SHA-256 Limited Statistical Analysis

Dr. Russell J. Davis
Femtosecond Inc.
9747 Water Oak Drive
Fairfax, VA 22031-1029
RDavis@femto-second.com

Abstract

This paper attempts to infer Secure Hash Algorithm (SHA) weaknesses without actually identifying the root cause. By examining the message digest generated from a given hash function statistical patterns are examined that could indicate algorithm weaknesses. This paper presents an analysis of the SHA-256 algorithm by analyzing 2-bit distributions extracted from the message digests. The test approach presented uses the Statistical Process Control (SPC) Threshold to identify those 2-bit samples indicative of a process out of control.

Test Approach

The approach taken was to code a sequential counter and then calculate a new hash for each incremented value. Consider that the SHA-256 produces a 256-bit message digest. Let the least significant bit be bit-location 0 and the most significant bit, be location 255. Then all bits within the message digest can be represented by their location within the message digest.

Each resulting message digest was mapped into 2-bit values. Using a [*i*,*j*] representation for each 2-bit pair within the 256-bit message digest, each bit represents the positional location (0, .., 255) within the message digest. As a further restriction, the first and second bits were not allowed to be the same. That is, let *i* represent the most significant bit and *j*, the lease significant bit; where $0 \le i \le 255$, $0 \le j \le 255$, and $i \ne j$. To keep track of the values, a 64K array was used to hold the (256*255 or 65280) possible SHA-256 2-bit values).

The reason for examining the 2-bit patterns was to examine the message digests bit patterns that need not be adjacent. Additionally, the standard deviations associated with the 2-bit ordering provided information unavailable when examining only single bit results. Thus, the entire hash value was examined for 2-bit pairs that were unusually far from the average of all samples. Next, runs of 10 million hashes were calculated over incremented values and each 2-bit sample accumulated. That is, a starting point was selected, hash value calculated over the incremented value, and the counter incremented. Given the value for each sample was between 0 and 3, the expected value was 1.5. For each of 10 million hashes, the 2-bit value was accumulated. So the expected accumulated sum was 15 million for each of the 65,280 2-bit pairs. Each of these bit pairs represented one accumulated sample measured against the SPC Threshold

Next, the average for all samples was calculated along with the standard deviation, σ. The standard deviation provided a measurement of how close the samples were to the average.

Statistical Process Control

One technique often used within quality control is the Statistical Process Control (SPC) Threshold. This is defined as the mean plus (or minus) three standard deviations.

$$SPCThreshold = \overline{x} \pm 3\sigma$$

The *SPC Threshold* can be plotted above and below (using -3σ for the lower Threshold) the average value. What we were interested in determining was the number (if any) samples that exceed the SPC Threshold. In other industries that utilize the SPC Threshold as a quality control measurement, samples exceeding this value are indicative of a "process out of control." The approach described in this paper was to identify how many samples were considered a "process out of control." To improve the overall sampling, 8 different starting points and/or initializations were selected. Note: SPC does not identify the root problem it only indicates that one (or more) exists. Changes to the existing processes (or algorithms as in this case) can be re-examined to see if there are still processes out of control. The approach presented builds on the established SPC measurement approach. Although only the SHA-256 is discussed in this paper, the approach could be applied to any hash algorithm.

Eight test run configurations were prepared. Given the large number of message digests to analyze, the hashed data was constructed to fit into a single block (SHA-256 uses 512-bit block sizes). The test fixture then could use the same padding, substitute the new value, and then calculate the new hash value. The box below summarized the eight test runs. Each of the 10 million hash runs included calculating 65,280 samples. For the first two runs, two different starting locations were selected. A 32-bit unsigned integer has over 4 billion possible values. The first run initialized the counter to a starting value of 10,000,000. It was assumed that during the algorithm development, starting values of 0 were likely already tested. For run 2, the starting location was selected just under 10 million. Note: the test fixture used unsigned integers. The negative number shown below is so the reader can quickly determine where in the value range the initial value was selected. For the remaining six runs, the size of the hashed value was 64-bits. Moreover, the counter was placed in the most significant (w[1]) 32-bit unsigned integer. So the starting points shown are with respect to the upper 32-bits. During some tests, the least significant 32-bit unsigned integer (w[0]) was initialized to either all 1's or an alternating pattern of 1's and 0's. In particular, runs 7 & 8 used a hex value 0xa55aa55a. The hex representation for 5 is 0101 and for A is 1010. So the resulting value was as follows: 10100101010110101010010101011010. This was selected to provide a mix of 1's and 0's within the lower 32-bit unsigned integer.

Run 1: Hashes = 10,000,000 starting value=10,000,000, size = 32
Run 2: Hashes = 10,000,000 starting value=-16,777,216, size=32. Note that the 10 million samples are at the high end of the possible addresses.

Run 3: Hashes = 10,000,000 starting value=1 size=64
Run 4: Hashes = 10,000,000 starting value=-16,777,216 size=64
Run 5: Hashes = 10,000,000 starting value=1 size=64 (all 1's). That is the w[0] value is set to all 1’s. That is, the first 32-bits are all 1’s. Also note that the SHA-256 provided its poorest results under this condition.
Run 6: Hashes = 10,000,000 starting value=-16,777,216, w[0] = 1’s
Run 7: Hashes = 10,000,000 starting value=1, size = 64, w[0] is set to a mix 0xa55aa55a
Run 8: Hashes = 10,000,000 starting value=-16,777,216, size = 64, w[0] is set to a mix 0xa55aa55a

Next the test fixtures were run using the SHA-256 algorithm. Of the 8 tests run, table 1 summarizes the number of samples considered a “process out of control.” (Later in the paper, figure 7 provides a graphic of this table.)

Run 1	Hi 10 Low 19	Run 5	Hi 29 Low 249
Run 2	Hi 6 Low 26	Run 6	Hi 49 Low 34
Run 3	Hi 80 Low 54	Run 7	Hi 109 Low 14
Run 4	Hi 62 Low 29	Run 8	Hi 57 Low 40

Table 1 SHA-256 sample summaries exceeding the SPC Threshold

Having identified a number of samples indicating "a process out of control,” the next step was to try and identify why. According to the National Institute of Standards and Technology (NIST) Federal Information Processing Standards (FIPS) Publication 180-2, “These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers.” No reason was provided as to why these values were selected. However, the implication was that prime numbers were considered necessary for the SHA-256. Close examination of the SHA-256 constants reveals that only four of the numbers are actually prime numbers. These numbers are listed below.

K_5 (3956c25b) is a prime number
K_7 (ab1c5ed5) is a prime number
K_{28} (c6e00bf3) is a prime number
K_{37} (766a0abb) is a prime number

To see if prime numbers were needed, a new array of prime numbers was selected and the tests run. Once again, there were numbers of samples outside of the statistical Threshold. To further examine why this might be, a short program was created to count the number of 1’s within the array. Assuming a uniform distribution, one would expect 1024 bits to be one. However, number of 1’s within the SHA-256 *k* array is 993 out of 2048. To further analyze the SHA-

256 default constants, another program was written to calculate the number of 1’s for each constant. For example, the prime number constant K_5 (3956C25B) has 16 1’s (0011-1001-0101-0110-1100-0010-0101-1011). The following table illustrates the SHA-256 results.

K_0 = 22
K_1 = 14
K_2 = 20
K_3 = 20
K_4 = 16
K_5 = 16
K_6 = 14
K_7 = 18
K_8 = 14
K_9 = 11
K_{10} = 14
K_{11} = 16
K_{12} = 19
K_{13} = 18
K_{14} = 17
K_{15} = 17
K_{16} = 16
K_{17} = 20
K_{18} = 16
K_{19} = 11
K_{20} = 18
K_{21} = 13
K_{22} = 16
K_{23} = 18
K_{24} = 14
K_{25} = 15
K_{26} = 12
K_{27} = 23
K_{28} = 16
K_{29} = 17
K_{30} = 13
K_{31} = 13
K_{32} = 15
K_{33} = 13
K_{34} = 18
K_{35} = 13
K_{36} = 14
K_{37} = 17
K_{38} = 13
K_{39} = 13
K_{40} = 17
K_{41} = 14
K_{42} = 14
K_{43} = 16
K_{44} = 14
K_{45} = 13
K_{46} = 15
K_{47} = 10
K_{48} = 12
K_{49} = 14
K_{50} = 15
K_{51} = 16
K_{52} = 14
K_{53} = 15
K_{54} = 18
K_{55} = 19
K_{56} = 17
K_{57} = 18
K_{58} = 11
K_{59} = 10
K_{60} = 13
K_{61} = 15
K_{62} = 23
K_{63} = 17

Consider the value K_{63} = BEF9A3F7. This is represented by the following: 1011-1110-1111-1001-1010-0011-1111-0111 for a total of 23 1’s (and 9 0’s). A new array was generated that had exactly 1024 1’s and 1024 0’s. Once again, the hash results contained many samples considered “a process out of control.” Considering that prime numbers will have the least significant bit set to a 1, the next thought was that by using prime numbers, there was consistency in the constants and therefore a reduction in randomness. Note, in the case of Cyclic Redundancy Checks (CRC) or polynomial checksums, the least significant bit is always inferred so representation is not necessary. Another point is that CRC uses a MOD 2 division across all bits.

In the hope of reducing the impact associated with the least significant bit always one, a bit rotating modification was applied to the SHA-256 algorithm. The following illustrates a code snippet that uses the least significant 5-bits (rotation is between 0 and 31 bits inclusive) to determine how many places to rotate the 32-bit unsigned integer. In retrospect, even this rotation provides some determinism. Nevertheless, it was hoped to see if any inference could be made regarding the use of all prime numbers with the least significant bit always set to 1.

```
T1 = h + Sigma1(e) + Ch(e,f,g) + k[t] + w[t];
T2 = Sigma0(a) + Maj(a,b,c);
h = ROTR(g, (0x0000001f & h));
g = ROTR(f, (0x0000001f & g));
f = ROTR(e, (0x0000001f & f));
e = d + T1;
d = ROTR(c, (0x0000001f & d));
c = ROTR(b, (0x0000001f & c));
b = ROTR(a, (0x0000001f & b));
a = T1 + T2;
```

Note that the current variable provided the 5-bit rotation value used in determining how many placed to rotate. It was hoped that this would provide a pseudo-random approach for rotating variables.

The next table, 2, summarizes the results and is referenced throughout the remainder of this paper. To delimit the various test runs, each starts with a shaded average.

	SHA-256	Using all Primes (Primes 1)	Random Rotation	New Primes (Primes 2)	New Primes and Random Rotation
1. Average	14999915.51	15000260.93	14999800.13	15000475.7	15000149.81
σ	3636.659235	3263.691944	3724.717058	3865.039014	3576.932707
3σ	10909.97771	9791.075831	11174.15118	11595.11704	10730.79812
Average - 3σ	14989005.54	14990469.86	14988625.97	14988880.59	14989419.01
Average + 3σ	15010825.49	15010052.01	15010974.28	15012070.82	15010880.61
Hi	10	85	70	62	79
Low	19	60	0	191	47
2. Average	14999690.71	14999901.24	14999948.04	14999822.51	14999841.35
σ	3525.590188	3547.734373	3501.028731	3453.650234	3462.272359
3σ	10576.77056	10643.20312	10503.08619	10360.9507	10386.81708
Average - 3σ	14989113.94	14989258.03	14989444.95	14989461.55	14989454.54
Average + 3σ	15010267.48	15010544.44	15010451.12	15010183.46	15010228.17
Hi	6	60	57	167	109
Low	26	36	5	29	0
3. Average	15000459.3	15000040.7	15000000.8	15000154.83	14999868.88
σ	3639.475825	3539.78344	3636.90566	3643.824875	3125.753231
3σ	10918.42748	10619.3503	10910.717	10931.47462	9377.259692
Average - 3σ	14989540.88	14989421.4	14989090.1	14989223.36	14990491.62
Average + 3σ	15011377.73	15010660.1	15010911.6	15011086.31	15009246.14
Hi	80	33	70	301	20
Low	54	105	44	71	206
4. Average	15000114.48	14999811.2	15000230.3	15000131.82	14999757.38
σ	3732.716614	3551.240409	3473.30192	3507.913847	3543.986283
3σ	11198.14984	10653.72123	10419.9058	10523.74154	10631.95885
Average - 3σ	14988916.33	14989157.48	14989810.3	14989608.08	14989125.42
Average + 3σ	15011312.63	15010464.92	15010650.2	15010655.56	15010389.34
Hi	62	28	54	133	191
Low	29	65	1	51	16
5. Average	15000097	15000681.5	15000007.2	15000005.7	15000331.5
σ	3508.76049	3360.19235	3303.36354	3344.63741	3786.495486
3σ	10526.2815	10080.5771	9910.09062	10033.9122	11359.48646
Average - 3σ	14989570.7	14990600.9	14990097.1	14989971.8	14988972.01
Average + 3σ	15010623.2	15010762.1	15009917.3	15010039.6	15011690.98
Hi	29	125	23	154	43
Low	249	67	248	2	21
6. Average	14999728.31	15000011.1	15000014.02	14999554.43	15000482.29
σ	3753.888625	3475.43587	3176.766305	3554.391145	3358.892557
3σ	11261.66588	10426.3076	9530.298915	10663.17343	10076.67767
Average - 3σ	14988466.65	14989584.8	14990483.72	14988891.25	14990405.61
Average + 3σ	15010989.98	15010437.4	15009544.32	15010217.6	15010558.97
Hi	49	2	88	100	22

	SHA-256	Using all Primes (Primes 1)	Random Rotation	New Primes (Primes 2)	New Primes and Random Rotation
Low	34	25	43	55	231
7. Average	14999972.26	14999635.2	14999928.05	15000259.55	15000237.94
σ	3458.229172	3332.01374	3296.225964	3690.038521	3379.704907
3σ	10374.68752	9996.04122	9888.677892	11070.11556	10139.11472
Average - 3σ	14989597.57	14989639.2	14990039.37	14989189.44	14990098.83
Average + 3σ	15010346.94	15009631.2	15009816.73	15011329.67	15010377.05
Hi	109	72	23	4	22
Low	14	86	10	46	0
8. Average	15000543.49	14999858.96	15000176.21	14999643.75	14999886.75
σ	3384.023608	3613.83896	3608.368567	3472.42854	3657.398568
3σ	10152.07082	10841.51688	10825.1057	10417.28562	10972.1957
Average - 3σ	14990391.41	14989017.44	14989351.11	14989226.47	14988914.55
Average + 3σ	15010695.56	15010700.47	15011001.32	15010061.04	15010858.94
Hi	57	38	39	61	5
Low	40	65	93	21	43

Table 2: Comparison of Results

Analysis

There appears to be a number of conditions that provide results outside of acceptable limits as measured using SPC Thresholds across 2-bit accumulated samples. While this analysis does not specifically identify specific weaknesses exist, it does suggest the current algorithm may have unexpected consistencies. Consider the following Primes 2 (new primes) with rotation Run 5 example where the last 5 samples and location are depicted. The largest sample at location [78,198] has a sum 1743 higher than the Statistical Process Control Threshold or 46% of one standard deviation. While on an absolute scale this may not seem like much; but when compared to all other samples, it is excessive.

15012845	[78,123]	15000331.5	Average
15012976	[78,46]	3786.495486	σ
15013018	[78,45]	11359.48646	3σ
15013323	[198,78]	14988972.01	Average - 3σ
15013434	[78,198]	15011690.98	Average + 3σ
		43	Hi
		21	Low

In contrast, looking at the SHA-256 run 5 results, the ten lowest samples, with the statistical information to the right, follows:

Sample	Location
14984045	[132,231]
14984284	[132,13]
14984562	[132,181]
14984670	[132,23]
14984718	[132,99]
14984801	[132,20]
14984998	[132,114]
14985153	[132,87]
14985186	[132,40]
14985231	[132,73]

Average	15000097
σ	3508.76049
3σ	10526.2815
Average - 3σ	14989570.7
Average + 3σ	15010623.2
Hi	29
Low	249

For the low value samples observed, bit 132 produced an inordinate number of 0's during the run. The lowest sample at the 2-bit location [132,73] was 16,052 below the average. This was 4.57 standard deviations below the average. In comparison, the ten highest samples taken from the Primes 2 run 3 are as follows:

15014650	[91,213]
15014697	[91,9]
15015047	[213,137]
15015132	[137,164]
15015141	[9,137]
15015419	[137,34]
15015759	[91,137]
15016156	[137,213]
15016203	[137,9]
15016512	[137,91]

15000154.83	Average
3643.824875	σ
10931.47462	3σ
14989223.36	Average - 3σ
15011086.31	Average + 3σ
301	Hi
71	Low

The largest sample at 2-bit location [137, 91], was 16,357 above the average. This is 4.49 standard deviations above the average. Note that the size of three standard deviations is critical in determining the Threshold. Given that changes to the constants produced different samples exceeding the SPC Threshold, indicates that there is likely something in the basic algorithm that could be improved. In the remaining figures, the results of eight test runs summarized in table 2 are graphically depicted. It should be noted that had additional runs been done, more information might have been inferred from the test results.

The SHA-256 was run using eight different initializations. As the average distance from the average increased, so too did the 3σ values as shown in Figure 1.

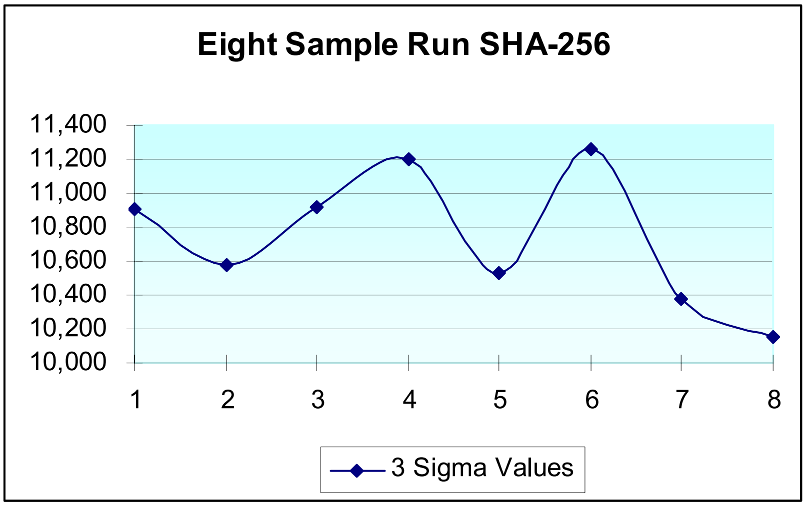

Figure 1: 3σ Values for SHA-256 Runs

Additionally, the sample averages also changed with each run. Figure 2 illustrates how the SHA-256 averages changed with each run. The expected value for the samples is 15 million, shown as the heavy line in the figure.

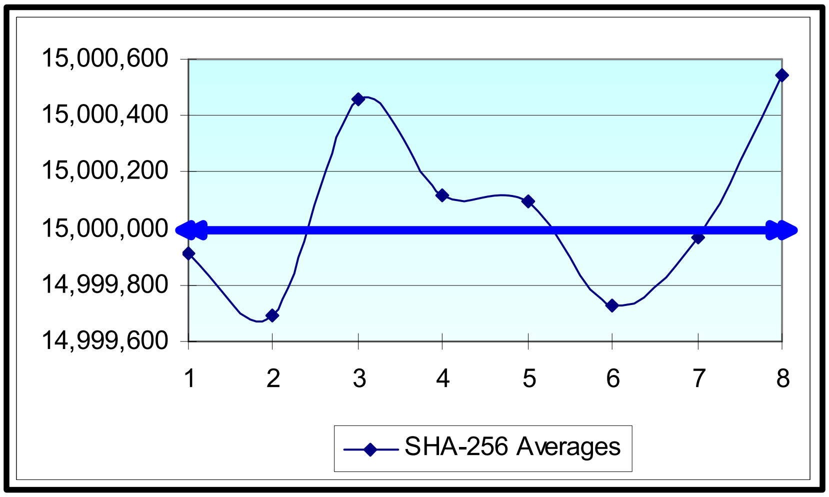

Figure 2: SHA-256 Sample Averages

The next figure, 3, illustrates the 3σ values for the SHA-256 as compared to the values obtained from using all primes (cases 1 & 2). From this figure, the three environments appear to be within the same value range.

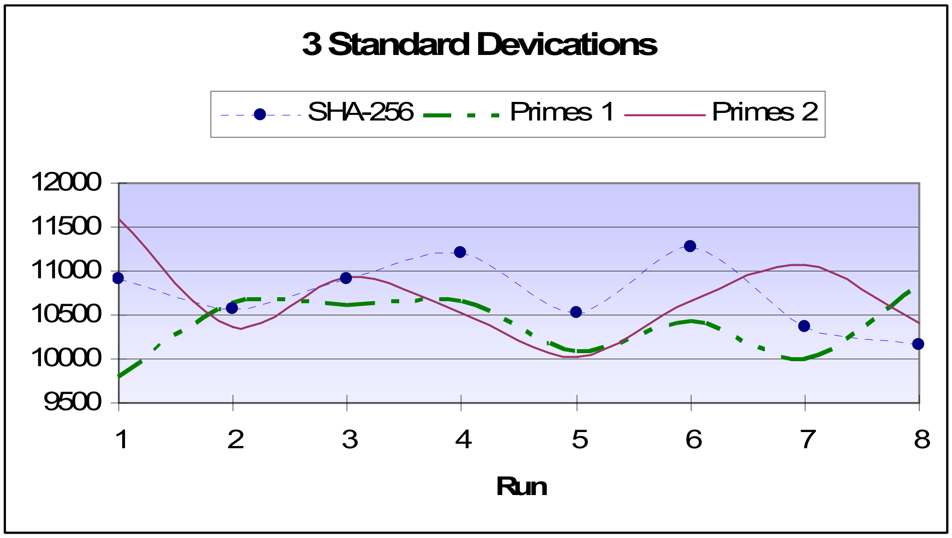

Figure 3: 3σ SHA-256 and Primes comparisons

The next figure, 4, depicts the sample averages of the SHA-256 compared with those of Primes 1 and 2. Again, the results all appear relatively close.

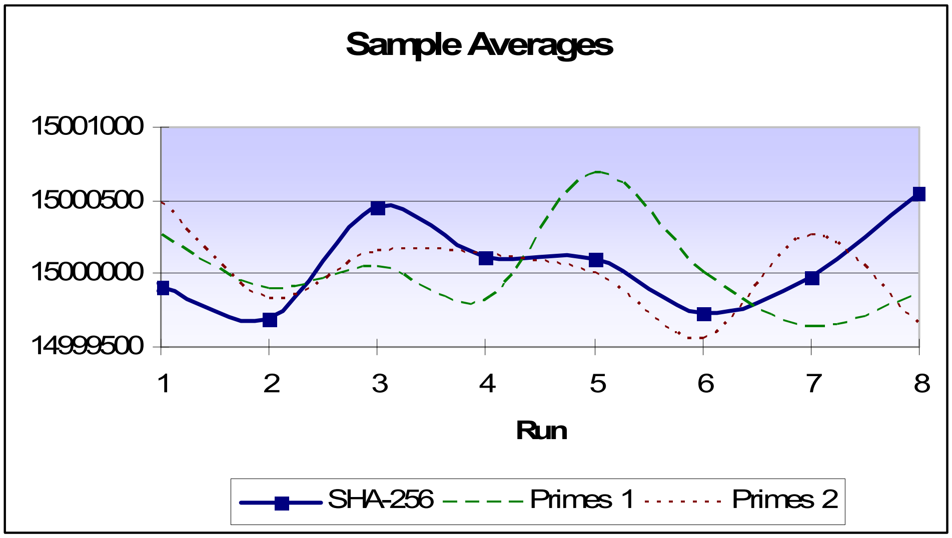

Figure 4: SHA-256 and Primes Sample Averages

To attempt compensating for the fixed least significant bit, an additional rotation function previously discussed was added to the hash algorithm. The 3σ values are shown in figure 5. It is interesting that there were wider variations in the rotated prime values.

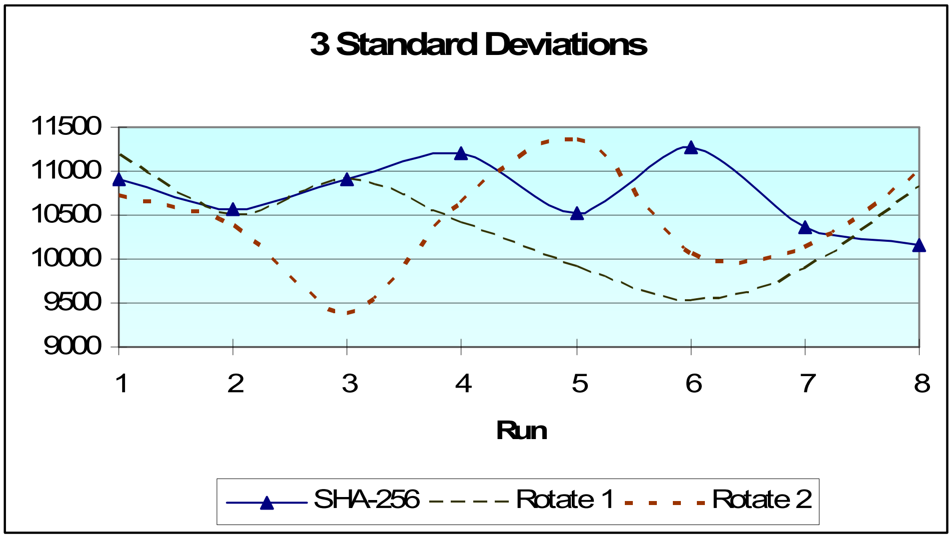

Figure 5: 3σ SHA-256 and Rotated Primes comparisons

The next figure, 6, illustrates the sample averages of the SHA-256 compared to the rotated primes values. Again, the values are relatively close.

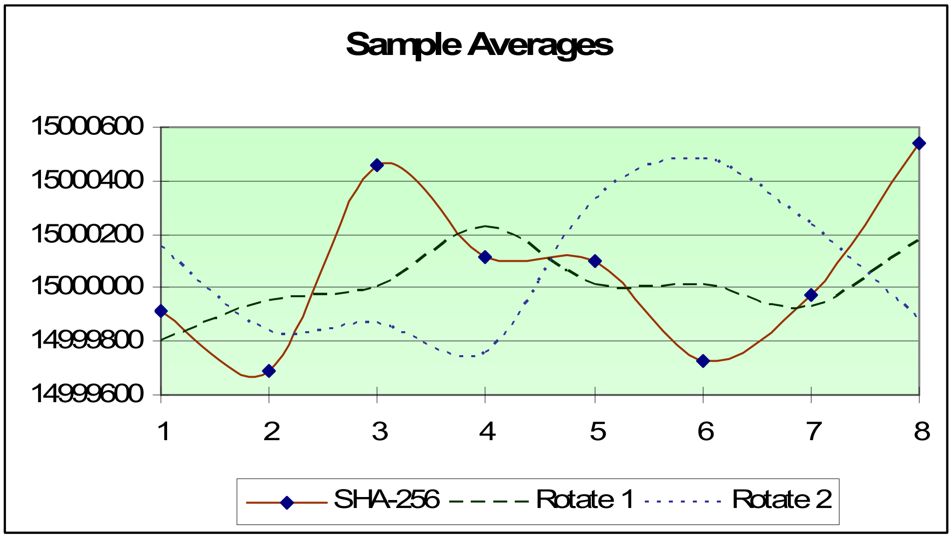

Figure 6: SHA-256 and Rotated Primes Sample Averages

The next figure, 7, illustrates the SHA-256 samples that exceeded the SPC Threshold for each test run. The total number of samples exceeding the SPC Threshold is the sum of those too hi and too low. The next three figures illustrate results obtained when changes were made to the *k* array of constants.

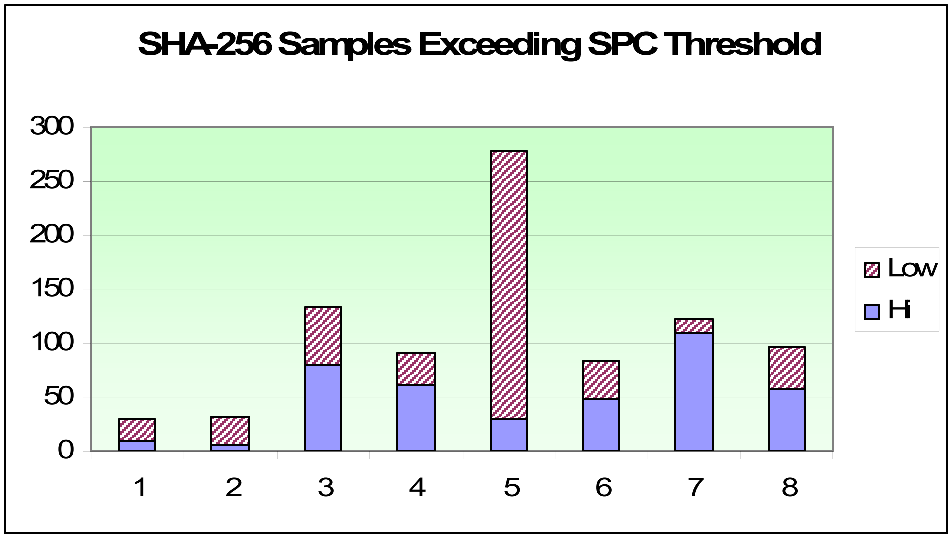

Figure 7: SHA-256 Exceeded SPC Threshold Summary

The next figure, 8, depicts the number of samples exceeding the SPC Threshold when using the first set of prime numbers.

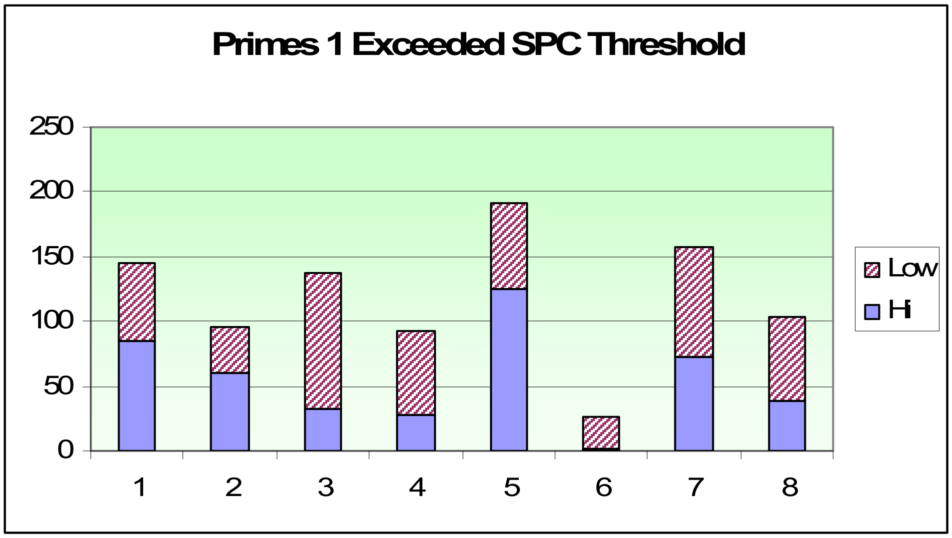

Figure 8: Primes 1 Exceeded SPC Threshold Summary

When selecting the second set of prime numbers, many were evenly selected in the range 0.5 – 4.0 billion. This selection process may have contributed to consistency in that the most significant hex value had a fixed distance. Nevertheless, figure 9 depicts the eight test run results using the second set of prime numbers.

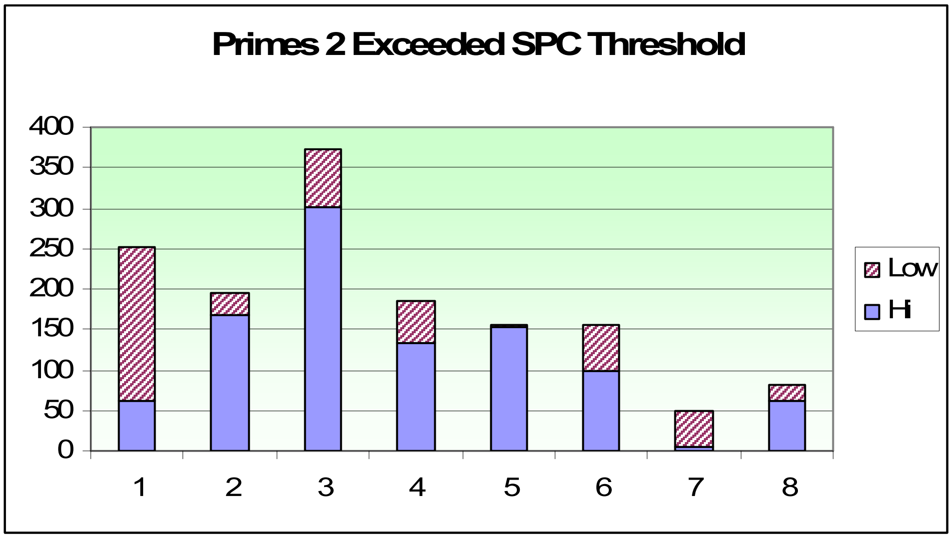

Figure 9: Primes 2 Exceeded SPC Threshold Summary

In the next figure, 10, the results from the three previous charts are compared side by side using the total number of samples exceeding the SPC Threshold. It is interesting to note, the number of samples exceeding the SPC Threshold are different for each run. Changing the array of constants did not appear to be a solution for getting samples within three σ of the mean.

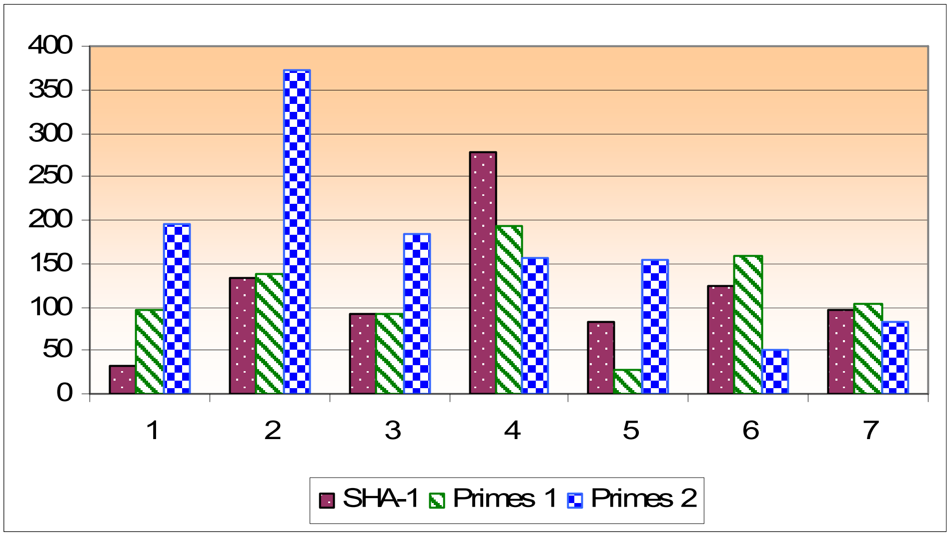

Figure 10: Comparison of Samples Exceeding SPC Threshold

As previously indicated, rotation was used to remove the consistent constant (least significant bit always set to a 1). The results from rotating the prime 1 constants array is shown in figure 11.

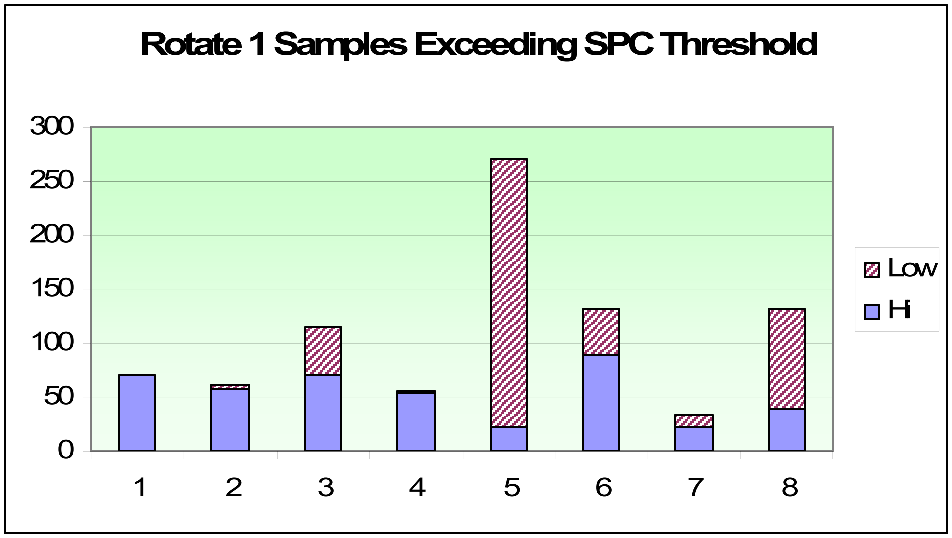

Figure 11: Rotated Primes 1 SPC Threshold Summary

Similarly, the rotation algorithm applied to the second array of primes is depicted in figure 12.

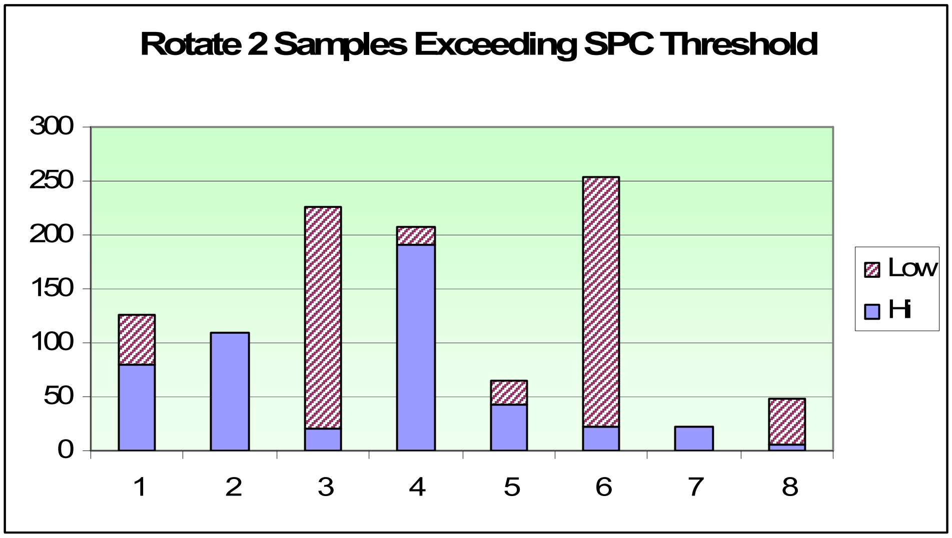

Figure 12: Rotated Primes 2 SPC Threshold Summary

Figure 13 provides a comparison of the SHA-256 to the rotated primes SPC Thresholds. That is, figure 7 is compared with figures 11 & 12. From this last chart and the limited number of runs, there was no run that had all samples within the SPC Threshold. It is unlikely there are 64 constants that will consistently produce results with no out of process samples.

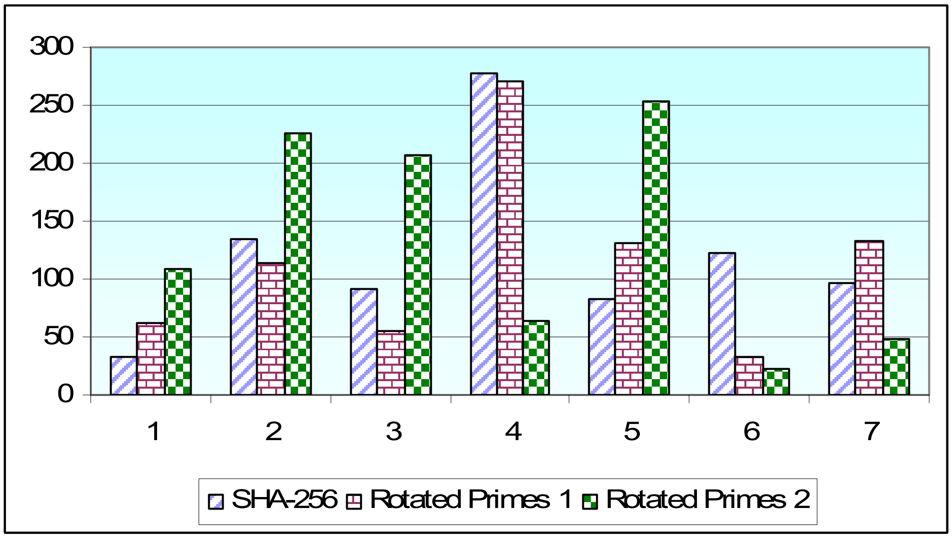

Figure 13: SPC Threshold Summary

Future work

Limited tests were run on one machine to generate the data used in preparing this paper. A more robust 3-bit sample scheme and additionally initializations would provide additional information regarding hash algorithm strengths. Much of the work presented in this paper focused on the constants, using primes, balancing the bit counts of the resulting array of constants, and applying random bit rotation. From the results analyzed, for every run there was a number of samples depicting processes out of control. Future work could focus on alterations to the Sigma, Ch, and Maj functions used within the SHA-256. Additionally, the other Secure Hash Algorithms should be tested. Perhaps testing the inclusion of MOD 2 division, such as is done with CRC algorithms, enhancements to the overall strength of secure hash algorithms could be explored. Future testing could answer this question. There is likely some algorithm modification that will eliminate sample values exceeding the SPC Threshold. This would in turn provide better confidence in the strength of the SHA algorithms.

Summary

This paper presented a testing approach using the Statistical Process Control Threshold to identify 2-bit values indicative of a process out of control. In each of the eight test runs, multiple samples were found to exceed the SPC Threshold. Changing the constants array was explored and found not to provide a consistent approach for eliminating excessive samples.