Date: Thu, 15 Apr 1999 15:15:00 -0500
From: Pete Mokros <pmokros@macalester.edu>
Subject: First round comments
To: aesfirstround@nist.gov
X-Mailer: Mozilla 4.5 [en] (WinNT; U)
X-Accept-Language: en

To whom it may concern:

Attached is a paper I have done indicating that Twofish should be considered the only contender for the AES title. Should further correspondence prove necessary, you can reach me at pmokros@macalester.edu until mid-May 1999. After then, I should be reachable at pmokros@yahoo.com

Sincerely,
Pete Mokros.

Comparative Analysis of the Advanced Encryption Standard Candidate Algorithms

Peter M B Mokros
Macalester College
15 April 1999

Introduction

On November 23, 1976 the government adopted a data encryption algorithm known as the Data Encryption Standard (DES). This symmetric block algorithm uses 56-bit keys to encrypt data, but brute force attacks have become more of a threat to the security of DES as computing power has continued to grow substantially while the monetary costs have plummeted. In the summer of 1998, the Electronic Frontier Foundation announced the existence of a $220,000 hardware unit that could crack DES keys in an average of 4.5 days [1]. DES is widely used by the government (including military), banks, the telecommunications industry, and most large corporations. It is clear that 56-bit keys are no longer sufficient or secure—the threat to DES-encrypted messages will continue to grow. On September 12, 1997, the National Institute for Standards and Technology (NIST) announced that it would begin soliciting candidates for an algorithm to replace DES: the Advanced Encryption Algorithm (AES). This paper examines one aspect of cryptographic strength, bit diffusion, and applies it to a group of AES candidate algorithms.

Cryptographic Terminology

Cryptography is the study of making and breaking codes. A cryptographic algorithm converts *plaintext* into *ciphertext* for secure transmission and storage. This conversion is known as *encryption*. A *key* is some string of characters used to encrypt the plaintext—it can be considered a sort of password. Using a *symmetric key* algorithm, the same key is used for both encryption and *de*cryption. One pitfall of using a symmetric key algorithm is the problem of communicating the key to the intended message recipient without anyone else finding out what the key is. *Public-key* algorithms use two keys that are mathematically correlated: a private key and a public key. The public key can be made available for people to use to encrypt messages meant for you and the private key is used to decrypt public key-encrypted messages. Disclosure of the public key does not diminish the security of the private key or any message encrypted by the public key.

Keys are measured in bits. At the lowest level, a message is a string of 1s and 0s and it is at this level where encryption takes place. Bits of the key, also known as *key material*, are used to manipulate bits of the message using operations such as exclusive-or (XOR). In general, more key bits mean more security in terms of message encryption and decryption.[1] For a brute force attack on a key with 56 bits to succeed, 2^{56} keys must be checked; this is a huge number, but given a reasonable amount of computing power it can be achieved in a realistic amount of time. Keys for symmetric algorithms are usually shorter than public-key keys because of the overhead involved in creating the public and private keys according to a mathematical correlation. Symmetric keys are on the order of 56-256 bits whereas public-key key sizes range from 512-2048 bits.

Encryption algorithms are either *stream* or *block* ciphers. A stream cipher is an encryption algorithm that encrypts plaintext one bit at a time whereas a block cipher encrypts bunches of bits, called blocks, all at once. Blocks can theoretically be any size, but they are frequently 64 bits. The reason for splitting a message up into blocks comes from the length of the key and the message. If our key is 128 bits long and our message is 256 bits long, then we would need at least two blocks because the maximum number of bits we can encrypt at once is dependent on the key size which is 128 bits. The common practice of having a block size smaller than the key size forces the encryption of consecutive parts of the message with different key material. This technique can thwart certain basic attacks.[2]

AES Information

The minimum acceptability requirements for AES candidates mandated in the Request for Candidate Algorithm Nominations [5] are stated as:

- The algorithm must implement symmetric (secret) key cryptography,
- The algorithm must be a block cipher, and

[1] A large key size won't make your messages secure if you leave your key on a disk in your disk drive or if you leave a printout of the plaintext message on your desk. Attacks can be made against any part of the system. Refer to [2] and [3] for excellent treatments of the "NP-hard" nature of cryptography.

[2] See pages 191, 196, 199, 203, and 205 in [4].

- The candidate algorithm shall be capable of supporting key-block combinations with sizes of 128-128, 192-128, and 256-128 bits.

The first two points are clear, but the third allows us to draw more information. The block size for all AES candidates must be 128 bits and the key size can be 128, 192, or 256 bits. As mentioned earlier, more key material generally translates into more security from a brute force attack. The table below shows the time needed to mount a brute force attack against various key sizes given a certain budget.[3]

Average Time Estimates for a Hardware Brute-Force Attack in 1995

Cost	**40 bit key**	**56 bit key**	**64 bit key**	**80 bit key**	**112 bit key**	**128 bit key**
$100 K	2 seconds	35 hours	1 year	70,000 years	10^{14} years	10^{19} years
$1 M	.2 seconds	3.5 hours	37 days	7000 years	10^{13} years	10^{18} years
$10 M	.02 seconds	21 minutes	4 days	700 years	10^{12} years	10^{17} years
$100 M	2 ms	13 seconds	9 hours	70 years	10^{11} years	10^{16} years

The exponential nature of base 2 works out nicely for us here. Increasing from a 56-bit to a 128-bit key may sound like we are only increasing the difficulty of a brute force attack by a factor of approximately 2.3, but that factor is actually an enormous 2^{72}. It should be clear that 128-bit keys (and above) are relatively immune from a straightforward brute force attack. The problems that NIST will be examining above all others in the AES candidates are alternate attacks. Designating a key size is trivial, but protecting against such things as linear cryptanalysis, differential cryptanalysis, chosen-plaintext attacks, related-key cryptanalysis, and heuristic arguments is the challenge.[4]

Description of Approach

The alternate approach to attacking an algorithm that we will be concerned with is in the area of heuristics. If we have a message's ciphertext and can encrypt messages on our own, what can we learn about the ciphertext by going backwards toward plaintext? This is precisely the thing an algorithm designer wants to protect against, and one way of doing so is to completely scramble the message at each stage, also called a *round*,[5] so that going backwards is difficult. If there is no correlation between the plaintext and the ciphertext, then you cannot intercept the ciphertext and decrypt it. Clearly, there is a correlation between the plaintext and the ciphertext: plaintext + algorithm + key = ciphertext. Assuming we know the details of the ciphertext and the algorithm, we will have to know the key in order to get what we want: the plaintext. The most we can ask for is that the security rests in the key, but this assumes that the algorithm by itself is perfectly secure.[6]

Given the intermediate ciphertext constructed at round 6, is there anything that I can conclude about the intermediate ciphertext constructed at round 5? If so, then if I can replicate the process for all the other rounds I can make some conclusions about the plaintext given a ciphertext. This is the challenge that we will concern ourselves with. Ideally, the intermediate ciphertext output at each round will be a random permutation of the previous intermediate ciphertext. If this best case scenario is true for some algorithm, the ciphertext will be of no use to an attacker in search of the plaintext; the attacker should get ready to mount a brute force attack.

One way to measure the feasibility of such a heuristic correlation attack is to examine the intermediate ciphertext at each round and compare it with "random"[7] data. If the intermediate ciphertext is random as far as we can tell, then there isn't anything we can necessarily conclude about any other intermediate

[3] The full table can be found on page 153 of [4].
[4] These topics are discussed in theory and practice with examples in [4].
[5] Most cryptographic algorithms have 10-20 rounds. DES has 16 rounds.
[6] This does not happen.
[7] It is an open question whether anything is truly random. In lieu of referring to "pseudo-randomness", I will assume the theoretical existence of true randomness.

ciphertext or the plaintext. A good algorithm will provide intermediate ciphertexts that are indistinguishable from random data. This requires that we begin to understand what constitutes randomness.

Randomness: What is it?
What constitutes random behavior? Is the sequence “00000” a random selection of five integers from 0-9? It certainly does not appear to be random, but in choosing five integers from 0-9 the sequences “00000”, “99999”, and “47823” have the same probability of appearing. What random means to most people is a uniform distribution. If, instead, we are looking for ten integers from 0-9, the probability of each integer appearing is $\frac{1}{10}$. Each pair of digits should appear $\frac{1}{100}$ of the time. In fact, with a sequence of a million integers it is unlikely that there will be exactly 100,000 zeros, 100,000 ones, and so forth. A sequence of million digit sequences will tend towards that uniform distribution though.

When one considers the sequence “00000” it is difficult to believe there is any random behavior, but is the sequence “14965” any more random? The person on the street would point to the second sequence as being more random, but it is nothing more than the least significant digits of the first five squares in order. If I was an attacker trying to figure out the sixth “random” digit in that sequence it would likely be trivial; let’s hope that no bank records depend on that one. The point here is that arithmetical means of attaining randomness are ill-advised. If the sequence “00000” is obtained from taking the least significant digit from the time in milliseconds between power spikes of a wall socket or between wind gusts, then this is much less predictable. If we extend the sequence obtained by choosing the least significant digits of squares from five to thirty-one digits, we get: “12 $\underline{4965694101}$ $\underline{4965694101}$ $\underline{4965694101}$.”

Randomness: Can we measure it?
There are many ways to measure randomness. Most are algorithmic, but some are arithmetic. Examples of algorithmic tests include the spectral test, the birthday spacing test, the permutation test, the coupon collector’s test, and the poker test.[8] One arithmetic measurement is the chi-square test and it is quite simple and straight-forward. Let’s examine these two approaches in the reverse order. The chi-square test is expressed below:

$$\chi^2 = \sum \frac{(observed - \exp ected)^2}{\exp ected}$$

The chi-square test makes perfect sense—if it didn’t already exist, anyone with intermediate algebra skills could invent it without difficulty. If we know how many occurrences of distinct digits to expect, we calculate the difference and square it in order to exaggerate any extremely unexpected results. This number is then divided by the expected number of occurrences to generate a relative measure. We then sum this across all possible digits that could appear to arrive at one quantitative value to measure a sequence’s uniform distribution. For example, if we had a sequence with 1,000,000 digits we would expect that the digit 0 would appear 100,000 times, the digit 1 would appear 100,000 times, and so on. If these proportions actually showed up in a sequence, the χ^2 value would be 0 since the number of observed occurrences and the number of expected occurrences were equal for all possible digits.

Notice that if the sequence was “00…00111…11222…22…999” we wouldn’t detect this as random behavior, but the χ^2 test only claims to measure uniform distribution over the interval we are measuring which, in this case, is one million digits. Over that interval, uniform distribution was achieved. If we only examined the interval from digit 1 to digit 100,000, our χ^2 value would be quite large because the differences between the observed and expected values for the digits 1, 2, 3, 4, 5, 6, 7, 8, and 9 are 100,000 which will be squared and divided by 100,000. The sum of these values is quite large. In that interval the sequence is only “00…000” which does not have uniform distribution of all digits. It is clear that smaller χ^2 value is desired when in search of uniform distribution.

[8] These tests, and others, are thoroughly discussed on pages 1-193 of [6].

The Poker Test

An example of an algorithmic test for randomness is the poker test.[9] This test is particularly useful because it is simple to learn and to use. A sequence made up of items from a set with *i* distinct elements can be thought of as a hand (or several hands) of playing cards. Anyone who has played poker will tell you that you are much more likely to get a hand with two of a kind than a flush. The same idea of probability is utilized in the poker test. If we have a five-digit sequence whose elements are chosen from the set {a,b,c,d,e}, then there are seven possible types of order-less poker hands:

All different ("ace-high"):	abcde
Two of a kind:	aabcd
Two pair:	aabbc
Three of a kind:	aaabc
Full house:	aaabb
Four of a kind:	aaaab
Five of a kind ("flush"):	aaaaa

So, if we have an *n*-digit long sequence of elements we can split it up into blocks of length 5 in order to classify them into the groups above. For random data, we will see a certain distribution based upon the likelihood that a certain "formation" would appear. It seems logical that there will be more ace-high and two-of-a-kind formations than four- and five-of-a-kind. We can work these distributions out mathematically in order to determine the expected results. If we use a χ^2 test to compare the expected results with the empirical results obtained from a poker classification, we can get a good measurement for how random a sequence of digits is. This is generally how I approached the testing of AES candidates.

Applying the Poker Test

In testing AES candidates, we are interested in how the ciphertext changes as it passes through various stages of the algorithm. The data is stored as 16 hexadecimal values which we convert into a binary representation to obtain 64 bits of information for each round. Instead of the classes we saw above, we want something simpler partially because we want to be able to utilize all 64 bits. Since 64 is not evenly divisible by five,[10] we can use the next closest number that does divide 64: four. With four distinct elements, the following classes of poker hands will be represented:

All different:	abcd
Two of a kind:	aabb, aabc
Three of a kind:	aaab
Four of a kind:	aaaa

Using bit representation, it only takes two bits to represent four distinct elements. So, we can replace 00 by *a*, 01 by *b*, 10 by *c*, and 11 by *d*. We then get a sequence of 32 letters selected from the set {*a*,*b*,*c*,*d*}. We can then split the sequence of 32 letters into eight sequences of four letters representing poker hands. Classifying these hands into the categories above, we will see a distribution among the groups. As before, we are more likely to have more two-of-a-kinds than four-of-a-kinds and so forth.

One thing that this approach will not take into account is an offset. For example, the sequence "aaaa bbbb cccc …" will be found to be essentially non-random when examined using the poker test algorithm we have seen so far. If we choose "ddaa aabb bbcc ccdd …" where the sequence is "padded" by an offset of several values (in this case a pair of d's), we will find a lot of two-of-a-kinds which is expected and when compared to the theoretical expected values it may not be too far off. To account for this possibility, we only need to make a slight modification to the poker test algorithm we already have. Not only should we divide the 32 characters into groups like "$C_1C_2C_3C_4$ $C_5C_6C_7C_8$ …", but we should also offset our counters so that we are checking groups such as:

C_1 $C_2C_3C_4C_5$ $C_6C_7C_8C_9$ … $C_{30}C_{31}C_{32}$
C_1C_2 $C_3C_4C_5C_6$ $C_7C_8C_9C_{10}$ … $C_{31}C_{32}$

[9] The poker test is discussed on pages 63-64 of [6].
[10] The number of distinct elements in the set we saw above.

For the sake of simplicity, we will not count groups whose length is less than four. This solves the problem of offsets in (non-) random behavior and it also gives us approximately four times as many groups to count in order to compare to theoretical expected values.

My implementation is a step-by-step translation of this poker description into C++ code. I have an array (intermedText[]) that stores the hexadecimal output of the encryption algorithms which is then converted into another array (numbers[]) which stores the binary representation of intermedText[]. The array numbers[] is reduced from binary 0's and 1's into representations of cards using the mapping described above. The resulting 32 characters are the basis for applying the poker test. The function pokerOrdinality(), counts the number of times each distinct card occurs in a given group and classifies the group as either one-, two-, three-, or four-of-a-kind. The implementation accounts for four possible offsets (zero through three) and examines 15 groups per offset.[11] The result is 60 card classifications that are split among the four possibilities.

The theoretical distribution of cards fall into four possible categories which we will calculate mathematically. First, there are 256 possible unordered arrangements ($4 \cdot 4 \cdot 4 \cdot 4 = 4^4 = 256$). The total number of **one**-of-a-kind arrangements is calculated by:

> (4 choices for the 1st card) · (3 choices for the 2nd card) · (2 choices for the 3rd card) · (1 choice for the 4th card) = **24**

As a percentage, this is 9.375% of all possible arrangements. The total number of **three**-of-a-kind arrangements is calculated by:

> (4 choices for the 1st card) · (1 choice for the 2nd card) · (1 choice for the 3rd card) · (3 choices for the 4th card) = 12
> (12 three-of-a-kinds) · (4 different cards that could be the one that occurs three times) = **16**

As a percentage, this is 18.75% of all possible arrangements. The total number of **four**-of-a-kind arrangements (a "flush") is calculated by:

> (4 choices for the 1st card) · (1 choice for the 2nd card) · (1 choice for the 3rd card) · (1 choice for the 4th card) = **4**

So, there are four possible four-of-a-kinds with four cards. As a percentage, this is 1.5625% of all possible arrangements. Now that we have three of the possible four categories, we can calculate the total number of **two**-of-a-kinds by accounting for the number of arrangements we have not yet described. We can calculate it as follows:

> (256 total) – (24 one-of-a-kind) – (16 three-of-a-kind) – (4 four-of-a-kind) = **180**

As a percentage, this is 70.3125%. The following table shows how many of each classification to expect for the 60 total poker hands that we are counting.

Classification	**Theoretical percentage**	**Expected occurrences from 60 groups**
One-of-a-kind	9.375%	6
Two-of-a-kind	70.3125%	42
Three-of-a-kind	18.75%	11
Four-of-a-kind	1.5625%	1

Once we have classified the observed number of occurrences for each possible type of poker hand, we use the χ^2 test to see how far the data has strayed from the theoretical distribution. This provides us with some information to use in analyzing the performance that we are concerned with. Clearly, we would like to minimize the χ^2 values at each step, but additionally it would be valuable to know how quickly a particular algorithm "recovers" from an instance of non-random behavior. If the χ^2 value remains low for the first several measurements and then jumps to a large value, will it then drop back down or will it maintain the

[11] It doesn't do 16 because it is only possible to get 16 with an offset of zero. For the sake of simplicity, it counts 15 for all four offsets rather than 15 for three offsets and 16 for another.

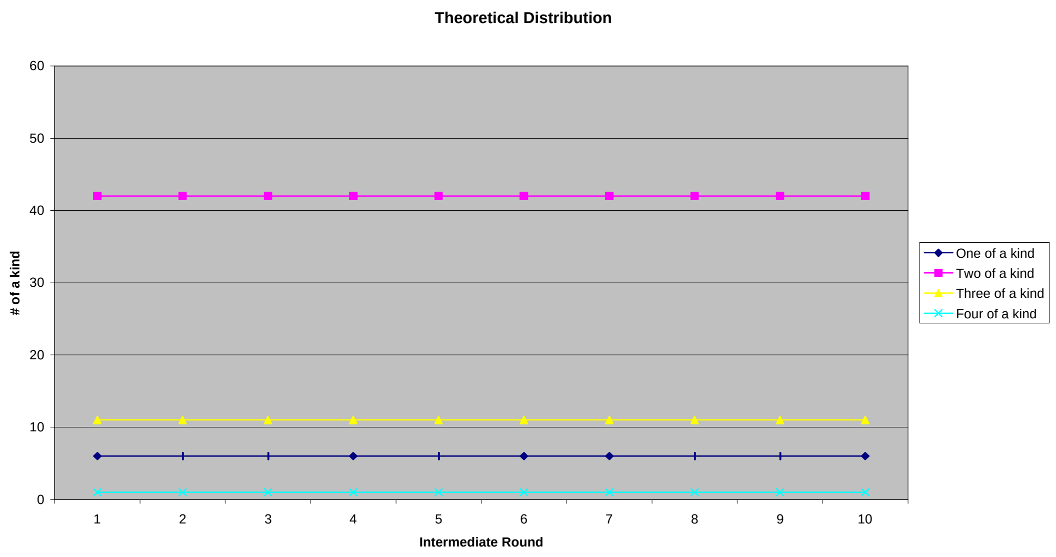
large values for several measurements? A good algorithm should tend toward the theoretical values at each stage. Thus, in the unlikely event that non-randomness occurs, there should be a very small “window” in which to analyze the behavior so that a reverse-engineering process proves infeasible or at least extremely difficult.

Results

Graphs of both the χ^2 and observed classification measurements are attached in Appendix A for all of the algorithms that were tested. The algorithms that were tested are:

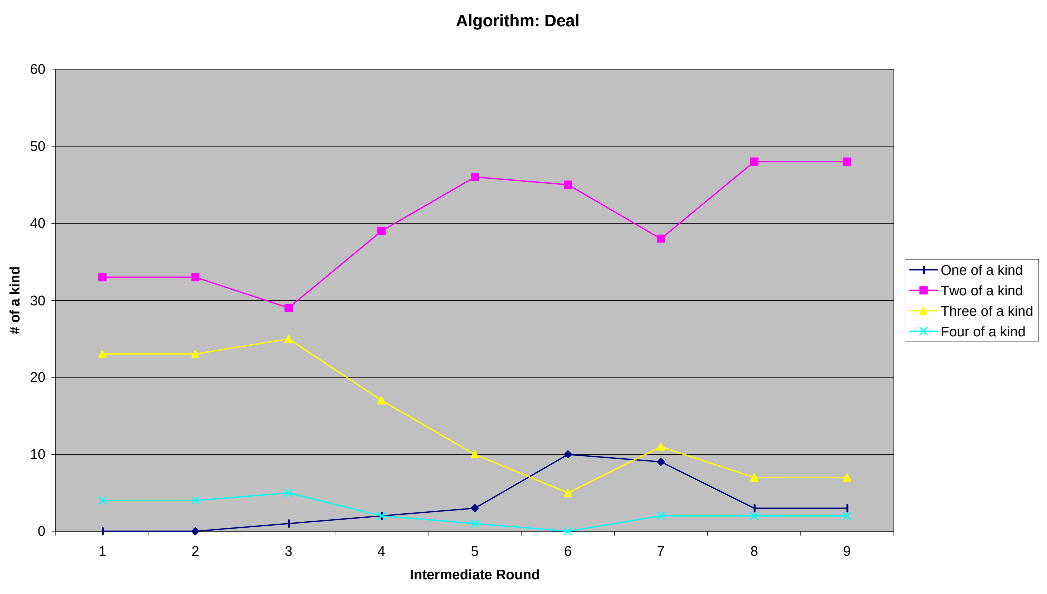
Algorithm name	**Principle submitters**	**Number of rounds**
Cast	Entrust Technologies, Inc.	14
Deal	Richard Outerbridge and Lars Knudsen	9
Dfc	Centre Natinal pour la Recherche Scientifique	9
Frog	TecApro Internacional S.A.	9
Loki97	Lawrie Brown, Josef Pieprzyk, and Jennifer Seberry	18
Magenta	Deutsche Telekom AG	7
Rijndael	Joan Daemen and Vincent Rijmen	11
Safer+	Cylink Corporation	9
Twofish	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson	20

A chart of the theoretical distribution of the various expected poker hands is shown below.

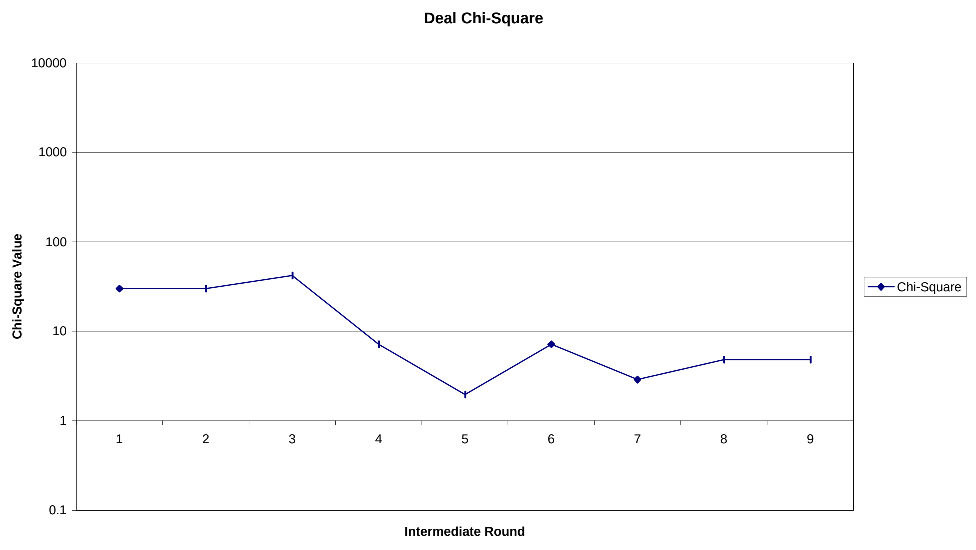


A good algorithm should have values as close to this as possible while figuring in random and statistical noise. Additionally, an algorithm whose poker hand classifications match this would give an attacker some information on which to analyze the data and potentially work backwards toward the original message. A certain amount of variation is, therefore, desired. A chart of the theoretical χ^2 values would not be interesting because it is a collection of zeros. A good algorithm should sustain a low χ^2 value for as many rounds as possible with preference going to the later rounds.

According to the poker test and the criteria set forth, the Deal algorithm performs poorly. The chart of the observed distribution of poker hands for a test run of the Deal algorithm is shown below.

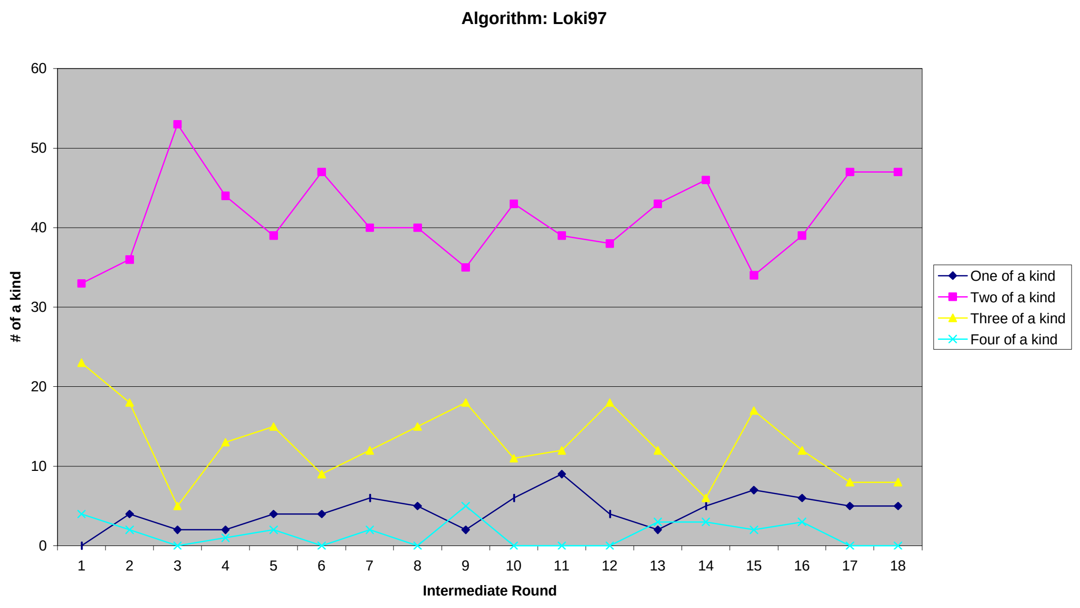


Notice that the distribution for the first half of the rounds is nowhere near the theoretical distribution. Furthermore, the initial distribution is perpetuated for three rounds (one-third of the entire algorithm) and does not quite reach a "reasonable" distribution until the fifth round (over half-way through the algorithm). This seems to imply that Deal is not quite as powerful as we would like with this particular input and with our criteria. Most of the candidate algorithms rely upon nine rounds to accomplish the plaintext to ciphertext translation, so Deal is not noticeably deficient in this regard, although more rounds would likely decrease the importance of the first few rounds being weak. The chart of χ^2 values for Deal is shown below.

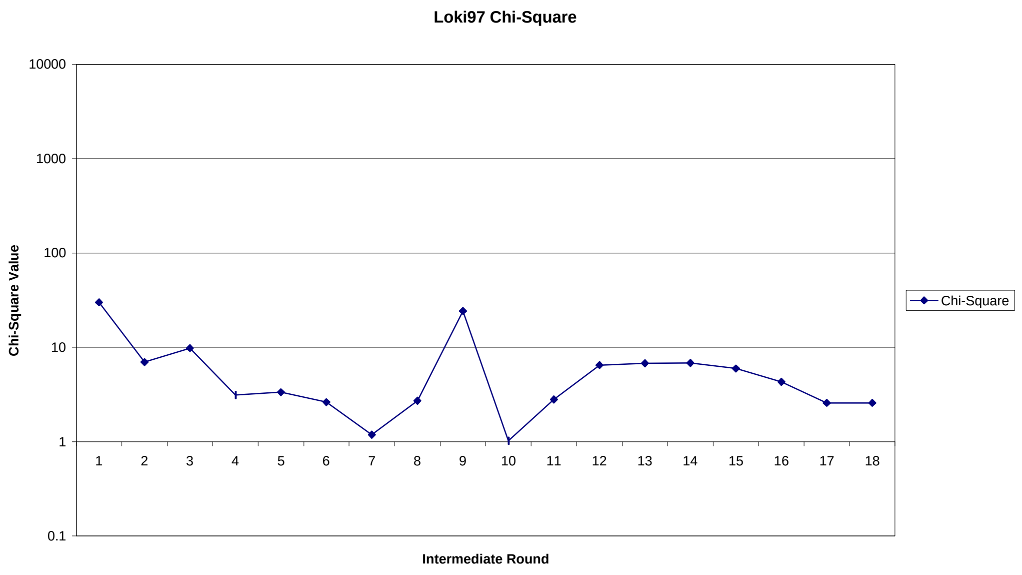


This chart illustrates the point that Deal performs poorly in the first several rounds, but then maintains a reasonable distribution of poker hand values.

In contrast to Deal, Loki97 performs quite well. The chart of poker hand distributions for Loki97 is shown below. Notice that it has many more rounds, it maintains a reasonable distribution for all intermediate rounds, and it "recovers" from the initially poor distribution quite quickly.



To further illustrate the difference between Loki97 and Deal, the chart of χ^2 values for Loki97 is shown below.



Other than the spike at round nine, this chart shows results that are desirable for our criteria. It should be clear to the reader that Loki97 fulfills our criteria to a much larger degree than Deal and would likely be a wiser choice as a dependable encryption algorithm.

The remaining charts are attached in Appendix A, but the algorithms fall into three categories. The excellent algorithms maintain a reasonable distribution of poker hands throughout the test, have a high number of rounds, and recover from apparent non-randomness in less rounds than the other algorithms. Average algorithms maintain an acceptable distribution of poker hands, but they may stray from the theoretical distribution too frequently or excessively. Average algorithms also have a smaller total number of rounds. Poor algorithms do not maintain an acceptable distribution of poker hands, do not recover from apparent non-randomness as quickly as the other algorithms, and may have too few rounds to be effective. The categories split up as follows:

Excellent	**Average**	**Poor**
Twofish	Cast	Deal
Loki97	Dfc	Magenta
	Frog	
	Rijndael	
	Safer+	

The reader is invited to investigate the attached charts in order to further explore the behavior of each individual algorithm.

Recent Work[12]
As expected, the academic community has been examining the 15 official AES candidates since the candidates have been made public. Initial results are as follows:[13]

- Frog's permutation functions used for each round have inherent weaknesses [7],
- Frog is susceptible to linear and differential cryptanalysis attacks [8],
- Deal is weak in some key areas and a certificational attack on the 192-bit version has been discovered [9],
- A variety of attacks (and a solution that prevents these attacks) have been proposed against Safer+ in [10],
- Magenta has been broken by the creators of Twofish [11], and
- Two authors claim to have an attack against Loki97 in [12].

Additionally, some of NIST's judging criteria will include speed and efficiency since one of the intended applications of AES is for use in smart cards. The fastest algorithms for encryption and decryption are in the following order from fastest to slowest:

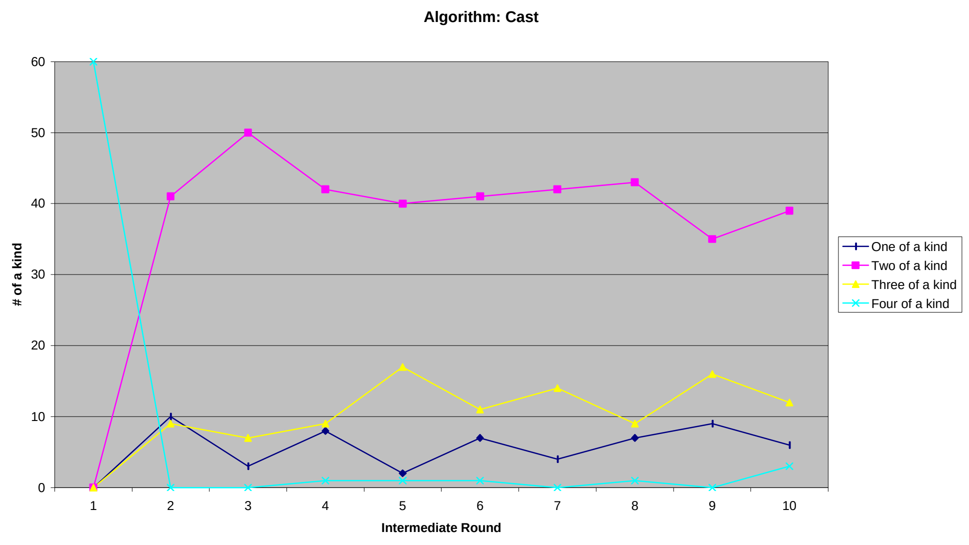
RC6, Rijndael, MARS, Twofish, Crypton, Cast, E2, Serpent, DFC, HPC, Safer+, Loki97, Frog, Deal, and Magenta [13].

Conclusion
My work indicates that Twofish and Loki97 are the two real contenders for the AES title. Deal and Magenta should be thrown out based on my work and Frog and Safer+ should be dismissed because of the recent cryptanalytic work. It is worthwhile noting that attacks have been found against both Deal and Magenta—the two weakest algorithms according to my results. Compared to Loki97, Twofish is faster in both hardware and software which give it an advantage in the official analysis. The Twofish team is made up of several well-known and respected cryptographers. Twofish is a variation of a Blowfish invented by Bruce Schneier. Blowfish has been public for five years and has not yet been broken. A whole suite of papers about Blowfish can be found at [14].

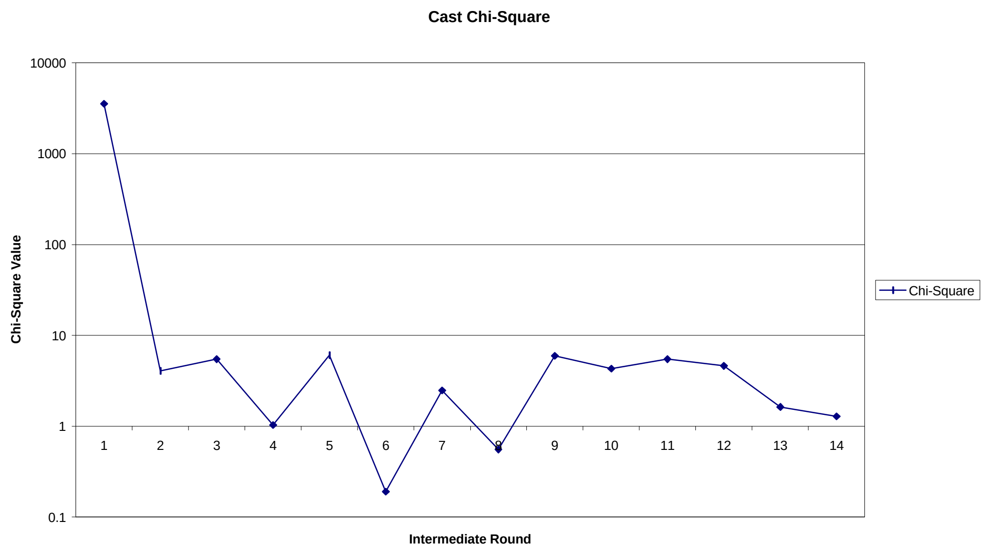
Taking all this information into account, Twofish is the recommended Advanced Encryption Standard algorithm. Loki97 has shown similar experimental results, but it is lacking in speed and efficiency as well as the amount of cryptographic experience represented by the development team compared to that of Twofish. Furthermore, the attack against Loki97 described in [12] put it out of the running. Only time will tell if this instance of using the poker test yields accurate results, but initial work suggests that it is relatively accurate when considering that the two weakest algorithms according to the poker test have already been broken.

[12] Every attempt has been made to collect work with merit from known individuals.
[13] These results have been collected from a variety of resources including the cAESar project based at Université Catholique de Louvain in Belgium and the cryptanalytic team comprised of the creators of Twofish.

Appendix A

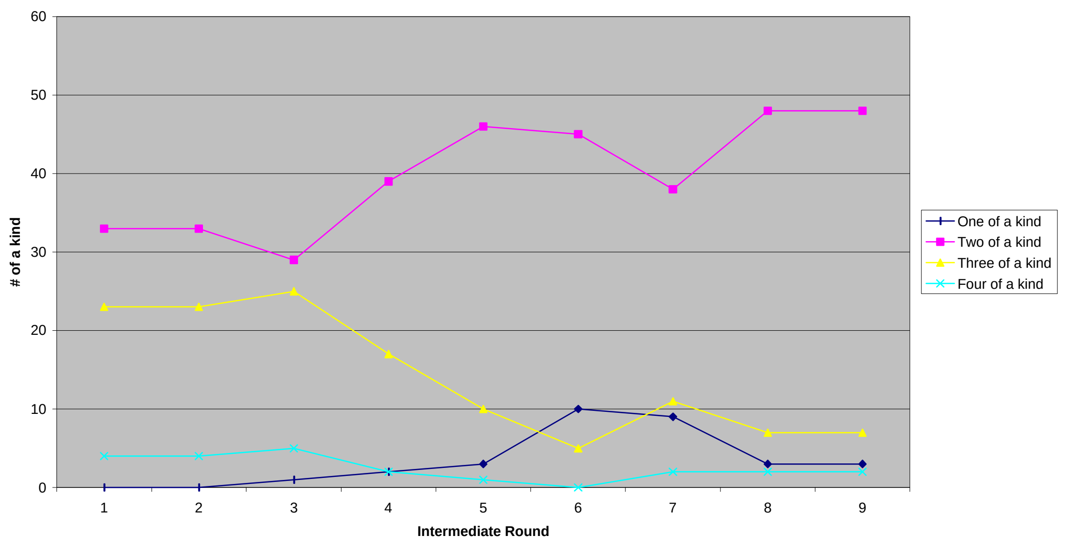


Algorithm: Cast
# of a kind
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind

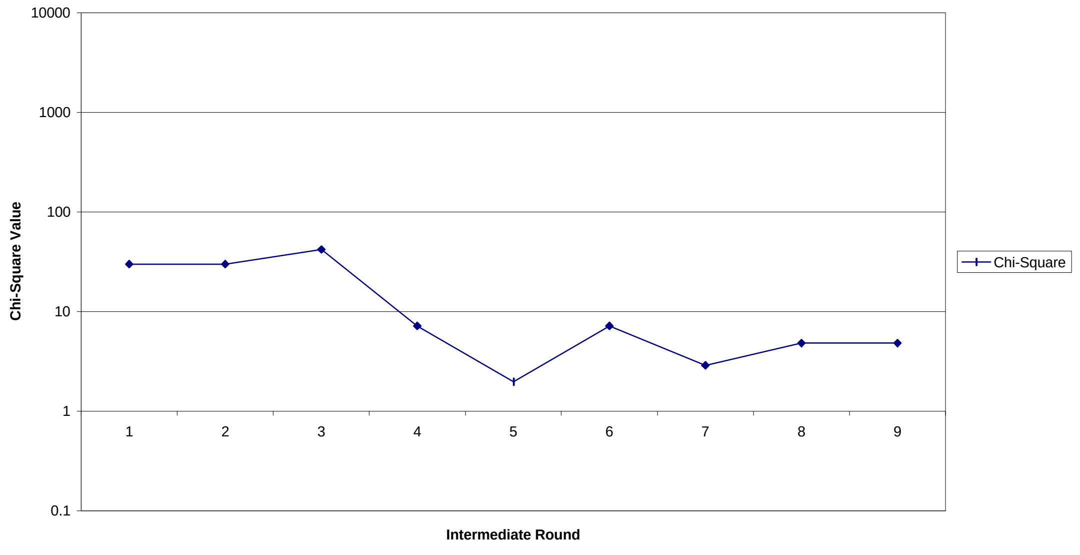


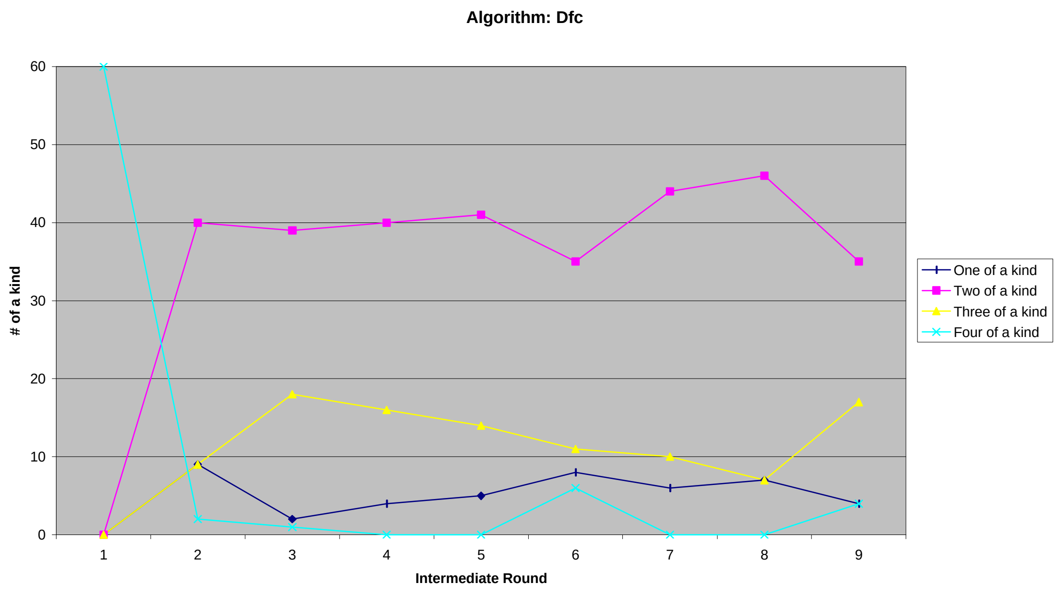
Cast Chi-Square
Chi-Square Value
Intermediate Round
Chi-Square

Algorithm: Deal

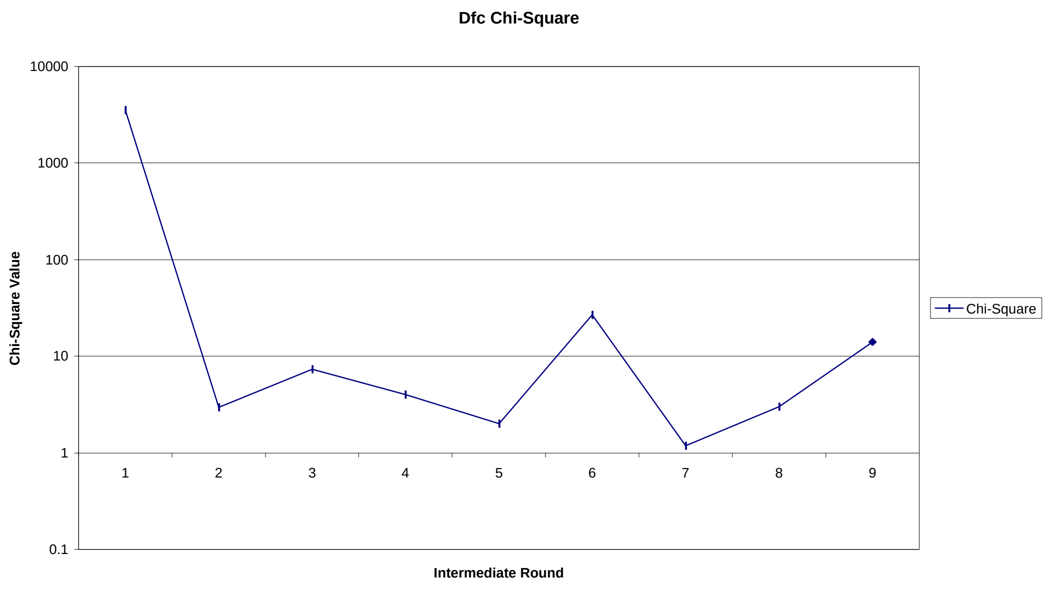


Deal Chi-Square

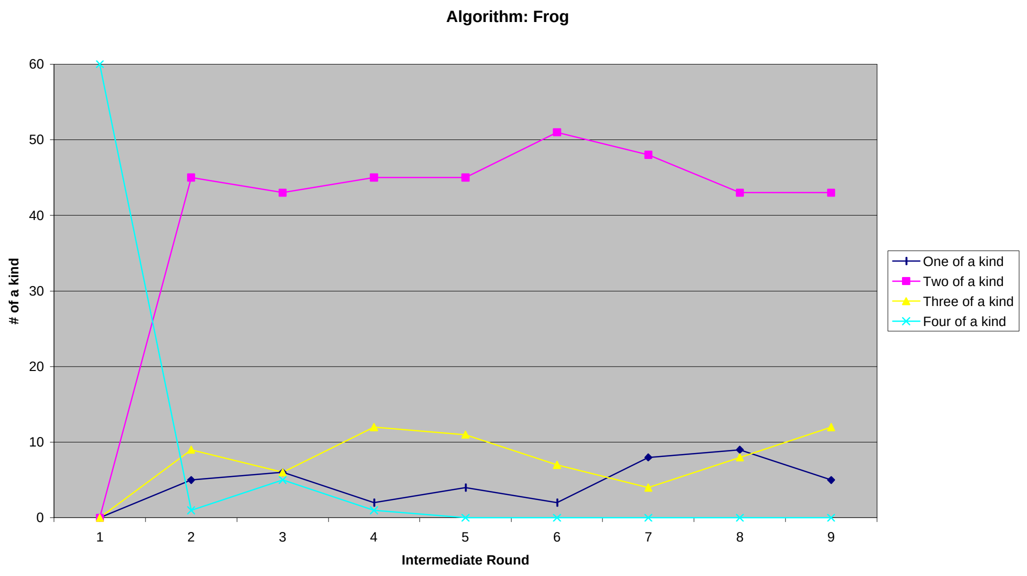


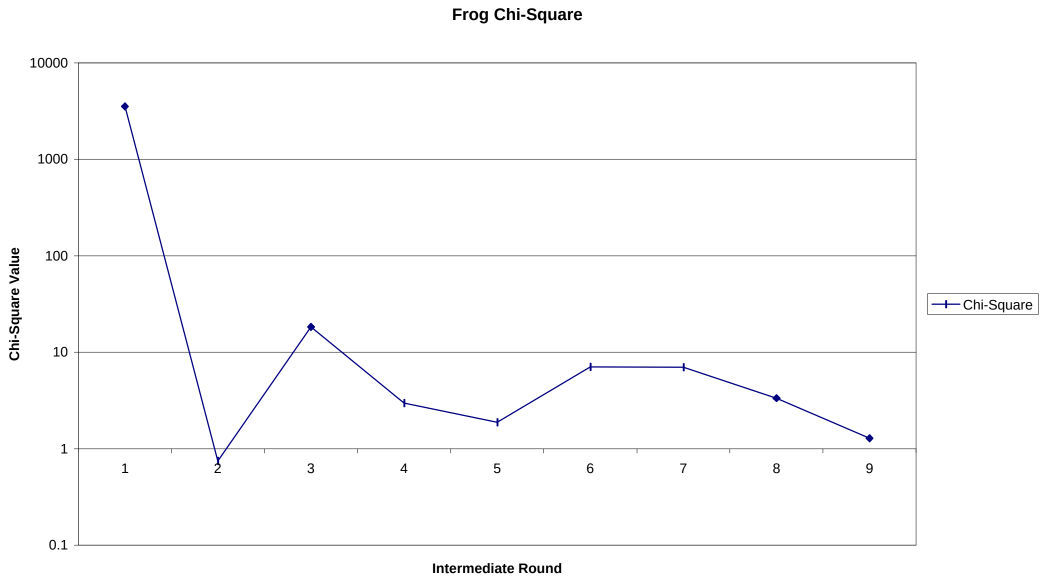


Algorithm: Dfc
# of a kind
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind

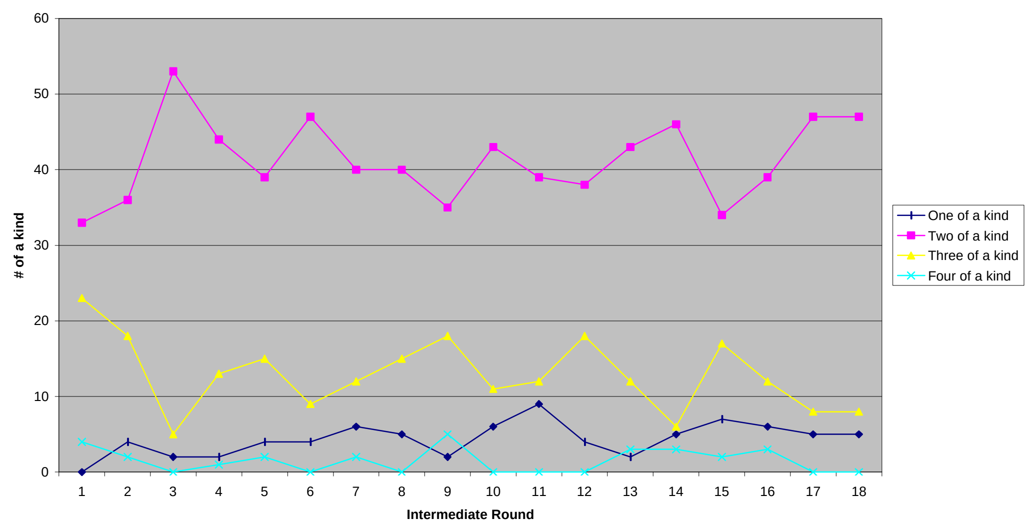


Dfc Chi-Square
Chi-Square Value
Intermediate Round
Chi-Square

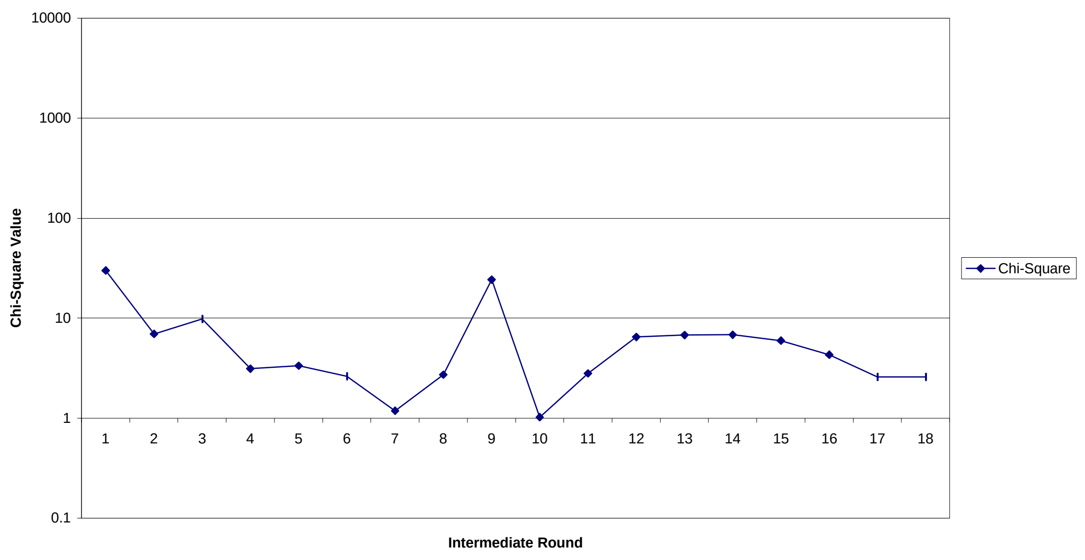

Algorithm: Frog
# of a kind
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind

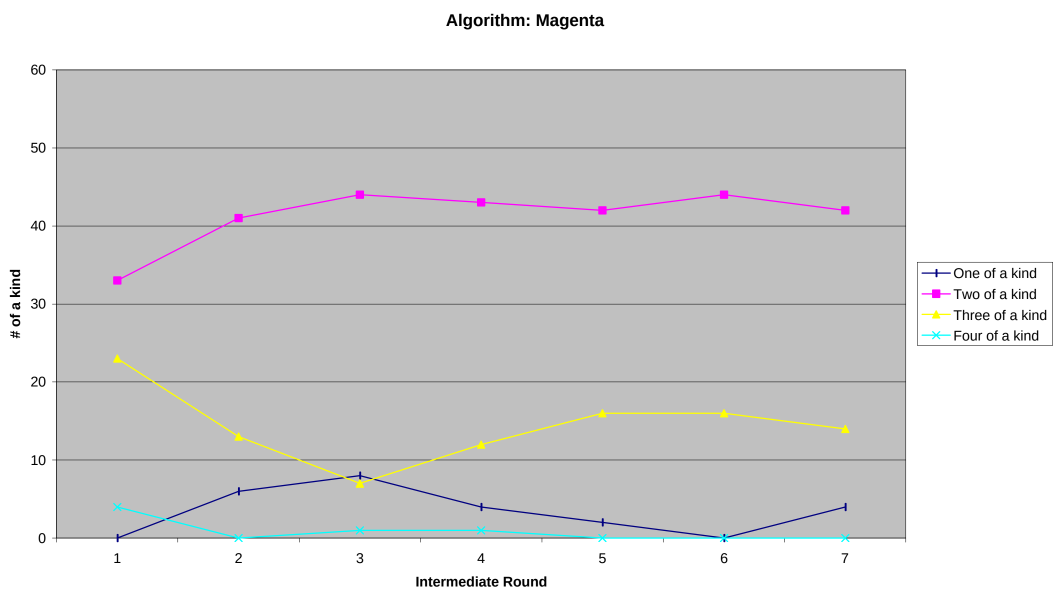

Frog Chi-Square
Chi-Square Value
10000
1000
100
10
1
0.1
1
2
3
4
5
6
7
8
9
Intermediate Round
Chi-Square

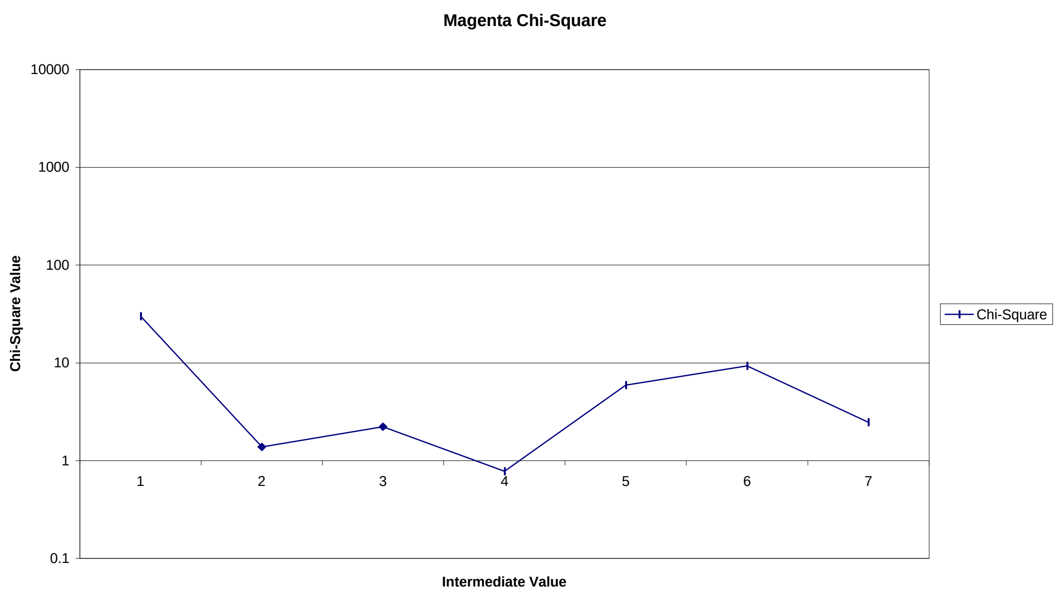
Algorithm: Loki97

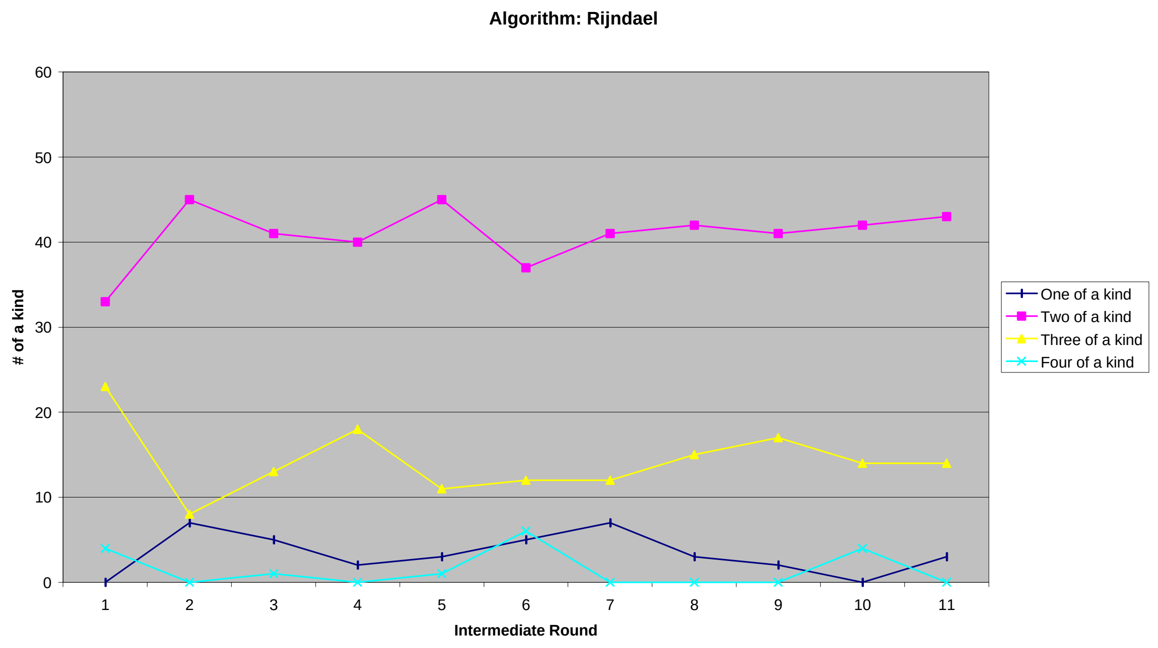


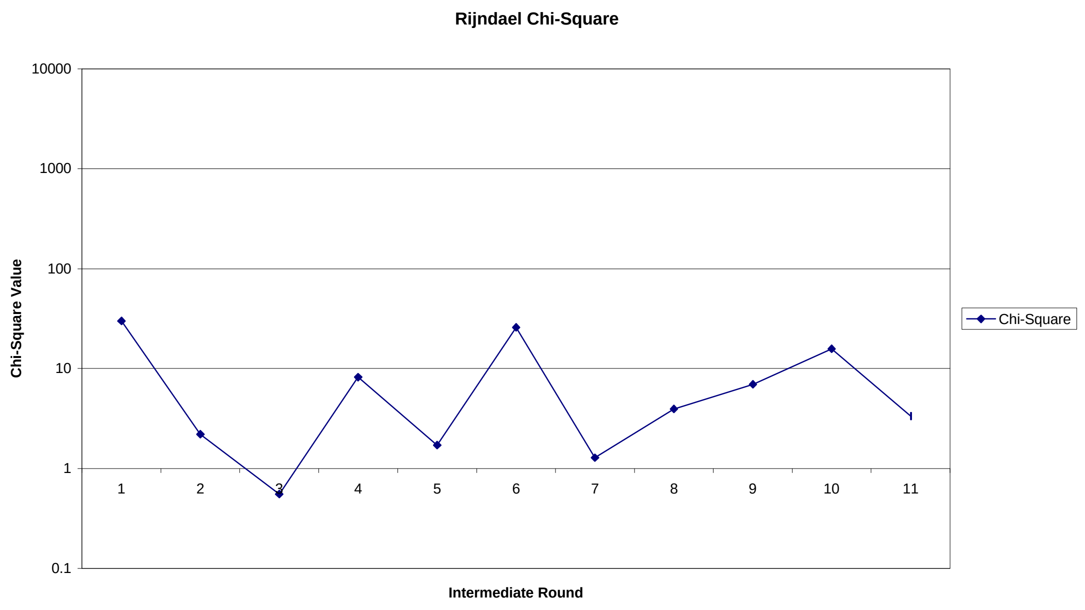
Loki97 Chi-Square



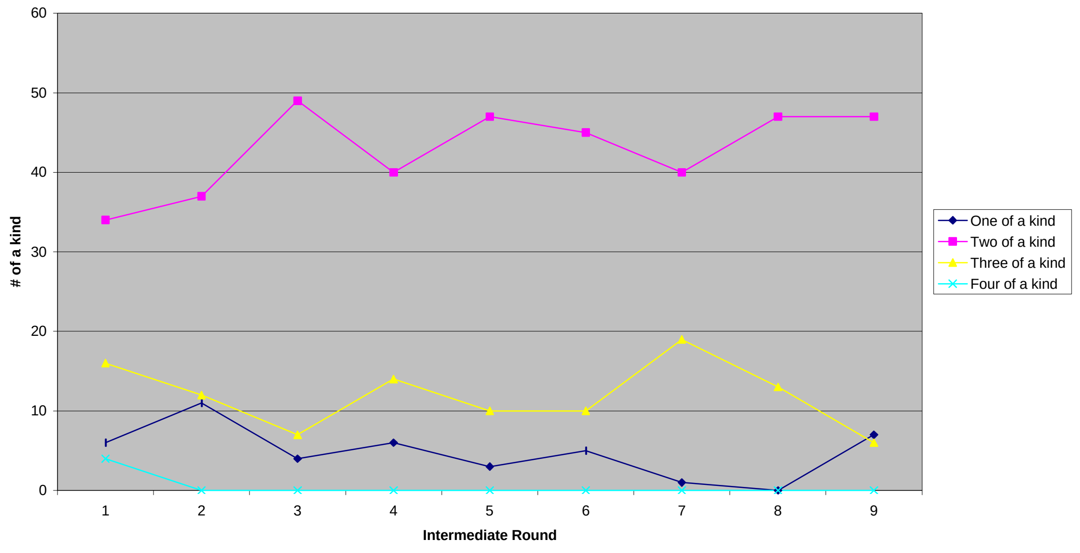

Algorithm: Magenta
60
50
40
30
20
10
0
# of a kind
1
2
3
4
5
6
7
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind


Magenta Chi-Square
10000
1000
100
10
1
0.1
Chi-Square Value
1
2
3
4
5
6
7
Intermediate Value
Chi-Square

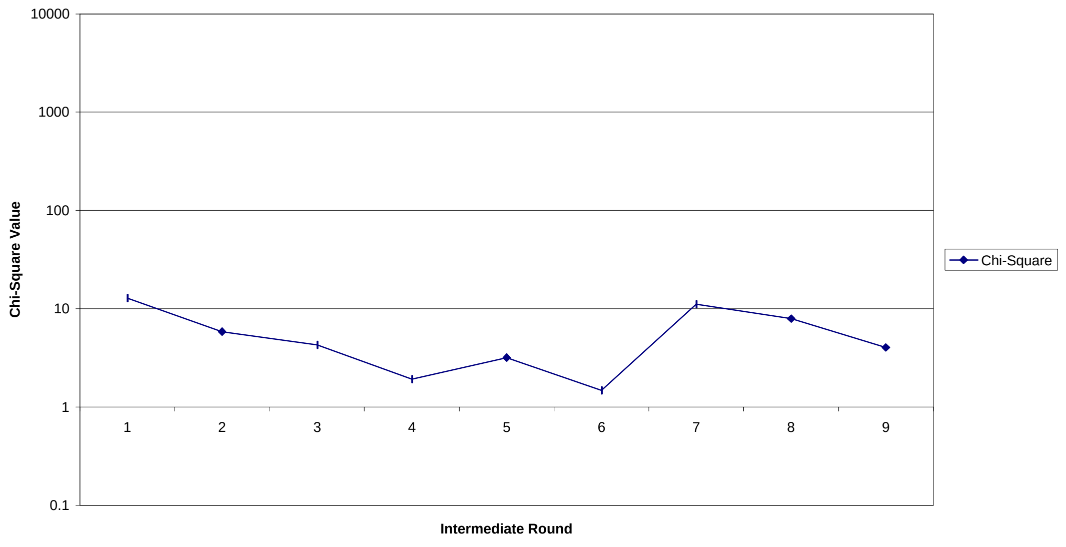

Algorithm: Rijndael
# of a kind
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind
0
10
20
30
40
50
60
1
2
3
4
5
6
7
8
9
10
11

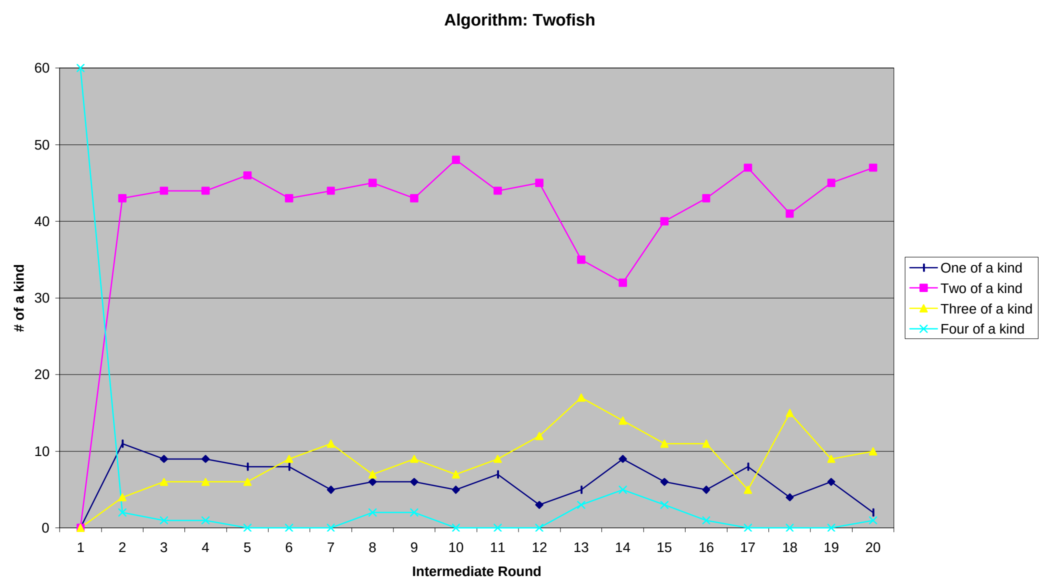

Rijndael Chi-Square
Chi-Square Value
Intermediate Round
Chi-Square
0.1
1
10
100
1000
10000
1
2
3
4
5
6
7
8
9
10
11

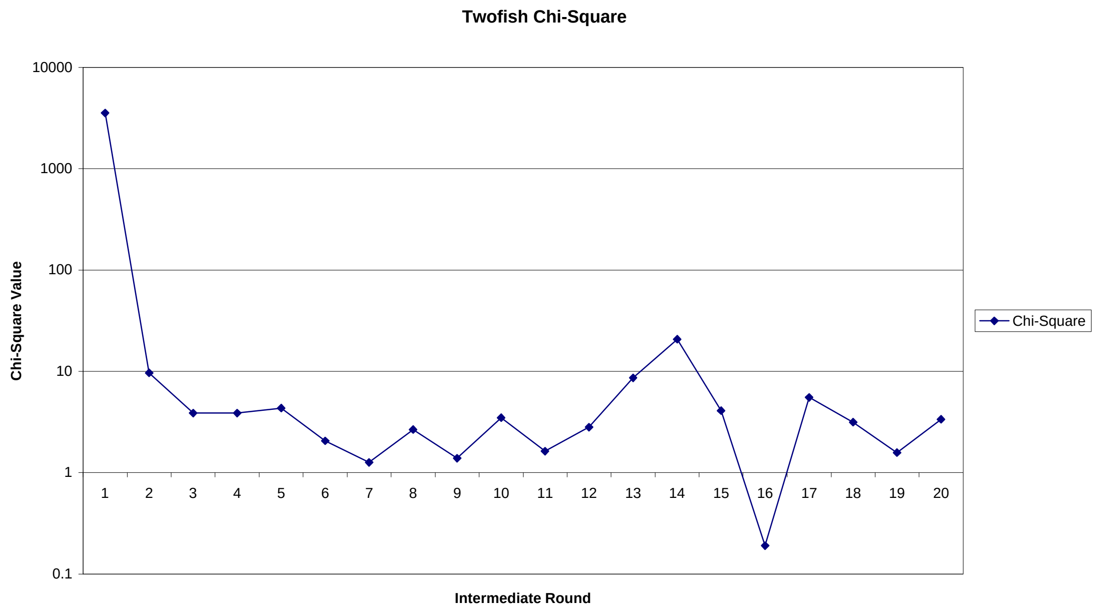
Algorithm: Safterpls



Saferpls Chi-Square




Algorithm: Twofish
# of a kind
Intermediate Round
One of a kind
Two of a kind
Three of a kind
Four of a kind


Twofish Chi-Square
Chi-Square Value
Intermediate Round
Chi-Square

Sources Cited

[1] http://www.eff.org/descracker/
[2] http://www.counterpane.com/pitfalls.html
[3] http://www.counterpane.com/whycrypto.html
[4] Schneier, Bruce, Applied Crytography: Protocols, Algorithms, and Source Code in C. Second edition. John Wiley & Sons, Inc, 1996.
[5] Federal Register; Department of Commerce: National Institute of Standards and Technology. September 12, 1997 (Volume 62, Number 177). Docket No. 970725180-7180-01. Pages 48051-48058.
[6] Knuth, Donald E. The Art of Computer Programming, volume 2: Seminumerical Algorithms. Third edition. Addison Wesley, 1993
[7] http://www.dice.ucl.ac.be/crypto/CAESAR/frog.html
[8] http://www.counterpane.com/frog.html
[9] http://th.informatik.uni-mannheim.de/m/lucks/papers/SEC-DEAL.ps.gz
[10] http://www.counterpane.com/safer.html
[11] http://www.counterpane.com/magenta.html
[12] ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/rijmen/loki97.pdf
[13] http://www.seven77.demon.co.uk/aes.htm
[14] http://www.counterpane.com/blowfish.html