical tests is a *chi-square test.* The chi-square test categorizes the series of random values being tested into discrete *bins* and counts the occurrences in each bin. The χ^2 statistic is

$$\chi^2 = \sum_{i=1}^{b} \frac{(N_i - E_i)^2}{E_i} \tag{2}$$

where b is the number of bins, N_i is the observed count in the i-th bin, and E_i is the expected count in the i-th bin if the null hypothesis is true. Typically, the E_i values are derived from the total of the counts in all the bins.

The χ^2 statistic obeys a chi-square distribution with d degrees of freedom. When the E_i values are determined as described above, d is $b-1$. The *significance* is the probability that a statistic greater than or equal to χ^2 would be observed by chance when the null hypothesis is true. The significance is 1 minus the cumulative distribution function of the chi-square distribution:

$$\text{Significance}(d, \chi^2) = 1 - P\left(\frac{d}{2}, \frac{\chi^2}{2}\right) \tag{3}$$

where $P(a, x)$ is the *incomplete gamma function:*

$$P(a, x) = \frac{\displaystyle\int_0^x t^{a-1} \mathrm{e}^{-t} \mathrm{d}t}{\displaystyle\int_0^\infty t^{a-1} \mathrm{e}^{-t} \mathrm{d}t} \tag{4}$$

Formulas for calculating $P(a, x)$ are well-known; see [12], for example.

As the observed counts N_i deviate farther from the expected counts E_i, χ^2 increases and the significance decreases. If the significance falls below a certain threshold p, the statistical test *fails* (the null hypothesis is disproved at a significance of p), otherwise the statistical test *passes.* Statistical tests of cryptographic pseudorandom number generators typically use p in the range 0.01 to 0.001 [13]. The results in Section 5 use $p = 0.001$. (The Conclusion will discuss the rationale for this choice.)

4.2 Balance Test

Under the null hypothesis that the cryptographic primitive is a random polynomial, each superpoly should behave like a fair coin. Over all the input samples, half the time the output should be 0 and half the time the output should be 1. The counts for the chi-square test of superpoly Q_i are N_1 = observed number of zeroes in the series of m values for superpoly Q_i, N_2 = observed number of ones, and $E_1 = E_2 = m/2$. Applying the balance test to one cube test program run yields 512 pass/fail results.

4.3 Independence Test

Under the null hypothesis, each pair of superpolys should behave like two independent fair coins. Therefore, over all the input samples, one-fourth the time the pair of outputs should be (0,0), and likewise for (0,1), (1,0), and (1,1). The counts for the chi-square test of superpoly pair (Q_i, Q_j) are N_1 = observed number of (0,0) pairs in the series of m values for superpolys Q_i and Q_j, N_2 = observed number of (0,1) pairs, N_3 = observed number of (1,0) pairs, N_4 = observed number of (1,1) pairs, and $E_1 = E_2 = E_3 = E_4 = m/4$. Applying the independence test to one cube test program run yields $512 \cdot 511/2 = 130,816$ pass/fail results, one for each pair of superpolys.

4.4 Off-By-One Test

Under the null hypothesis, over all the input samples, when one of the superpoly input bits is flipped from 0 to 1 or 1 to 0, half the time the output bit should also flip and half the time the output bit should not flip. The counts for the chi-square test of output Q_i and input s_j are N_1 = observed number of times Q_i did not flip when s_j flipped, N_2 = observed number of times Q_i flipped when s_j flipped, and $E_1 = E_2 = m/2$. Applying the off-by-one test to one cube test program run yields $512 \cdot s$ pass/fail results, one for each combination of a superpoly output and a superpoly input.

One subtlety in the off-by-one test is that the same occurrence must not be counted twice. For example, suppose two of the m superpoly input samples happen to be 101110 and 001110 ($s = 6$). Flipping the first bit in the first sample will cause the output bits to flip or not flip in the same way as flipping the first bit in the second sample. Thus, the outcomes from flipping the first bit for these two samples are not independent. In each such case, the number of samples will be reduced by 1.

4.5 Summary Statistics

Applying one of the statistical tests to one cube test program run on the cryptographic primitive yields multiple pass/fail results. To summarize the primitive's behavior for that program run, a further statistical test is applied. t samples of the pass/fail results are chosen at random. (The results in Section 5 use t = 512 for the balance test and t = 1,000 for the independence and off-by-one tests.) If the null hypothesis is true, the pass/fail results should behave like a biased coin, with a probability $(1 - p)$ of a pass result and a probability p of a fail result. A chi-square test is used to test this hypothesis, with N_1 = observed number of pass results, N_2 = observed number of fail results, $E_1 = (1 - p) \cdot t$, and $E_2 = p \cdot t$.

If the significance of the summary χ^2 statistic falls at or below a threshold p, the null hypothesis is considered to be disproven, and the cryptographic primitive is considered to display nonrandom behavior, for that cube test program run. (The results in Section 5 use $p = 0.001$, the same threshold as for the individual statistical tests.)

4.6 Statistical Test Program

A Java program performs the statistical tests on the cube test program run data. The program takes as an input the name of the Java class for the *analyzer* that embodies the statistical test. Instances of the analyzer class are created using Java reflection to do the actual analysis. By defining an appropriate subclass of the base class Analyzer, the program can apply any statistical test to the cube test results.

Figure 5 shows an example of the analysis program's output. Each line of output shows the total number of statistical tests that passed and failed for all cube test program runs with a certain choice of the number of superpoly inputs s and the number of cube inputs c, along with the summary χ^2 statistic and its significance.

```
S       C       Pass    Fail    Chi^2           P
2       2       2048    0       2.050050        0.152201
2       3       2048    0       2.050050        0.152201
2       4       2048    0       2.050050        0.152201
2       5       2048    0       2.050050        0.152201
2       6       2048    0       2.050050        0.152201
. . .
24      20      2045    3       0.442974        0.505690
24      21      2046    2       0.001126        0.973230
24      22      2048    0       2.050050        0.152201
24      23      2044    4       1.862362        0.172352
24      24      2047    1       0.536818        0.463754
```

Figure 5: Example of analysis results on cube test program runs

5 Results

The cube test program runs were done on a 10-node hybrid parallel computer, each node with two AMD Opteron 2218 dual-core CPUs, 2.6 GHz clock, and 8 GB main memory; 40 CPUs total. The nodes are interconnected with a dedicated 1 Gbps switched Ethernet. Each run consisted of one choice of each of the following:

- Cryptographic primitive (target): CubeHash16/32-512 or Skein-512-512
- Number of cube inputs $c = 2, 3, \ldots, 23, 24$
- Number of superpoly inputs $s = 2, 3, \ldots, 23, 24$
- Four different random seeds

Altogether, $23 \cdot 23 \cdot 4 = 2,116$ runs were performed on CubeHash and 2,116 runs were performed on Skein. Each run computed $m = \min(100, 2^s)$ superpoly input samples, plus additional samples needed for the off-by-one test (see Section 2.2). Each superpoly input sample required 2^c evaluations of the cryptographic primitive.

A total of 3,606,910,695,720 CubeHash16/32-512 evaluations were performed (a bit less than 2^{42} evaluations). The evaluations took 1.25×10^6 seconds. Computing the CubeHash16/32-512 hash of a single message block took 346 nsec on 40 CPUs, which is equivalent to 13.9 μsec on one CPU. (This includes the overhead of the cube test program as well as the actual hash computation.)

A total of 3,603,992,046,760 Skein-512-512 evaluations were performed. The evaluations took 3.30×10^5 seconds. Computing the Skein-512-512 hash of a single message block took 91.7 nsec on 40 CPUs, which is equivalent to 3.67 μsec on one CPU.

The cube test program results were subjected to the balance, independence, and off-by-one tests with parameters $m = 100$ random superpoly input samples, $t = 512$ random pass/fail result samples for the balance test, $t =$ 1,000 random pass/fail result samples for the independence and off-by-one tests, and significance p $= 0.001$.

Figure 6 depicts the balance test summary statistics for CubeHash16/32-512 and Skein-512-512. Likewise, Figure 7 depicts the independence test summary statistics and Figure 8 depicts the off-by-one test summary statistics. In the visualizations, each grid square represents the cube test program runs for a certain number of superpoly inputs s and a certain number of cube inputs c. Each grid square's gray shade represents the significance of the summary χ^2 statistic, with black being a significance of 0.0 and white being a significance of 1.0. Thus, darker gray shades are more nonrandom. If a significance falls at or below the threshold p $= 0.001$, the grid square is marked with a white dot. Thus, white dots appear where the statistical tests detected individual cube test program runs with nonrandom behavior.

If the null hypothesis is true, that the cryptographic primitive is a random polynomial, then the probability of a white dot appearing in a grid square is the significance threshold $p = 0.001$. Thus, each visualization

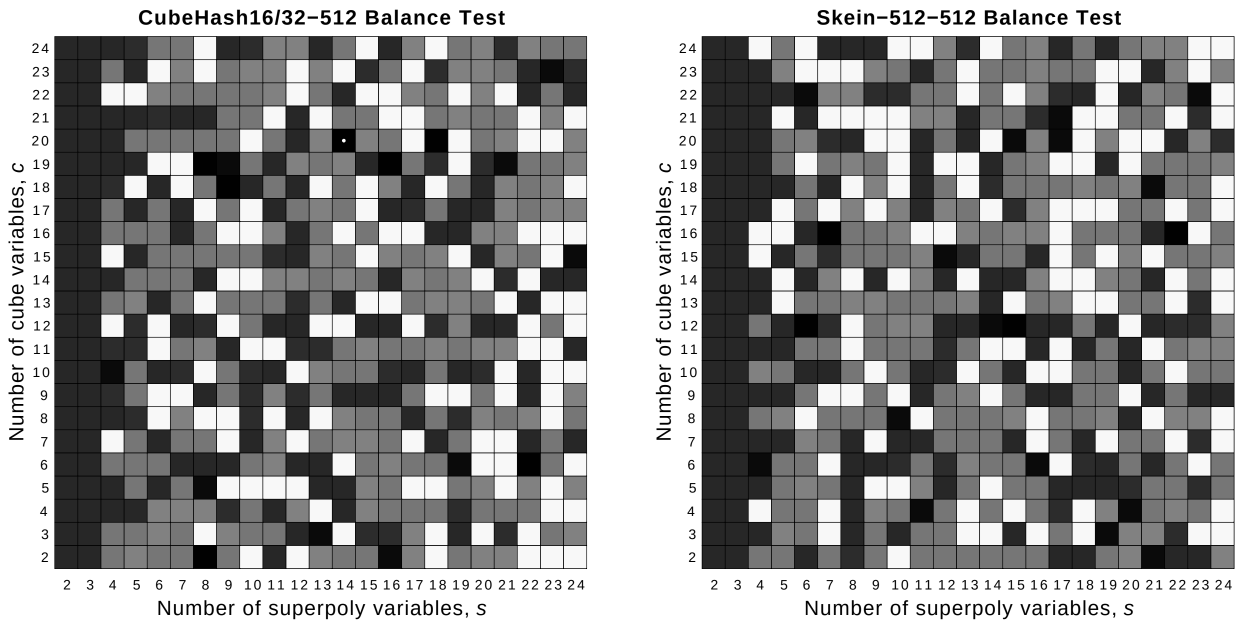


Figure 6: Balance test summary statistics for CubeHash16/32-512 and Skein-512-512

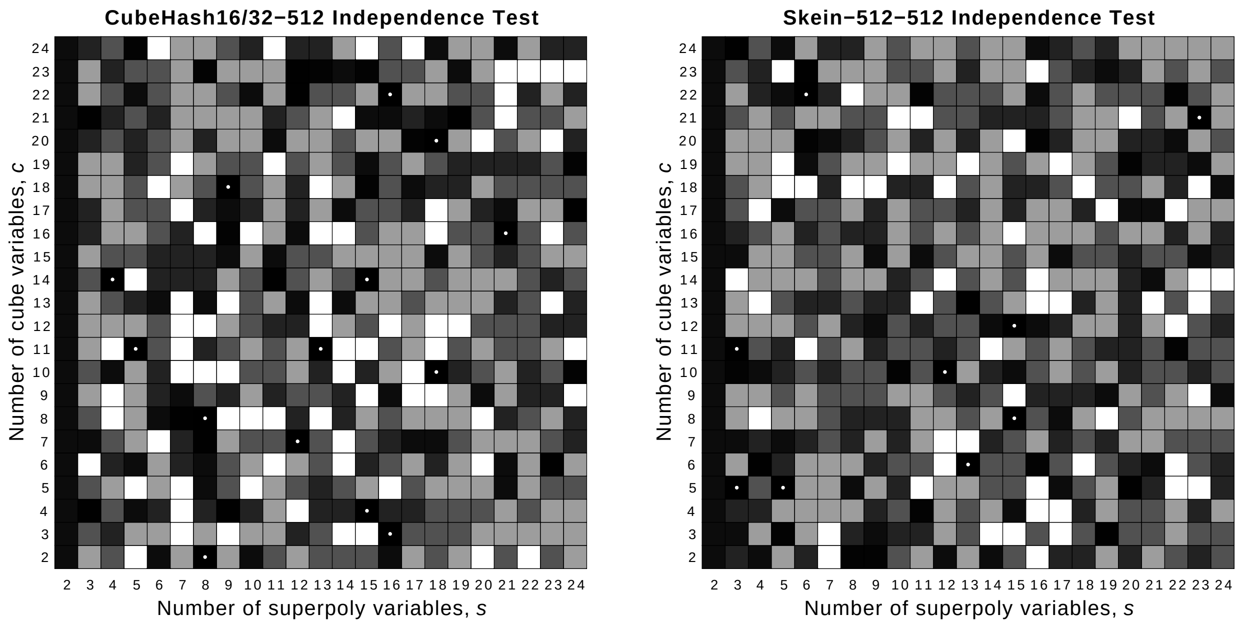
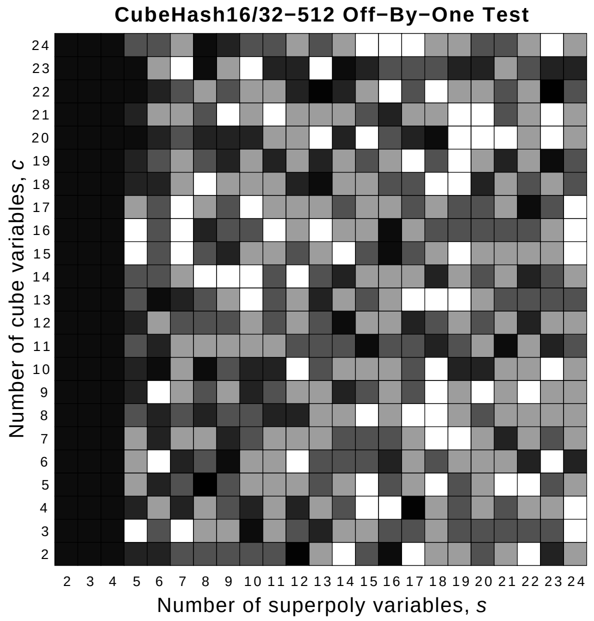


Figure 7: Independence test summary statistics for CubeHash16/32-512 and Skein-512-512



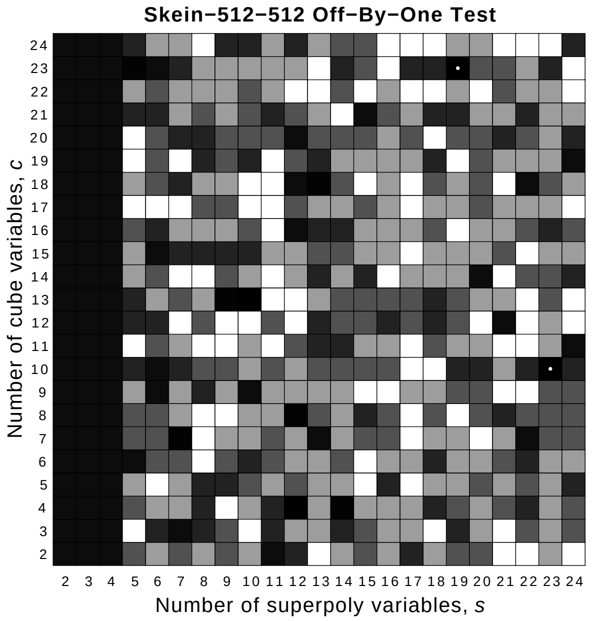


Figure 8: Off-by-one test summary statistics for CubeHash16/32-512 and Skein-512-512

Table 1: Instances of nonrandom behavior detected for various seeds

Test	*Hash Function*	*Seed* = 142857	*Seed* = 285714	*Seed* = 428571	*Seed* = 571428
Balance	CubeHash16/32-512	1	1	1	1
	Skein-512-512	0	0	0	0
Independence	CubeHash16/32-512	14	16	16	14
	Skein-512-512	9	11	12	14
Off-by-one	CubeHash16/32-512	0	0	1	0
	Skein-512-512	2	2	0	1

can be viewed as consisting of 529 flips of a biased coin, and a chi-square test can determine the probability that the null hypothesis is true given a certain number of white dots. If there are three or more white dots, the probability that the null hypothesis is true is less than 0.001. In this case, the statistical test detects *overall* nonrandom behavior in the cryptographic primitive, not just nonrandom behavior for isolated cube test program runs.

For CubeHash16/32-512, the balance test does not detect overall nonrandom behavior (one white dot), and the off-by-one test does not detect overall nonrandom behavior (no white dots). However, the independence test does detect overall nonrandom behavior (14 white dots).

For Skein-512-512, the balance test does not detect overall nonrandom behavior (no white dots), and the off-by-one test does not detect overall nonrandom behavior (two white dots). However, the independence test does detect overall nonrandom behavior (9 white dots).

For the independence and off-by-one tests, the significance (gray shade) of each grid square depends on the choice of pseudorandom number generator seed when the statistical analysis program is run, because the pseudorandom number sequence determines which individual statistical test pass/fail results are sampled to compute the summary χ^2 statistic. (This does not pertain to the balance test because all 512 pass/fail results are included in the summary χ^2 statistic.) Table 1 shows the number of instances of nonrandom behavior (white dots) detected for various seeds. Even when different seeds were chosen, the independence test consistently detected overall nonrandom behavior both in CubeHash16/32-512 and in Skein-512-512.

6 Conclusion

This study has shown that when the independence test is applied to selected superpolys inside CubeHash and Skein, the null hypothesis that CubeHash or Skein is a random polynomial is disproved at a significance level of 0.001. In other words, nonrandom behavior was detected in both CubeHash and Skein. The balance and off-by-one tests did not disprove the null hypothesis.

Why choose a significance level of 0.001? When p is larger, say 0.01, the chi-square tests are more stringent; smaller differences between the observed and expected counts will cause the significance to fall below the threshold and the test to fail. But this means a “false failure,” where the test fails even though the function really is random, is more likely. On the other hand, when p is smaller, say 0.001, the chi-square tests are more lenient; larger differences between the observed and expected counts are required to cause the significance to fall below the threshold and the test to fail. But this means a “false pass,” where the test passes even though the function really is nonrandom, is more likely. This study used the more lenient significance level, $p = 0.001$. Even so, nonrandom behavior was still detected, although in only one of the three statistical tests.

Future work includes running the cube tests on other SHA-3 candidate hash algorithms. Because of the long running times involved, even on a parallel computer, this may have to wait until the SHA-3 Competition progresses to Round 3 and only a few candidate hash algorithms remain. Future work also includes applying additional statistical tests to the cube test results.

The cube test program output files, analysis program output files, and Java source code for all the programs described herein are available [9].

Acknowledgments

I would like to thank Stanisław Radziszowski for helpful comments on an earlier draft of this paper.

References

[1] J. Aumasson, I. Dinur, W. Meier, and A. Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. *Fast Software Encryption,* 2009.

[2] 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`

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

[4] B. Bloom and A. Kaminsky. Single block attacks and statistical tests on CubeHash. Cryptology ePrint Archive, Report 2009/407, August 21, 2009.

[5] I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. Cryptology ePrint Archive, Report 2008/385, January 26, 2009.

[6] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein hash function family, version 1.2. September 15, 2009. `http://www.skein-hash.info/sites/default/files/skein1.2.pdf`

[7] 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.

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

[9] A. Kaminsky. Cube test analysis of the statistical behavior of CubeHash and Skein web site. `http://www.cs.rit.edu/~ark/parallelcrypto/cubetest01/`

[10] NIST Cryptographic Hash Algorithm Competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/`

[11] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. *Federal Register,* 72(212):62212–62220, November 2, 2007.

[12] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes: The Art of Scientific Computing, Third Edition.* Cambridge University Press, 2007.

[13] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, and L. Bassham. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST Special Publication 800-22, Revision 1a.* April 2010.