Q

A Proposal for NESSIE$_{v2.00}$

Leslie ‘Mack’ McBride
Tuesday, October 03, 2000

Q

Abstract. Q is a symmetric block cipher. It is defined for a block size of 128 bits. It allows arbitrary length passwords. The design is fairly conservative. It consists of a simple substitution-permutation network. In this paper we present the cipher, its design criteria and our analysis. The design is based on both Rjindael[DR98] and Serpent[ABK98]. It uses an 8-bit s-box from Rjindael with the linear mixing layers replaced with two Serpent style bit-slice s-boxes and a linear permutation. The combination of methods eliminates the high level structure inherent in Rjindael while having better speed and avalanche characteristics than Serpent. Speed is improved over Serpent. This version 2.00 contains better analysis, editorial changes, and an improved key scheduling algorithm. The number of recommended rounds is also increased.

1. Introduction.

The call for an Advanced Encryption Standard (AES) received wide attention. A number of ciphers were analyzed. Among them were Rjindael, Serpent, Twofish[SKWHF98], MARS[IBM98], RC6[RRSY98], and Crypton[LCH99]. These ciphers vary in structure and speed. Of the finalist Rjindael, Serpent and Twofish seem to be the most promising. Serpent is the more conservative design and is slower in software. Twofish is designed with a trade off between s-box memory and speed. Rjindael is fastest but has a high level structure that may lead to future cryptanalysis. Q allows a relatively small amount of memory to be used and the keys to be calculated at encrypt time slowing operation. The time-memory tradeoff is important in limited memory environments. In Q we use a fixed s-box but the two key mixing operations surrounding the s-box equate to a keyed s-box. This representation allows fast operation and the added security of a key dependent s-box. On the lighter side the name of this cipher was derived as a file extension. On my system *.Q doesn't conflict with any of the other file types. It is also the first letter on the keyboard.

2. Cipher Structure Specifications.

All decimal and hexadecimal numbers follow the normal Arabic convention of most significant digit on the left.

All descriptions are 'Little-endian' ie. Smallest byte first (Figure 1). This is the format used in the x86 family of CPUs.

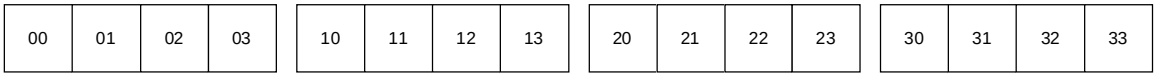

Figure 1 Byte Order

These bytes are grouped into words (Figure 2).

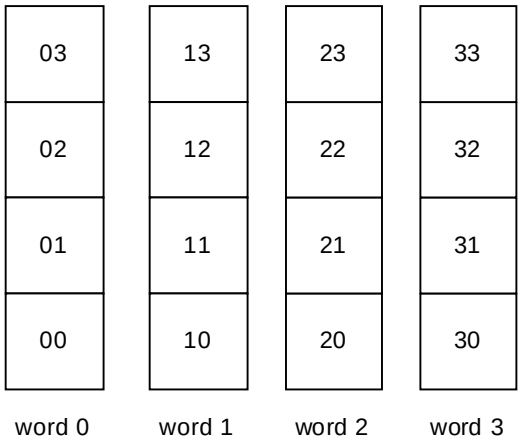

Figure 2 Word Grouping

The cipher consists of several rounds of operations. Each round consists of:

1) Key A Mixing
2) Byte Substitution
3) Key B Mixing
4) Bit-slice substitution A
5) Key n Mixing
6) Permutation
7) Bit-slice substitution B

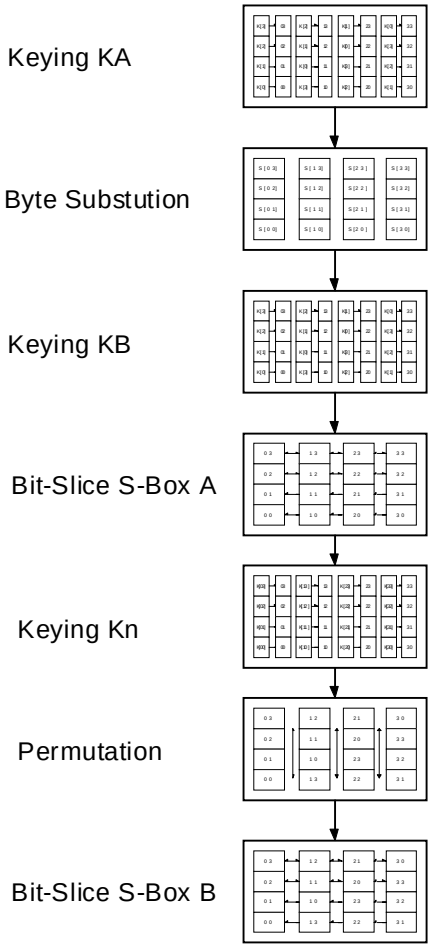

Figure 3 Round Function

In an actual implementation steps 1, 2, and 3 will be combined for faster round functioning. This will result in four different byte substitution tables. This lowers key agility but provides faster round functioning. These factors should be weighed when developing an implementation.

The key mixing steps are simple. 16 bytes of key are XORed with 16 Bytes of data. Keys n (where n is the round number), W1 and W2 are defined as four independent 32-bit words (Figure 4). The keys Key A and Key B are four identical 32 bit words used in each round (Figure 5). Notice that the keys Key A and Key B are shifted by the permutation when applied to words 1, 2 and 3.

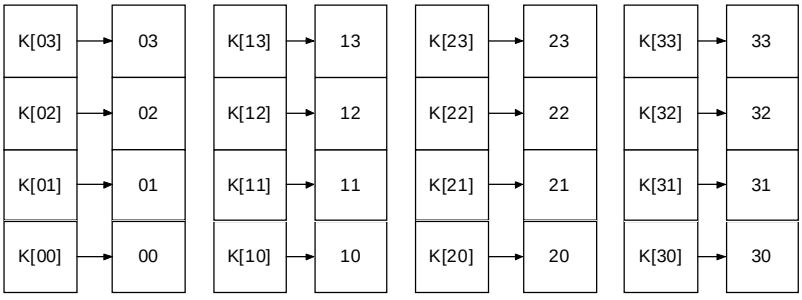

Figure 4 Key Mixing Of Kn

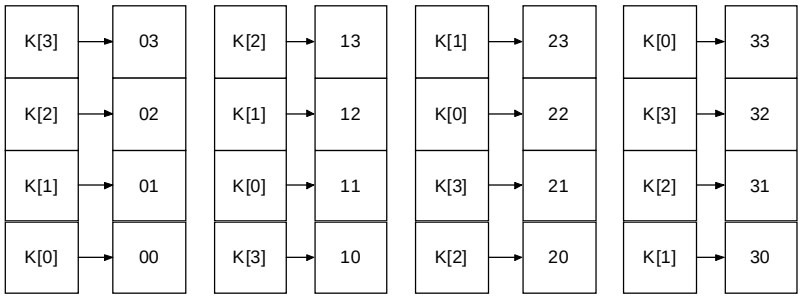

Figure 5 Key Mixing of KA and KB

The byte substitution (Figure 6) is presented in tabular form suitable for direct use in C programs in Appendix A. This byte substitution is taken from Rjindael.

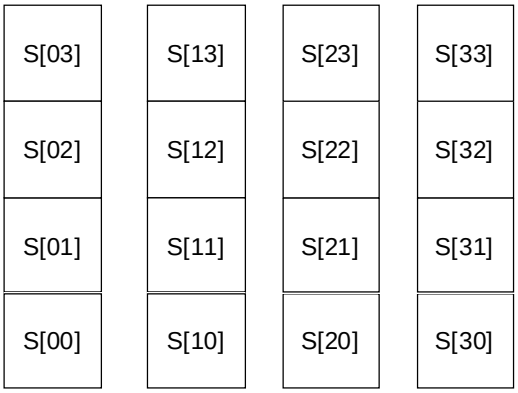

Figure 6 Byte Substitution

The KA, Byte Substitution, KB sequence can be combined to an equivalent set of four s-boxes (Figure 7) for implementation purposes on processors with sufficient memory. This representation must be weighed against the cost of keying the table.

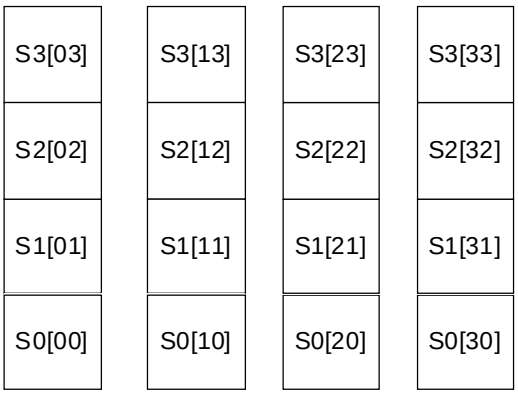

Figure 7 Combined KA, Byte Substitution, KB as four s-boxes.

The bit-slice substitution (Figure 8) takes the data as four 32-bit words and performs the operations necessary to equate to a 4-bit s-box operation. The bit order of the bit-slice s-boxes is word 0 as least significant bit and word 3 being the most significant bit. There are three bit-slice s-boxes described in Appendix A. The instructions to carry out the bit-slice s-boxes are listed in Appendix B.

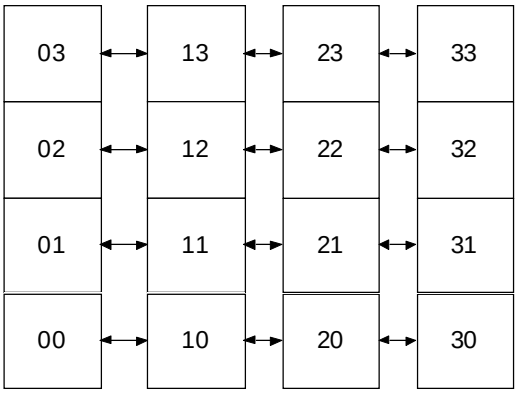

Figure 8 Bit-slice s-box A

The permutation (Figure 9) rotates takes the words 1, 2, and 3; and rotates them by 1, 2, and 3 bytes respectively.

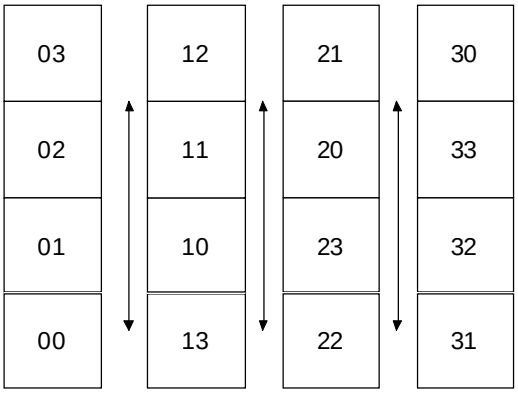

Figure 9 Permutation

Finally the second bit-slice substitution (Figure 10) performs a second 4-bit s-box operation with a different s-box.

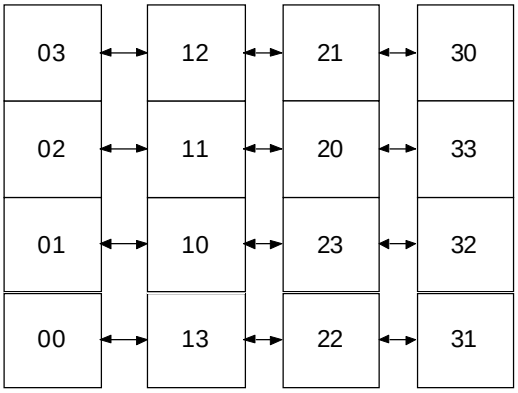

Figure 10 Bit-slice s-box B

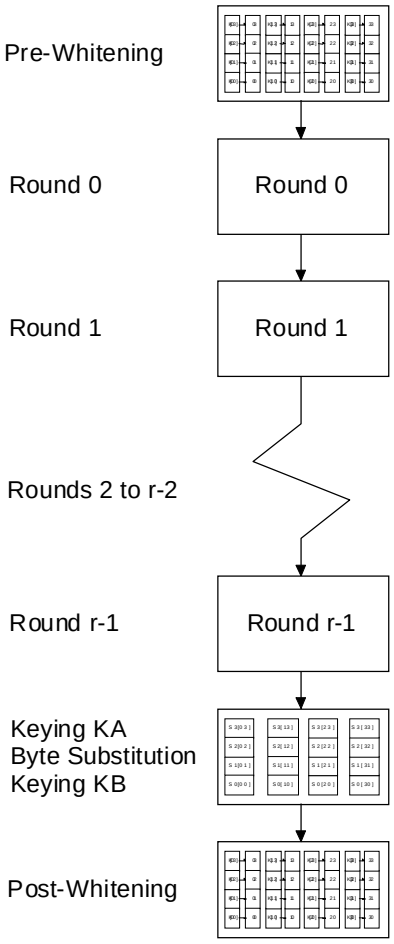

Figure 11 Cipher structure

Now that each step in the round has been described we can define the cipher in terms of the rounds (Figure 11).

1) First a pre-whitening step with KW1 is performed.
2) r rounds are then carried out
3) Next a KA step is done after the last round
4) Then a byte substitution step is performed
5) Followed by a KB step
6) Finally a post-whitening step with KW2 is performed

Note that in the diagram steps three, four, and five are combined as described in the round description. This makes the algorithm very symmetrical. Any analysis applied to the encryption function applies equally to the inverse function.

An important property of this cipher is that the number of rounds can be viewed as 3r+1 with KA and KB being repeated keys.

The number of round is selected as 8 for low security applications and 9 for high security applications.

3. The Inverse Cipher

The inverse is, as noted above, structurally very similar to the forward version. Only two steps with permutation and key mixing are reversed. All other steps occur in the same order but with the key orders reversed and different s-boxes. The inverse of each s-box is listed in Appendix A.

The bit-slice s-boxes obviously require different instructions in the inverse than the forward version. These are noted in Appendix B. Instruction sequences corrected for permutation are also listed Appendix C.

4. Key Scheduling Specifications

For keys longer than 256 bits a preprocessing step involving the polynomial in GF(2) $X^{256}+X^{193}+X^{113}+X^{6}+1$. This polynomial is used to divide the key in GF(2). Keys less than 256 bits will not be affected by this function. As this polynomial is primitive it is will provide good mixing. This is primarily for ASCII keys. It is not cryptographically strong but it is useful. No strength above 256 bits is created.

For keys there are two cases, those with less than or equal to 128 bits and those of greater than 128 bits and less than 256 bits. In the case where the key is less than or equal to 128 bits, the high order half of the bits is assumed to be zero.

A constant C is used in key setup. This is one word from the fractional part of the golden ratio $(\sqrt{5}+1)/2$ 0x933779b9. This value was chosen because it was used in Serpent. This step was originally before the byte substitution and after the key mixing. It was moved to a location after the byte substitution and before the bit-slice s-box.

A round counter is XORed at the beginning of each keying round. It was previously considered and rejected for the 128-bit key, however it is necessary for the 256-bit case so it is now reapplied in this version.

The operation of the keying is similar to a single round of encryption applied to the low order half of the 256 bit key. The steps of keying are as follows:

1) A single byte round counter is XORed to KL[00] (this step is new it insures that each round key is unique).
2) XOR the high order half of the key to the low order half of the key, that is KL^=KH.
3) Perform a byte substitution on the low order half of the key.
4) XOR constant C by word KL[0], that is KL[0]^=C. (This differs from version 1 where this step was before the byte substitution).
5) Perform a bit slice s-box (using s-box C) operation on the low order half of the key.
6) Perform a permutation
7) Take off key words

The key byte ordering is as shown in Figure 12. This convention is the same as before.

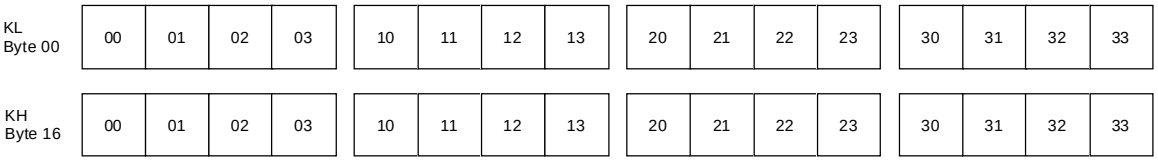

Figure 12 Key byte ordering

The round counter is initialized to zero and incremented at each step. The byte values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16. This sequence can be continued to 255 before repeating. The XOR operation is shown in Figure 13.

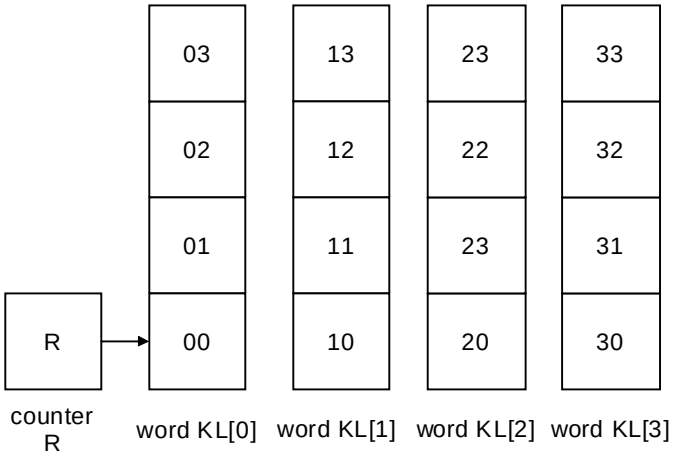

Figure 13 XOR byte counter

The mixing of the two key halves is shown in Figure 14. Notice that for KH=0 step 2 is a null operation. This is very important in memory limited environments.

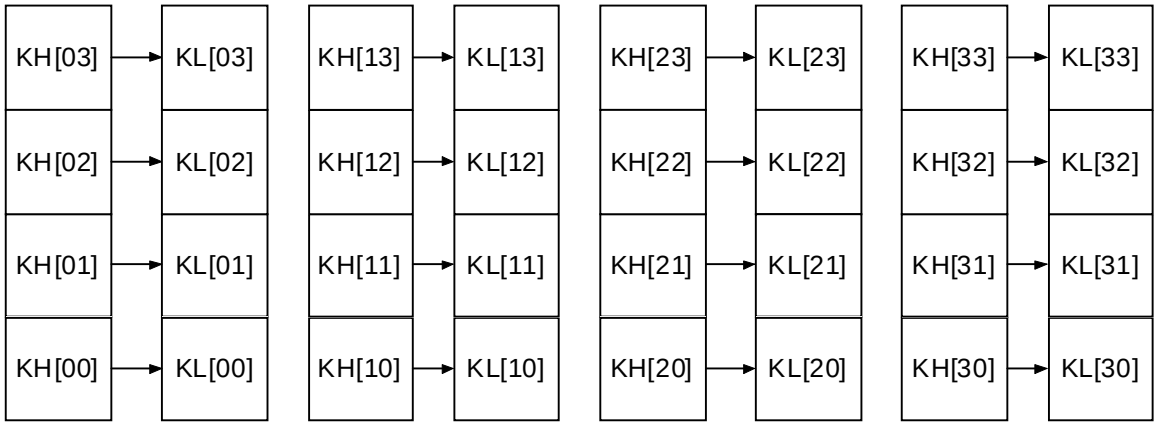

Figure 14 XOR high half of key to low half - KL^=KH

The fixed value XOR is shown in Figure 15.

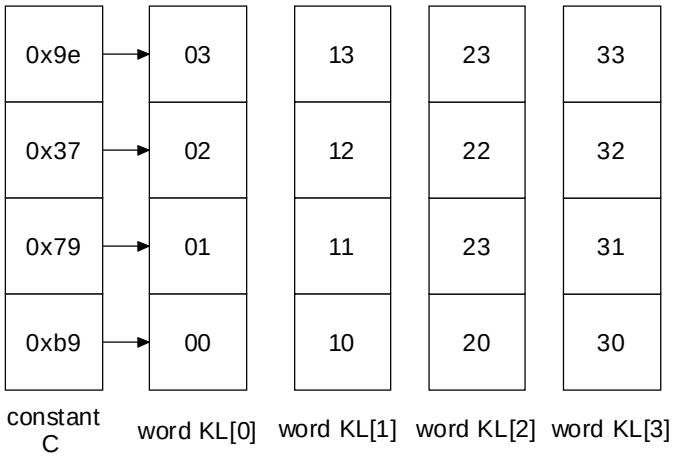

Figure 15 Constant XOR

The byte substitution and permutation are the same as for the round function. The bit-slice s-box C is from a different class than A and B. The key information produced is taken off in 4 word increments. The use is as follows:

Discard, KW1, KAB, K0, K1, …, Kr-1, KW2

KAB is broken down into four words with KA=word 0 and KB=word 1. When used they are four identical words with the permutation applied. The initial discard step is new. It was determined that two rounds are needed for good mixing. Previously KW1 was considered sufficiently mixed. In light of better analysis this was changed. Single bit changes will potentially propagate to all bits in two rounds.

5. Implementation Notes.

If memory is available it is much faster to setup four lookup tables if any quantity of data is to be processed. This avoids a set of 8 XORs in each byte substitution step.

A simple method of checking the consistency of the tables byteS and invbyteS is I=byteS[invbyteS[I]] for all I. Another method is constructing an XOR table. This shows any random inconsistency with XORs greater than 4 or (0,0) greater than 256 or the zero row (or column) not zero. A final method is summing the boxes. The sum is 32640 for any 8-bit bijective s-box. This applies if the tables are used directly or with the KA/KB modification. Any or all of these methods may be used to verify correct table distribution. In high-speed implementations this step will be removed once verification is completed. Actual validation with the test vectors should still be done.

The ordering of inputs and outputs in the sequences given in Appendix B are NOT the sequences that would produce the actual s-boxes. As noted they are permuted. The corrected version is shown in code in Appendix C. To achieve correct implementation in a true bit-slice algorithm only one test vector is needed for each independent bit-slice s-box. But the ordering of inputs and outputs must be thoroughly tested for interaction with the other components.

The reversibility of the key schedule provides an important check of functionality.

Test vectors at intermediate points in each round should be used during development.

The byte counter XORed in the key generation should be XORed into the least significant byte. This may be a problem on some architectures. On 'big-endian' architectures a byte is aligned with the most significant, instead of least significant, byte of the word.

6. Design Criteria.

One of the primary design criteria was that the cipher be faster than Serpent. Another design goal was that the cipher be immune from differential [BS93] and linear cryptanalysis.

The selection criteria for the four bit s-boxes are the same as those for Serpent.

1) All s-boxes are bijective.
2) The s-boxes consisted of no more than one Boolean equation of order 2 when expressed in algebraic normal form. All other equations must be order three.
3) The XOR table[BS93] has no entry higher than 4.
4) No single bit change results in a one-bit change in the output.
5) The minimum hamming distance[MS77] between any one output and any one input or its inverse is 6.
6) The Linear Approximation Table[OL95] has no entry greater than 4.

After thorough analysis of the Serpent s-boxes it was determined that only 14 classes of s-boxes exist. In these class all s-boxes have an inverse in a different class so there are only 7 pairs of classes. This was accomplished by enumeration using the methods similar to Yang[YJH91] in his enumeration of DES s-boxes. Specifically all equations of order 2 & 3 that are balanced and have non-linearity of 4 were found. Then those pairs were found whose hamming distances are 8. These pairs then had one more equations added to form triples that were checked for orthogonality. Finally a fourth equation was added and checked that the properties above hold. Several optimizations were made to the search since the order of the equations does not matter. Finally a set of numbers representing all 67584 possible s-boxes and the 24 permutations of their outputs was produced. Permutations of inputs and outputs and negations of inputs and outputs yield the s-boxes in a class. Under the negations of the inputs and outputs any permutation may be converted to a form with zero as a fixed point. This will be referred to as the 'natural form' of the permutation. This search of classes was inspired by Dag Arne Osvik in private correspondence when he suggested that the form of the enumeration might suggest that there are only 11 s-boxes as 67584=24*256*11.

Only 3 broad classes of these permutations exist when taken over the set of all (non-degenerate) linear transformations of inputs and outputs. This is obvious since order 2 and order 3 equations remain order 2 and order 3 respectively over the set of all linear transformations. The s-boxes can be grouped into those with one equation of order 2 in the s-box and its inverse. Those with no equation of order 2 in the s-box

and its inverse. And finally those with one equation of order 2 in the s-box but not in the inverse (only one class and its inverse).

In Serpent the s-boxes used were selected arbitrarily. The s-boxes here were selected with speed as a prime consideration. A search was conducted using a method developed by Dag Arne Osvik[ODA00]. A state machine modeling the x86 class of processors was adopted. Only 5 registers are used due to restrictions of the general programming environment. Only 6 operations are used NOP, OR, AND, XOR, MOV, and NOT. Certain heuristics are used to prune the search tree[SS98]. Since the process was still very slow randomization was done to prune the tree further[AC99]. The output was examined to find the fastest pairs of s-boxes and inverses. The search was broken down into those s-boxes with an order 2 equation and those with no order 2 equation. No s-boxes with all equations of order 3 were found during the runs. The search depth was limited to 18 instructions on those runs. It was concluded that on this state machine Serpent S3 was optimal. A quicker equation in natural form was found for the S2 s-box. S-boxes with one direction requiring 15 instructions and the other direction requiring 16 were also found. There should be s-boxes with both directions being 15 instructions but none have been found. All of the 15 instruction forms are from the same class as Serpent S2 or its inverse's class.

One of the s-boxes used is actually Serpent s-box 3 (s-box B). S-box B was chosen as it has all equations of order 3 in both forward and inverse forms. The other s-box (s-box A) was found using the search. It is equivalent to the class of s-boxes that include Serpent s-box 2. The s-box C used for the keying is another s-box found from the search of s-boxes. The condensed data from the run finding these the two s-boxes used is found in Appendix B. The s-box A with its inverse are the first pair found. The keying s-box C is the first s-box not from the class represented by that s-box. No fast corresponding inverse was found. However an 18-instruction form was found for the inverse of s-box C in a new search. The output of the search program indicates how it was initialized. This allows duplication of the results. The program to do so will be made available.

A large number of bit-slice s-boxes should be avoided as the number of classes is small and leads to long code. The s-boxes selected may not be ‘optimal’ in the sense that no faster s-boxes exist. However the code sequences are faster than the s-boxes used in Serpent. The three s-boxes used here are all from different classes. The s-boxes are used in the natural form although it should be possible to find fast pairs with one bit set in the zero position. The use of s-boxes with zero as a fixed point is not necessarily a weakness since ANY s-box can be converted to this form with the appropriate XOR masks on input and outputs. The key material being inserted between the two bit slice x-boxes effectively changes them to forms without zero as a fixed point.

The eight bit s-box is taken directly from Rijndael. A search was attempted for an s-box that had no single bit output changes for a single input bit change but the resulting s-boxes had linear and differential characteristic that were about double those for the s-box used. Various hill-climbing and randomization techniques were also tried [MCD98][MCD99][MCD97][MBCCD99][MZ92][KF99]. Using a known s-box increases the confidence that no trap doors are located in the cipher.

The additions of KA and KB were introduced to provide variability to the 8-bit s-box. This idea was introduced by Harris and Adams[HA98]. They recommended key dependent XOR and permutation operations on s-boxes as a way to increase the security of ciphers. The rotation of KA and KB, as it is introduced into the bit-slice s-boxes, is designed to ensure that all the changes to the bit-slice s-box are different. If the rotation was not present then each s-box input would be exactly complemented or exactly the same as the natural form.

The pre- and post-whitening steps were introduced with DESX[KR69]. In fact it can be seen that any non-key dependent operation can be peeled off of the cipher.

The permutation was selected as a simple rotate of the words. This provides even diffusion of the words. The cipher is very uniform. Any differentials will have rotated components. However this is not seen as a weakness. By keeping symmetry no new imbalances in the differentials are introduced. This was experimented with on 8-bit s-boxes. The conclusion was that breaking the symmetry destroys the low

linear probabilities. This was for the eight bit s-box used. Another s-box may fair better. In this case no additions were made to destroy symmetry since the keying step will add non-symmetric data.

The keying operation was given the same general structure as the cipher. One round is not sufficient to provide one-wayness with a known second half. This structure equates to keying a similar cipher with identical round keys. The second s-box is dropped to increase keying speed. This is also a factor of the number of fast s-box classes available.

The keys are output in the order they are first used. This order is important. There is not sufficient mixing to use KA and KB before two iterations. After two iterations a single bit change in the key affects at least 4 bits meaning there is a high probability that at least one bit will change in KA or KB.

The constant in the keying step is taken directly from the Serpent keying procedure. Selecting a good known mathematical constant limits the search basically to three numbers, pi, e, and the golden ratio. There are an infinite number of mathematical constants but these are the ones most commonly used in cryptography. The use of the constant ensures that the s-box acts as if zero was not a fixed point for half of the s-boxes. This number may vary as there are 64 bit-slice s-boxes and the operation can be seen as affecting either one set of 32 or both depending on the most convenient point of view.

The keying procedure was changed slightly to eliminate a class of weak keys. This class existed when a 256-bit key was used. All bytes in each word were required to be equal after the constant was added. This class left all words in each round key equal. This was a class of 2^{64} weak keys. By moving the constant to a location away from the KH/KL mixing point, it no longer commutes with the key mixing. The round counter was added to ensure that all round keys are different. This is important in preventing attacks that might be based on such keys. Note that KA and KB come from two words of a single round key and may be the same.

The number of rounds is selected as eight for low security and nine for high security. The reason for these selections relates the number of total rounds when viewed with 3r+1 taken as the number of rounds. For the low security case there are 25 s-box layers and for the high security case there are 28 s-box layers. Each s-box layer is accompanied by a corresponding key. For many of these rounds the key is the same (KA/KB) but it is not known to be exploitable. These values were chosen to provide a speed gain over Serpent while providing equivalent security. The number of rounds has been increased from version one. The security margin was inadequate. 7 rounds are required for differential security and 8 for linear security. The high security case adds an extra round as a precaution.

7. Security of Q

a) General Security

To determine security we must define feasible. Every one will have their idea of what is ‘feasible’.

It is conjectured that no ‘feasible’ attack requiring less than 2^64 plaintext/ciphertext elements exists except for the obvious plaintext collision attack (probabilistic dictionary attack) which occurs with the frequency determined by the birthday paradox (text regularities may improve this). This is true for any 128-bit block size cipher.

The Method of Formal Coding attack may also be used. With this number of total bits MFC requires an infeasibly large amount of memory. The number of terms is about 2^{K-1}.

b) Differential Cryptanalysis

The first and most obvious attack is due to the fact that a one bit change in the input to the 8-bit s-box can yield a one bit change in the output. To get a working attack we need a one bit change in the 8-bit s-box that maps to itself. The choices are bits 0, 5 and 7 (Table 1).

Table 1 Input (left) vs. Output XOR with one bit changed

	0	1	2	3	4	5	6	7
0	M		X	X				X
1			X				X	X
2						X		
3		X			X			
4	X			X		X		
5		X	X			M	X	X
6	X				X	X		
7		X		X				M

Next we need a one to two bit characteristic that reverses itself. Additionally this characteristic must be in words 0 and 2 or words 1 and 3. This requirement is due to the permutation. In this case we find such a characteristic with word 1 as the one bit and words 1 and 3 as the two bits (Table 2).

Table 2 Two-One bit characteristics of 4-bit s-boxes

	s-box A	s-box B	is-box A	Is-box B
In 0xA Out 0x2	4/16 (-2)	2/16 (-3)	2/16 (-3)	4/16 (-2)
In 0x2 Out 0xA	2/16 (-3)	4/16 (-2)	4/16 (-2)	2/16 (-3)

Next we need a recurrence. We define four affected bytes A, B, C and D. We can fix a location based on the permutation and have the rest of the cipher revolve around it.

	C		D
	A		B

In examining Table 3, it can be seen that the best one round characteristic has probability 2^{-18}. This expands to a characteristic of 7 rounds of 2^{-126}. With the final byte substitution (two active s-boxes) this is 2^{-140}. With one additional round the probability is 2^{-158}. The 7 round characteristic is above the minimum characteristic as quoted in Serpent of 2^{-120}. In Table 3 several items are listed as No DC meaning that D cannot expand to C.

Other characteristics are possible but none have significantly higher probabilities. For example a second bit in each byte could be used raising the probabilities to 1/64 for the byte substitution but this is offset by the added active bit-slice s-boxes. Differentials with more rounds are possible. In fact in the 7 round case it would seem that any good differential is going to be significantly more complex.

This attack can be applied to each of the bits 0, 5, and 7 in 4 different ways leading to 12 sets of differentials.

Table 3 Active bytes and $\log_2$ differential probabilities

Input	ByteSub	Bit Slice A	Perm	Bit Slice B	Output (total)
A	A,-7	AB,-3	AD,0	No DC	No DC
AB	AB,-14	A,-2	A,0	AB,-2	AB,-18
AC	AC,-14	ABCD,-6	ABCD,0	AC,-6	AC,-26
ABC	ABC,-21	ACD,-5	ACB,0	ACD,-5	ACD,-31
ABD	ABD,-21	No DC	No DC	No DC	No DC
ABCD	ABCD,-28	AC,-4	AC,0	ABCD,-4	ABCD,-32

Now that the best characteristics are calculated, we observe that the classic differential attack to Feistel networks does not apply to an SPN. However the attack from Chen and Tavares[CT98] is applicable. In that attack the number of s-boxes active in the second and last rounds are calculated when one s-box in round one is active. The differentials described above help in that respect. When they hold, the input to the first round s-box must be two values and the output from the first s-box is also one of two values. But we note that no high probability differentials with only one active s-box exist. An avalanche occurs quickly.

Noting that an attack with one active s-box in round one fails spectacularly, we again look at our iterative characteristic. There are a number of false characteristics that will give identical results. For example if bit 1 is chosen in round one then the outputs may actually be bit 7 and propagated through the network till they swap back to 1 via 3 and 4. This observation makes the signal to noise ratio high. With the extremely low probability of the characteristic we will need a huge number of pairs to get a satisfactory result. This property exists because of the regularity of the cipher. Breaking symmetry at any point prevents the false differentials from being the same probability.

c) Linear Cryptanalysis.

Taken from Keliers, Meijer and Tavares[KMT99] result for a 16 layer SPN has differential probabilities on the order of 2^{-41}. For the 8-bit s-box the highest linear characteristic with any number of inputs has probability 16/256 (1/16). The highest probability single bit linear characteristic for a four bit s-box is 2/16 (1/8) and for multiple bits is 4/16 (1/4). For one round the probabilities are for one bit active (1/1024)*4 and for more active bits (1/256)*4. The theoretical minimum (with multiple active inputs) for 8 rounds is 2^{-45}. However using the Table 1 from Serpent and extrapolating to 21 rounds the result is 2^{-102}. A more practical bound (single input) on the eight round probability is 2^{-60}. This however only allows for one active s-box in each layer. Due to the avalanche this number is at least 2^{-64}. Even this is probably an over estimate but it would not allow an attack. Some of these low numbers may seem disturbing but this is a guaranteed lower bound.

Table 4 Input (left) vs. Output single bit LAT (over 256) for byteS

	0	1	2	3	4	5	6	7
0	12	0	14	12	8	-4	4	12
1	2	8	2	6	-2	8	-16	-2
2	-8	2	6	6	12	-16	2	-2
3	2	2	4	0	12	6	2	4
4	-12	-2	-6	-2	-8	-10	0	8
5	6	-10	-2	-12	2	0	-8	12
6	4	-4	-12	16	12	-8	-12	4
7	-12	-12	16	14	-8	12	-4	-4

As can be seen from the single bit LAT Table 4 most of the probabilities are lower. Only a few good one-bit characteristics exist.

Sample paths are
0,0 and 6,6 prob. $2^{-31.7}$
7,2,5,7,2,5,7,2,5 higher prob. $2^{-28.8}$

Next we can construct a path through the bit-slice s-boxes. The simplest path is through word 2. This path is iterative with probability 1/8 for both s-boxes. This allows a characteristic path in the range of 2^{-33} for 8 rounds of both s-boxes. When we combine these two the bound is indeed $2^{-61.8}$. This single bit characteristic includes 9 bits of KA and 9 bits of KB as well as 8 bits of round keys and two bits of whitening. And this only gives us a parity of these bits. Note that linear cryptanalysis assumes independent variables. 3 bits each from KA and KB are in fact repeated. Multi-round linear characteristics will not necessarily follow the high probability path. In fact interference will occur on every round in the byteS and on every even round as the linear bit rotates back to the position synchronized with other bits from the bit-

slice s-boxes. In actual four s-box (1 and 1/3 rounds) testing the probabilities did not hold, probably due to the interference effect, therefore the minimum probability of a characteristic is at least 2^6 times worse. This is the minimum possible probability. No linear characteristic will have a probability this high. For the 9 round case the probability is higher by at least 7 bits.

The actual best linear characteristic is more likely in the 2^{90} range.

More analysis in this area is needed.

d) Reduced brute force work

It is possible to reduce the search work given a single ciphertext/plaintext pair. To perform this attack will require performing a large number of key generation operations. Each key generation operation produces a new key, stepping through the key space by taking the next key from the discarded key. These will tend to form cycles. This reduces the brute force search work factor by a factor of 10 or 12 depending on the number of rounds. The key generation is much quicker using this improved method.

This can be sped up even more. All keys forming a specific KA/KB pair are grouped. These groups can be formed in a simple manner. The remaining key bits are chosen in some order. Back stepping three times using the inverse of the key generation step allows finding the original key and stepping forward will yield the remaining key schedule. But the KA/KB can modify the s-box byteS. So the steps required are reduced to the fastest possible key generation and the fastest possible encryption.

e) Key security

The key transformation is highly non-linear however one complete round key is sufficient to recover all round keys for a 128 bit key size. The effort required to recover one complete round key is conjectured to be within a factor of 16 of enumerating all possible keys or requiring more than 2^{64} number of data pairs. Unrolling the round keys once one is recovered is trivial. For the 256 bit case two round keys are required that are adjacent. For separated round keys the attack is complex but still feasible. If 2 round keys separated by two other unknown round keys are attacked the effort approaches 128 bits quickly.

Methods requiring a number of key bits from each round are possible. These should be explored further since they are required for a successful differential or linear cryptanalysis.

Due to the nature of the key generation algorithm no keys have equivalent round keys.

Keys that generate KA or KB as zero may be considered weak. However only one in 2^{-31} keys will do this. No way is known to exploit this weakness. This condition is equivalent to choosing any particular value for these keys. If fact they may be guessed with an effort on the order of 2^{63}. This would seem to be required to attack the cipher. I do not know of any oracle that will allow such an attack.

The following observation on version 1 of this cipher:

> A class of very low entropy 256 bit keys will exist when the key contains all bytes in each word are identical after the constant C is XORed with the KH[0]. In this case all bytes in a round key word will be identical. There will be 2^{64} of these keys. This condition is easy to check for, simply XOR KH[0] by C then XOR all bytes in a word by the lowest byte.

This was corrected in this version. A round counter was also added to insure no sets of round keys contained more than two adjacent round keys that are identical.

f) Slide Attacks

Due to non-linear nature of the key generation algorithm, general slide attacks are unlikely to be successful.

The round function in this cipher does not fall under the category of 'weak' as defined by Biryukov and Wagner[BW99]. Further a slide attack suggests that the round transformation F[j]=F[j+1]. This is not the case if the key cannot be easily extracted from the basic round function.

Now slide attack can be applied to any variant with only keys KA and KB. This requires removing the round keys and pre and post whitening.

g) Related key attacks

Certain key pairs will have 'slide' key schedules. When the same ciphertext is encrypted with a pair of 'slide' keys the keys will differ by N key generation steps. These 'slide' keys are the outputs of each round of the key generation. However finding a 'slide' key for a given key and N is equivalent to finding the actual key. An attack of this nature is very difficult due to the variable KA and KB. However there will be 'slide' keys that will leave KA and KB the same. Encrypting the same text with a 'slide' key is dangerous. Doing so with a 'slide' key that leaves KA and KB unchanged is even more dangerous. An actual application of this weakness to an attack has not been found.

h) Davies-Murphy

Davies-Murphy attacks require multiple inputs to adjacent s-boxes. Generalizations of this attack relate to any s-box that is not bijective. All of the s-boxes in Q are bijective.

i) Boomerang attacks

The existence of 12 differentials suggests that an improvement with a boomerang might be possible. However due to all of the 12 differentials existing in a set of two planes (words 1 and 3 and two set of alternating bytes) with respect to the cipher, it will be impossible to match the differentials. The comments about differential attacks above hold for boomerang attacks as well.

j) Approximation attacks

Approximation of the byte substitution s-box and of the bit-slice s-boxes would have different requirements. The 4-bit s-boxes have order of 3 so it may be possible to eventually develop a low order attack but the 8-bit s-box has order 7. The total order of one round is no less than 41. The 8-bit s-box would require interpolation approximation while the 4-bit s-boxes do not have 'nice' GF(2) representations. Since these approximations do not form a group, these attacks are not applicable.

k) Impossible Differentials

Although the possibility of impossible differentials cannot be entirely discounted the actual use of such differentials seems unlikely. All possible differentials have uniformly low possibility therefore distinguishing between impossible and possible is difficult.

l) Other attacks

Partitioning cryptanalysis and statistical cryptanalysis do not seem to be better tools than those applied above. Extension to linear attack using low order approximation seems the most promising attack. More work is needed in that area.

8. Estimated Efficiency

The estimated efficiency in software is about 550 cycles on an AMD Athlon™ 650 with 128MB of RAM running under Windows 98SE. The round time measured at 67 cycles. This was in function call. The byte substitution with KA/KB addition is about 48 cycles. The total is finally reached of less than 584 cycles/block of about 36.5 cycles/byte. The inverse is the same speed. Due to the smaller amount of key material generated the keying is also fairly fast. This is fixed at about 50 cycles/subkey. This yields a total

keying time of about 500 cycles. The keying of the byte substitution with KA and KB takes about 400 cycles. This will reduce the round time by 8 XOR operations or about 8 cycles since they require memory lookups. That will reduce the total round time to less than 60 cycles. The code was not optimized for Athlon™ execution. The optimizations were for Pentium Pro™.

Efficient memory usage requires only a counter in addition to the key material for key setup. This is a single byte. The key setup requires additional memory to store KA and KB or speed will be impacted severely. They can however be calculated without additional memory overhead. Due to the reversible nature of the key schedule keys can be calculated in forward or reverse direction from a given round key.

For ciphering only one byte is needed in addition to the counter mentioned above. Plus possible an additional round counter for the cipher rounds. If ROM is limited this may be necessary. This allows the cipher to run with only 3 bytes of ram in addition to the round key and cipher text.

In the 128-bit version of keying the mixing of the upper 256 bytes of key material can be dropped. This is important in memory limited environments.

This cipher should be more efficient in hardware than Rijndael.

9. Acknowledgements

I would like to thank Dag Arne Osvik and Clyde Yokoi for their assistance in this endeavor. In closing I would like to state that I believe Q has great potential. Additional tuning may improve the speed even more.

10. References.

[ABK98] R. Anderson, E. Biham, L. Knudsen, Serpent: A proposal for the Advanced Encryption Standard, 1998, available online

[AC99] G. Ausiello, P Crescenzi, et al, Complexity and Approximation, Springer-Verlag, 1999

[BS93] E. Biham, A. Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993.

[BW99] A. Biryukov, D. Wagner, Slide Attacks, Fast Software Encryption '99, LNCS, pages 245-259, Springer-Verlag, 1999

[CDN98] G. Carter, E. Dawson, L. Nielsen, Information Security and Privacy, ACISP 98, LNCS 1587, pages 80-89, Springer-Verlag, 1998

[CT98] Z-G Chen and S. Tavares, Towards Provable Security of Substitution-Permutation Encryption Networks, Selected Areas in Cryptography, LNCS 1556, pages 43-56, Springer-Verlag 1999

[DR98] J. Daemen, V. Rijmen, AES Proposal: Rijndael, 1998, available online

[HA98] S. Harris, C. Adams, Key Dependent S-Box Manipulations, Selected Areas in Cryptography 98, LNCS, Springer-Verlag, 1998

[IBM98] IBM, 11 Authors, MARS – a candidate cipher for AES,1998, available online

[KF99] C. Karr, L.M. Freeman, Industrial Applications of Genetic Algorithms, CRC Press, 1999

[KMO97] On Strict Estimation Method of Provable Security against Differential and Linear Cryptanalysis, ICICS 97, LNCS 1587, pages 258-268, Springer-Verlag, 1997

[KMT99] L. Keliher, H. Meijer, S. Tavares, Modeling Linear Characteristics of Substitution-Permutation Networks, Selected Areas in Cryptography 99,LNCS 1758,pp 78-91, Springer-Verlag, 2000

[KR69] J. Kilian, P Rogaway, How to Protect DES Against Exhaustive Key Search, Crypto 96, LNCS, pages 252-267, Springer-Verlag 1996

[LCH99] C.H. Lin, Specification and Analysis of CRYTPON Version 1.0, 1999, available online at

[LRK99] L.R. Knudsen, Contemorary Block Ciphers, Lectures on Data Security, LNCS 1561, pages 105-126,Springer-Verlag, 1999

[MBCCD99] W. Millan, L. Burnett, G. Carter, A. Clark, E. Dawson, Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes, ICICS 99, LNCS, pages 263-274, Springer-Verlag,

1999

[MCD97] W. Millan, A. Clark, E. Dawson, An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions, ICICS 97, LNCS, pages 149-158, Springer-Verlag, 1997

[MCD98] W. Millan, A. Clark, E. Dawson, Heuristic Design of Cryptographically Strong Balanced Boolean Functions, Eurocrypt 98, LNCS, pages 489-499, Springer-Verlag, 1998

[MCD99] W. Millan, A. Clark, E. Dawson, Boolean Function Design Using Hill Climbing Methods, ACISP 99, LNCS, pages 1-11, Springer-Verlag, 1999

[MS77] F.J. MacWilliams, N.J.A. Sloane, The Theory of Error Correcting Codes, North-Holland, 1977

[MZ92] Z. Michalewicz, Genetic Algorithms+Data Structures=Evolutionare Programs, Springer-Verlag, 1992

[ODA00] Dag Arne Osvik, Speeding Up Serpent, 2000, available online

[OL95] Luke O'Conner, Properties of Linear Approximation Tables, Fast Software Encryption 95, LNCS, pages 131-136, Springer-Verlag, 1995

[RRSY98] R. Rivest, M.J.B. Robshaw, R. Sidney, Y.L. Yin, The RC6™ Block Cipher, 1998, available online

[SKWHF98] Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, Twofish: A 128-Bit Bock Cipher, 1998, available online

[SS98] S. Skiena, The Algorithm Design Manual, Springer-Verlag, 1998

[YJH91] Jun-Hui Yand, The Data Base of Selected Permutations, Asiacrypt 91, LNCS, pages 73-81, Springer-Verlag

Appendix A

```
unsigned byteS[256] = {
 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171, 118,
202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113, 216,  49,  21,
  4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226, 235,  39, 178, 117,
  9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214, 179,  41, 227,  47, 132,
 83, 209,   0, 237,  32, 252, 177,  91, 106, 203, 190,  57,  74,  76,  88, 207,
208, 239, 170, 251,  67,  77,  51, 133,  69, 249,   2, 127,  80,  60, 159, 168,
 81, 163,  64, 143, 146, 157,  56, 245, 188, 182, 218,  33,  16, 255, 243, 210,
205,  12,  19, 236,  95, 151,  68,  23, 196, 167, 126,  61, 100,  93,  25, 115,
 96, 129,  79, 220,  34,  42, 144, 136,  70, 238, 184,  20, 222,  94,  11, 219,
224,  50,  58,  10,  73,   6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121,
231, 200,  55, 109, 141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8,
186, 120,  37,  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138,
112,  62, 181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158,
225, 248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,  22,
};

unsigned invbyteS[256] = {
 82,   9, 106, 213,  48,  54, 165,  56, 191,  64, 163, 158, 129, 243, 215, 251,
124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,  68, 196, 222, 233, 203,
 84, 123, 148,  50, 166, 194,  35,  61, 238,  76, 149,  11,  66, 250, 195,  78,
  8,  46, 161, 102,  40, 217,  36, 178, 118,  91, 162,  73, 109, 139, 209,  37,
114, 248, 246, 100, 134, 104, 152,  22, 212, 164,  92, 204,  93, 101, 182, 146,
108, 112,  72,  80, 253, 237, 185, 218,  94,  21,  70,  87, 167, 141, 157, 132,
144, 216, 171,   0, 140, 188, 211,  10, 247, 228,  88,   5, 184, 179,  69,   6,
208,  44,  30, 143, 202,  63,  15,   2, 193, 175, 189,   3,   1,  19, 138, 107,
 58, 145,  17,  65,  79, 103, 220, 234, 151, 242, 207, 206, 240, 180, 230, 115,
150, 172, 116,  34, 231, 173,  53, 133, 226, 249,  55, 232,  28, 117, 223, 110,
 71, 241,  26, 113,  29,  41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27,
252,  86,  62,  75, 198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,
 31, 221, 168,  51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,  95,
 96,  81, 127, 169,  25, 181,  74,  13,  45, 229, 122, 159, 147, 201, 156, 239,
160, 224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,  83, 153,  97,
 23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,  85,  33,  12, 125,
};

unsigned sA[16]={0,13,6,8,11,7,1,14,9,10,5,15,2,4,12,3};
unsigned invsA[16]={0,6,12,15,13,10,2,5,3,8,9,4,14,1,7,11};
unsigned sB[16]={0,15,11,8,12,9,6,3,13,1,2,4,10,7,5,14};
unsigned invsB[16]={0,9,10,7,11,14,6,13,3,5,12,2,4,8,15,1};
unsigned sC[16]={0,9,10,4,11,7,12,1,13,6,3,15,14,8,5,2};
unsinged invsC[16]={0,7,15,10,3,14,9,5,13,1,2,4,6,8,12,11};
```

the constant C in hexadecimal with the leading 1 and a large number of digits from UBASIC.
1.9E3779B9 7F4A7C15 F39CC060 5CEDC834 1082276B F3A27251 F86C6A11 D0C18E95 2767F0B1 53D27B7F 0347045B 5BF1827F 01886F09 28403002 C1D64BA4 0F335E36 F06AD7AE 9717877E 85839D6E FFBD7DC6 64D325D1 C5371682 CADD0CCC FDFFBBE1
Note that the last few digits probably contain some error.

Appendix B

Note that the sequence given for the inverse of s-box A is in permuted form and that these operations give the outputs in a permuted form. Care must be taken that these are properly aligned See appendix see for actual code. The sequences are given in Intel™ instruction format.

Parameters 1 4 3 20000 1 29 216750081
Programs 15 7 11 14

found 7 r 16 o 2 n 33334
in 32894 in 33462 in 14521 in 29171 in 33465 in 32897 in 10877 in 32263 in 28748 in 25143 in 19612 in 22900 in 27613 in 23095 in 18516 in 23838 in 30689 in 29253 in 12778 in 29857 in 33025 in 32881 in 10646 in 32309
p 33334 p 32759 p 33313 p 17437 p 30051 p 30632 p 29467 p 18154 p 34254 p 32748 p 34250 p 11776 p 33680 p 30159 p 33676 p 11144 p 24769 p 26769 p 24628 p 11173 p 24441 p 27146 p 23392 p 12889
0 2 0001111011100001 0001100010000000
3 3 0110011000101101 0110000001011000
1 3 0010110010110110 0011100111100010
2 3 0100100101100111 0101101100111010
s 0 10 6 1 13 7 3 8 5 9 15 4 2 14 12 11
i 0 3 12 6 11 8 2 5 7 9 1 15 14 4 13 10
mov 4,0 and 0,1
xor 0,2 mov 2,4
xor 4,0 xor 2,1
and 1,4 xor 0,3
xor 1,3 and 3,4
xor 3,2 mov 2,1
or 1,3 and 2,3
xor 1,4 xor 2,4

found 20 r 16 o 2 n 23838
in 34254 t 32 in 34250 t 32 in 32748 t 32 in 11776 t 32 in 33680 t 32 in 33676 t 32 in 30159 t 32 in 11144 t 32 in 33334 t 32 in 33313 t 32 in 32759 t 32 in 17437 t 32 in 30051 t 32 in 29467 t 32 in 30632 t 32 in 18154 t 32 in 24441 t 32 in 23392 t 32 in 27146 t 32 in 12889 t 32 in 24769 t 32 in 24628 t 32 in 26769 t 32 in 11173 t 32
p 23838 p 12778 p 27613 p 25143 p 22900 p 10646 p 28748 p 23095 p 29857 p 18516 p 30689 p 33462 p 32309 p 19612 p 33025 p 32897 p 32263 p 14521 p 33465 p 32881 p 29171 p 10877 p 32894 p 29253
1 2 0100111010110001 0101110010000000
4 3 0010110101011001 0011100001110010
0 3 0110010100110110 0110001001000110
2 3 0101100111010010 0100101011110010
s 0 13 6 8 11 7 1 14 9 10 5 15 2 4 12 3
i 0 6 12 15 13 10 2 5 3 8 9 4 14 1 7 11
mov 4,0 xor 0,1
xor 0,2 xor 1,3
or 4,0 and 2,0
xor 1,4 xor 4,2
mov 2,4 and 4,1
xor 4,0 xor 0,3
and 0,1 xor 3,4
xor 0,3 xor 2,3

found 8 r 16 o 2 n 17887
in 31834 in 31833 in 31621 in 18791 in 31046 in 31045 in 29776 in 20439 in 31755 in 31747 in 31633 in 13643 in 30502 in 30474 in 29802 in 13648 in 27466 in 27437 in 24180 in 18810 in 28140 in 28132 in 24175 in 20449

p 17887 p 11215 p 24652 p 7790 p 15351 p 12919 p 24316 p 7783 p 12167 p 11218 p 23488 p 9972 p 8627 p 12920 p 21752 p 8388 p 2847 p 7862 p 21636 p 2723 p 5327 p 7859 p 22568 p 4043
3 2 0100110110110010 0101111010000000
1 3 0010110001111001 0011100101000000
0 3 0001011011011010 0001011110100000
2 3 0110101010011100 0110101010001110
s 0 9 10 4 11 7 12 1 13 6 3 15 14 8 5 2
i 0 7 15 10 3 14 9 5 13 1 2 4 6 8 12 11
xor 0,1 mov 4,2
xor 2,3 xor 3,1
or 2,0 and 3,0
xor 3,2 xor 2,1
or 1,3 xor 4,2
xor 0,2 xor 1,4
or 4,0 xor 2,3
xor 0,1 xor 2,4

the following search produced a sequence for Sbox inverse C

```
Parameters 1  4  4  20000  6  90  245628934
Programs  17  8  10  16
Starting 0
Starting 1
found 1 r 18 o 2 n 23
 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36
in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36
in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36 in 0 t 36
in 0 t 36 in 0 t 36 in 0 t 36
 p 23 p 22 p 18 p 2 p 17 p 16 p 12 p 4 p 21 p 20 p 19 p 0 p 15 p 14 p
13 p 1 p 9 p 8 p 7 p 3 p 11 p 10 p 6 p 5
3  2  0110011010010110   0110000010001000
0  3  0100001110101101   0101011110010000
1  3  0010111011100001   0011101010100010
2  3  0111100010001011   0111100010001010
s 0  11  13  8  12  5  7  2  15  4  6  1  10  3  9  14
i 0  11  7  13  9  5  10  6  3  14  12  1  4  2  15  8
xor 0,1 mov 4,1
xor 1,2 xor 2,3
or 1,2  and 3,2
xor 1,3 xor 4,2
xor 3,0 or 0,1
xor 0,4 xor 1,3
xor 1,2 xor 2,0
and 2,1 xor 1,4
xor 1,2 xor 2,4
```

The sequences for S-box three are listed in Dag Arne Osvik's paper. And in code in Appendix C.

Appendix C

The following code fragments define Sboxes A,B, and C and their inverses. r0 to r4 may be any unused registers. These sequences are from code assembled with NASMW.

Sbox A

```
                mov esi,[esp+plaintext+frame]
                mov r0,[esi+0]
                mov r1,[esi+4]
                mov r2,[esi+8]
                mov r3,[esi+12]

                ;start inverse s-box A
                mov r4,r0               ;mov 4,0
                xor r0,r1               ;xor 0,1
                ;should synchronize for pentium here
                xor r0,r2               ;xor 0,2
                xor r1,r3               ;xor 1,3
                or r4,r0                ;or 4,0
                and r2,r0               ;and 2,0
                xor r1,r4               ;xor 1,4
                xor r4,r2               ;xor 4,2
                mov r2,r4               ;mov 2,4
                and r4,r1               ;and 4,1
                xor r4,r0               ;xor 4,0
                xor r0,r3               ;xor 0,3
                and r0,r1               ;and 0,1
                xor r3,r4               ;xor 3,4
                xor r0,r3               ;xor 0,3
                xor r2,r3               ;xor 2,3

                mov esi,[esp+ciphertext+frame]
                mov [esi+0],r1
                mov [esi+4],r4
                mov [esi+8],r0
                mov [esi+12],r2
```

```
Sbox inverse A
                mov esi,[esp+plaintext+frame]
                mov r1,[esi+0]
                mov r3,[esi+4]
                mov r2,[esi+8]
                mov r0,[esi+12]

                ;start inverse sboxa
                mov r4,r0     ;mov 4,0
                and r0,r1     ;and 0,1
                ;should synchronize for pentium here
                xor r0,r2     ;xor 0,2
                mov r2,r4     ;mov 2,4
                xor r4,r0     ;xor 4,0
                xor r2,r1     ;xor 2,1
                and r1,r4     ;and 1,4
                xor r0,r3     ;xor 0,3
                xor r1,r3     ;xor 1,3
                and r3,r4     ;and 3,4
                xor r3,r2     ;xor 3,2
                mov r2,r1     ;mov 2,1
                or r1,r3      ;or 1,3
                and r2,r3     ;and 2,3
                xor r1,r4     ;xor 1,4
                xor r2,r4     ;xor 2,4

                mov esi,[esp+ciphertext+frame]
                mov [esi+0],r2
                mov [esi+4],r3
                mov [esi+8],r1
                mov [esi+12],r0
```

Sbox B

```
mov esi,[esp+plaintext+frame]
mov r0,[esi+0]
mov r1,[esi+4]
mov r2,[esi+8]
mov r3,[esi+12]

;start s-box b
mov r4,r0       ;r4 = r0
or r0,r3        ;r0|= r3
;should synchronize for pentium here
xor r3,r1       ;r3^= r1
and r1,r4       ;r1&= r4
xor r4,r2       ;r4^= r2
xor r2,r3       ;r2^= r3
and r3,r0       ;r3&= r0
or r4,r1        ;r4|= r1
xor r3,r4       ;r3^= r4
xor r0,r1       ;r0^= r1
and r4,r0       ;r4&= r0
xor r1,r3       ;r1^= r3
xor r4,r2       ;r4^= r2
or r1,r0        ;r1|= r0
xor r1,r2       ;r1^= r2
xor r0,r3       ;r0^= r3
mov r2,r1       ;r2 = r1
or r1,r3        ;r1|= r3
xor r1,r0       ;r1^= r0

mov esi,[esp+ciphertext+frame]
mov [esi+0],r1
mov [esi+4],r2
mov [esi+8],r3
mov [esi+12],r4
```

```
Sbox inverse B
                mov esi,[esp+plaintext+frame]
                mov r0,[esi+0]
                mov r1,[esi+4]
                mov r2,[esi+8]
                mov r3,[esi+12]

                ;start inverse s-box b
                mov r4,r2       ;r4 = r2
                xor r2,r1       ;r2^= r1
                ;should synchronize for pentium here
                xor r0,r2       ;r0^= r2
                and r4,r2       ;r4&= r2
                xor r4,r0       ;r4^= r0
                and r0,r1       ;r0&= r1
                xor r1,r3       ;r1^= r3
                or r3,r4        ;r3|= r4
                xor r2,r3       ;r2^= r3
                xor r0,r3       ;r0^= r3
                xor r1,r4       ;r1^= r4
                and r3,r2       ;r3&= r2
                xor r3,r1       ;r3^= r1
                xor r1,r0       ;r1^= r0
                or r1,r2        ;r1|= r2
                xor r0,r3       ;r0^= r3
                xor r1,r4       ;r1^= r4
                xor r0,r1       ;r0|= r1

                mov esi,[esp+ciphertext+frame]
                mov [esi+0],r2
                mov [esi+4],r1
                mov [esi+8],r3
                mov [esi+12],r0
```

Sbox C

```
                mov esi,[esp+plaintext+frame]
                mov r0,[esi+0]
                mov r1,[esi+4]
                mov r2,[esi+8]
                mov r3,[esi+12]

                ;start inverse s-box b
                xor r0,r1       ;xor 0,1
                mov r4,r2       ;mov 4,2
                ;should synchronize for pentium here
                xor r2,r3       ;xor 2,3
                xor r3,r1       ;xor 3,1
                or r2,r0        ;or 2,0
                and r3,r0       ;and 3,0
                xor r3,r2       ;xor 3,2
                xor r2,r1       ;xor 2,1
                or r1,r3        ;or 1,3
                xor r4,r2       ;xor 4,2
                xor r0,r2       ;xor 0,2
                xor r1,r4       ;xor 1,4
                or r4,r0        ;or 4,0
                xor r2,r3       ;xor 2,3
                xor r0,r1       ;xor 0,1
                xor r2,r4       ;xor 2,4

                mov esi,[esp+ciphertext+frame]
                mov [esi+0],r3
                mov [esi+4],r1
                mov [esi+8],r0
                mov [esi+12],r2
```