M2 L3 , Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. The AES cipher (& other candidates) form the latest generation of block ciphers, and now we see a significant increase in the block size - from the old standard of 64-bits up to 128-bits; and keys from 128 to 256-bits. In part this has been driven by the public demonstrations of exhaustive key searches of DES. Whilst triple-DES is regarded as secure and well understood, it is slow, especially in s/w. In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr.Vincent Rijmen.

NIST's requirements for the AES candidate submissions. These criteria span the range of concerns for the practical application of modern symmetric block ciphers.

In fact, two set of criteria evolved. When NIST issued its original request for candidate algorithm nominations in 1997, the request stated that candidate algorithms would be compared based on the factors shown in Stallings Table5.1, which were used to evaluate field of 15 candidates to select shortlist of 5. These had categories of security, cost, and algorithm & implementation characteristics.

The final criteria evolved during the evaluation process, and were used to select Rijndael from that short-list, and more details are given in Stallings Table 5.2, with categories of: general security, ease of software & hardware implementation, implementation attacks, & flexibility (in en/decrypt, keying, other factors).

The AES shortlist of 5 ciphers was as shown. Note mix of commercial (MARS, RC6, Twofish) verses academic (Rijndael, Serpent) proposals, sourced from various countries.

All were thought to be good – it came down to the best balance of attributes to meet criteria, in particular the balance between speed, security & flexibility.

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128,192,or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. Rijndael is an academic submission, based on the earlier Square cipher, from Belgium academics Dr Joan Daemen and Dr Vincent Rijmen. It is an iterative cipher (operates on entire data block in every round) rather than feistel (operate on halves at a time), and was designed to have characteristics of: Resistance against all known attacks, Speed and code compactness on a wide range of platforms, & Design simplicity.

The input to the AES encryption and decryption algorithms is a single 128-bit block, depicted in FIPS PUB 197, as a square matrix of bytes .This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output.

The key is expanded into 44/52/60 lots of 32-bit words (see later), with 4 used in each round.

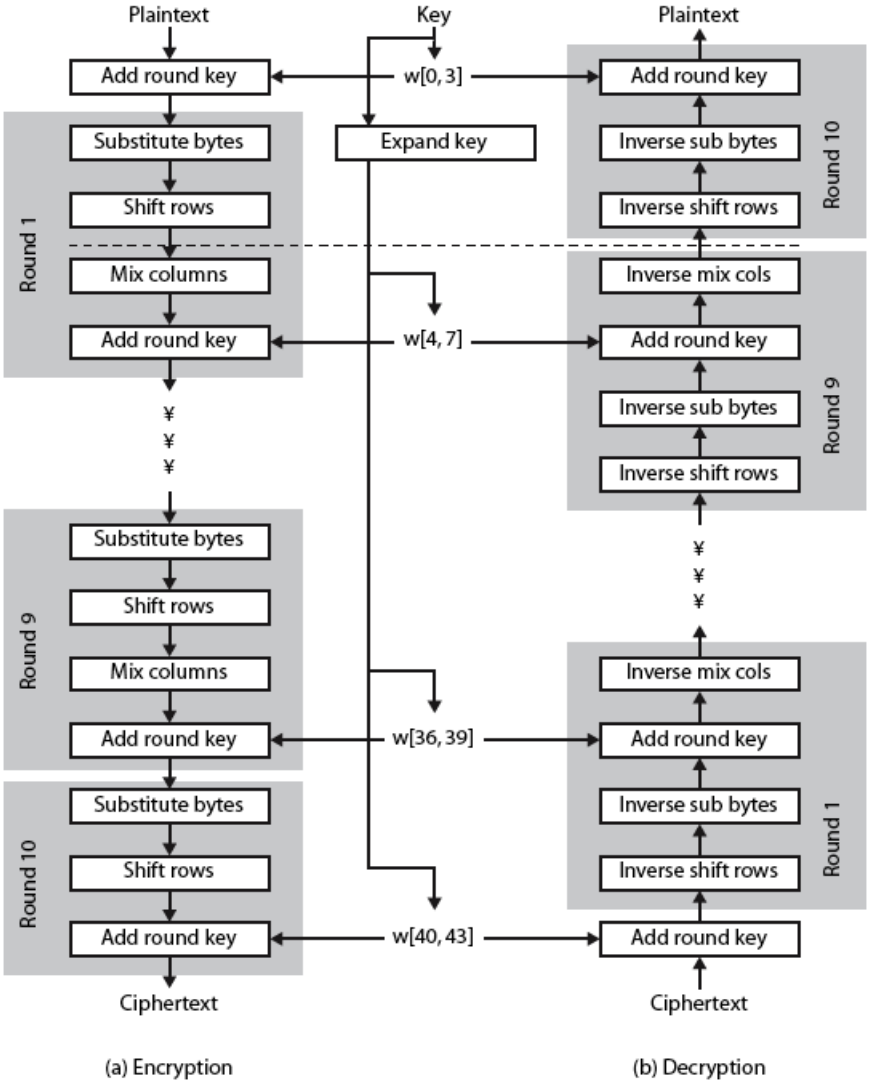
The data computation then consists of an "add round key" step, then 9/11/13 rounds with all 4 steps, and a final $10^{th}/12^{th}/14^{th}$ step of byte subs + mix cols + add round key. This can be viewed as alternating XOR key & scramble data bytes operations. All of the steps are easily reversed, and can be efficiently implemented using XOR's & table lookups.

The input to the AES encryption and decryption algorithms is a single 128-bit block, depicted in FIPS PUB 197, as a square matrix of bytes .This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output.

The key is expanded into 44/52/60 lots of 32-bit words (see later), with 4 used in each round.

The data computation then consists of an "add round key" step, then 9/11/13 rounds with all 4 steps, and a final $10^{th}/12^{th}/14^{th}$ step of byte subs + mix cols + add round key. This can be viewed as alternating XOR key & scramble data bytes operations. All of the steps are easily reversed, and can be efficiently implemented using XOR's & table lookups.

Figure below shows the overall structure of AES, as detailed on the previous slide.

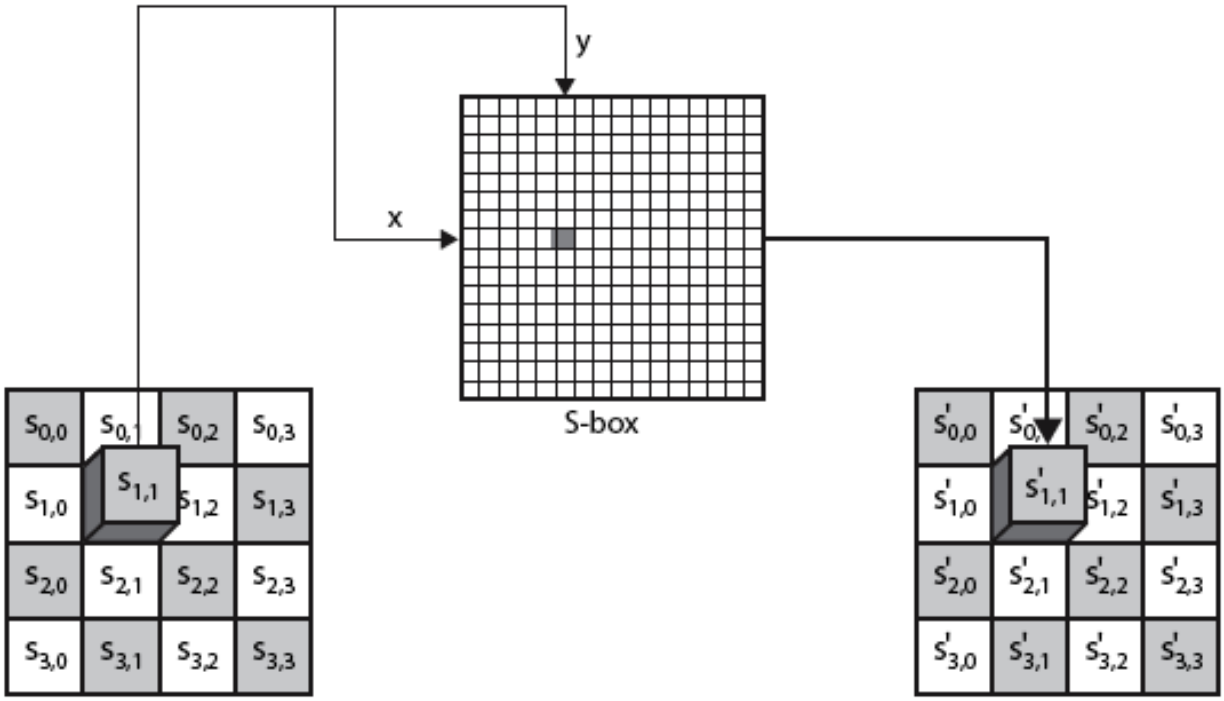


Now discuss each of the four stages used in AES. The Substitute bytes stage uses an S-box to perform a byte-by-byte substitution of the block. There is a single 8-bit wide S-box used on every byte. This S-box is a permutation of all 256 8-bit values, constructed using a transformation which treats the values as polynomials in $GF(2^8)$ – however it is fixed, so really only need to know the table when implementing. Decryption requires the inverse of the table. These tables are given in Stallings Table 4.5.

The table was designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, with the property that the

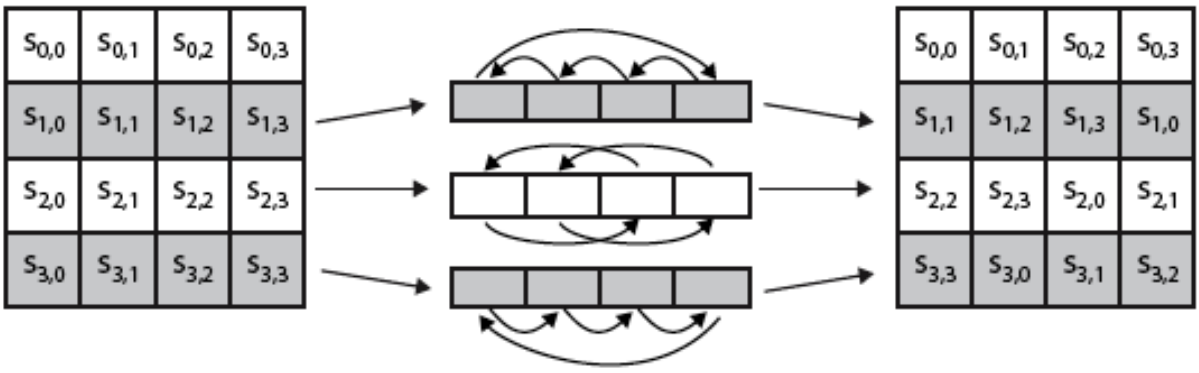
output cannot be described as a simple mathematical function of the input, with no fixed points and no "opposite fixed points".

As this diagram Fig below shows, the Byte Substitution operates on each byte of state independently, with the input byte used to index a row/col in the table to retrieve the substituted value.



The ShiftRows stage provides a simple "permutation" of the data, whereas the other steps involve substitutions. Further, since the state is treated as a block of columns, it is this step which provides for diffusion of values between columns. It performs a circular rotate on each row of 0, 1, 2 & 3 places for respective rows. When decrypting it performs the circular shifts in the opposite direction for each row. This row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes, and ensures that the 4 bytes of one column are spread out to four different columns.

Figure Below illustrates the Shift Rows permutation.

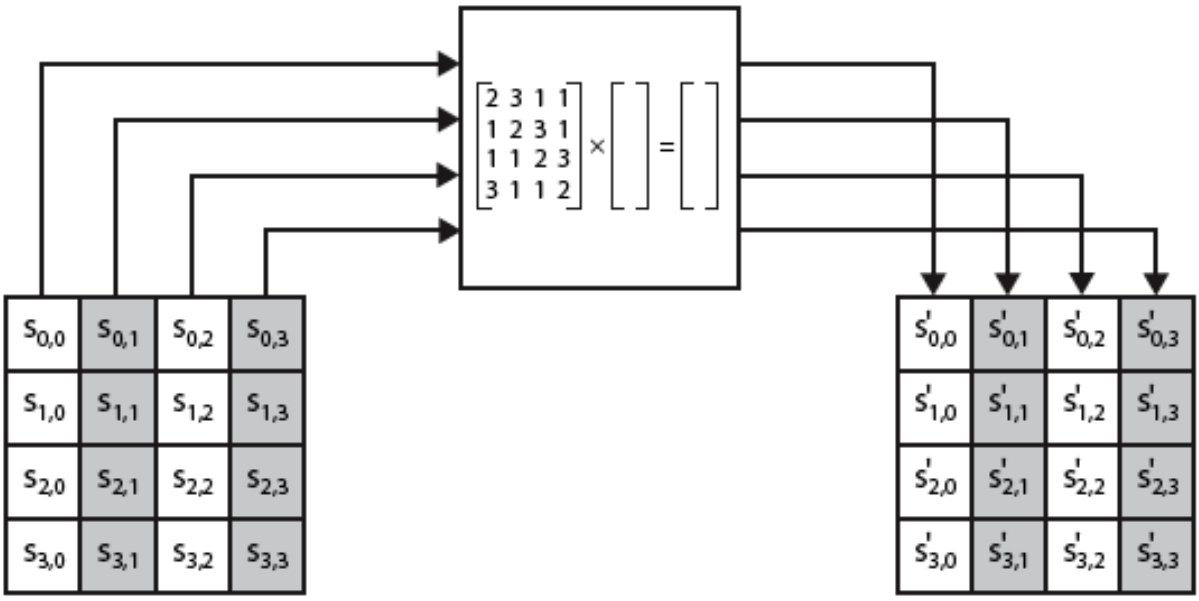


The MixColumns stage is a substitution that makes use of arithmetic over GF(2^8). Each byte of a column is mapped into a new value that is a function of all four bytes in that column. It is designed as a matrix

multiplication where each byte is treated as a polynomial in GF(2^8). The inverse used for decryption involves a different set of constants.

The constants used are based on a linear code with maximal distance between code words – this gives good mixing of the bytes within each column. Combined with the "shift rows" step provides good avalanche, so that within a few rounds, all output bits depend on all input bits.

Figure below illustrates the Mix Columns transformation.



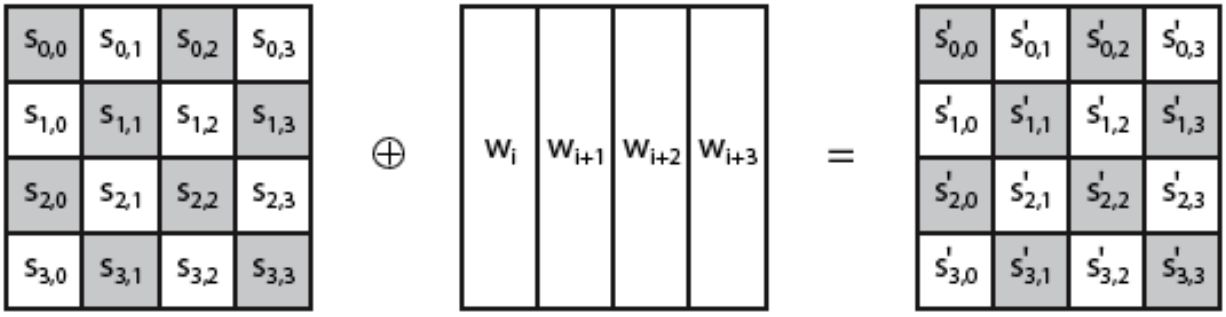
In practise, you implement Mix Columns by expressing the transformation on each column as 4 equations (Stallings equation 5.4) to compute the new bytes for that column. This computation only involves shifts, XORs & conditional XORs (for the modulo reduction).

The decryption computation requires the use of the inverse of the matrix, which has larger coefficients, and is thus potentially a little harder & slower to implement.

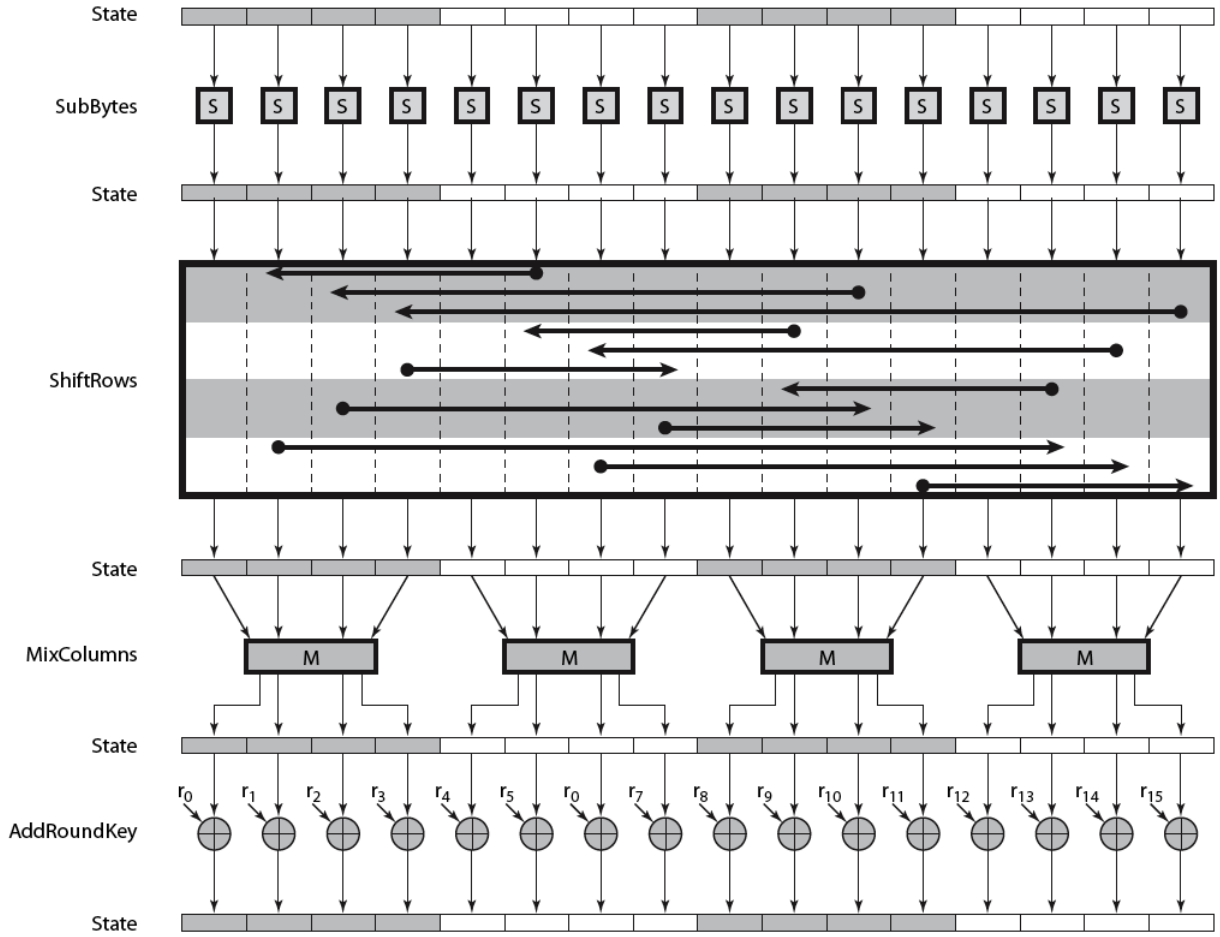
The designers & the AES standard provide an alternate characterisation of Mix Columns, which treats each column of State to be a four-term polynomial with coefficients in GF(2^8). Each column is multiplied by a fixed polynomial a(x) given in Stallings eqn 5.7. Whilst this is useful for analysis of the stage, the matrix description is all that's required for implementation.

Lastly is the Add Round Key stage which is a simple bitwise XOR of the current block with a portion of the expanded key. Note this is the only step which makes use of the key and obscures the result, hence MUST be used at start and end of each round, since otherwise could undo effect of other steps. But the other steps provide confusion/diffusion/non-linearity. That us you can look at the cipher as a series of XOR with key then scramble/permute block repeated. This is efficient and highly secure it is believed.

Figure illustrates the Add Round Key stage, which like Byte Substitution, operates on each byte of state independently.

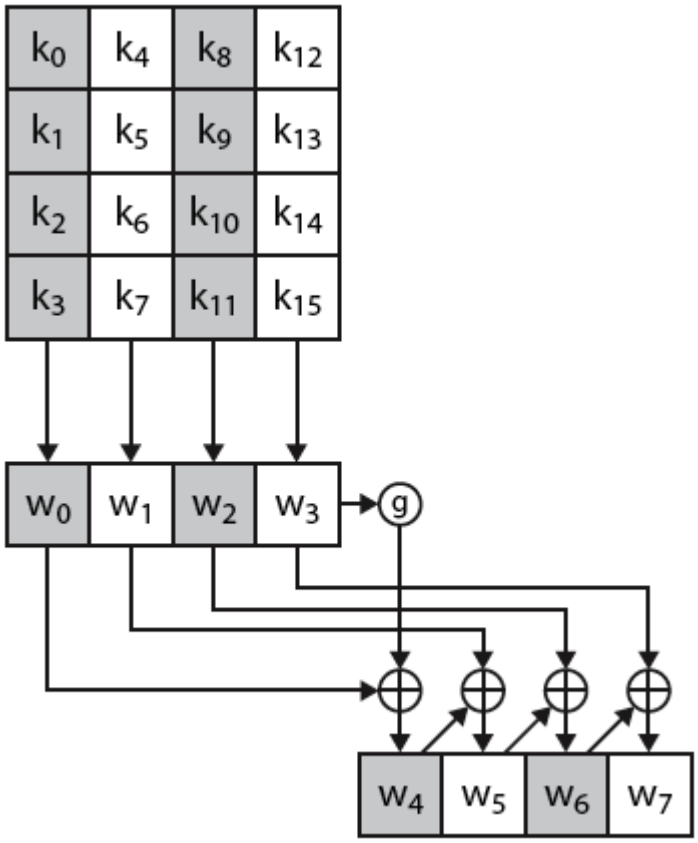


Can thus now view all the internal details of the AES round, showing how each byte of the state is manipulated, as shown in Stallings Figure 5.3.



The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of words, providing a 4-word round key for the initial AddRoundKey stage and each of the 10/12/14 rounds of the cipher. It involves copying the key into the first group of 4 words, and then constructing subsequent groups of 4 based on the values of the previous & 4th back words. The first word in each group of 4 gets "special treatment" with rotate + S-box + XOR constant on the previous word before XOR'ing the one from 4 back. In the 256-bit key/14 round version, there's also an extra step on the middle word.

The first block of the AES Key Expansion is shown here in Stallings Figure 5.6. It shows each group of 4 bytes in the key being assigned to the first 4 words, then the calculation of the next 4 words based on the values of the previous 4 words, which is repeated enough times to create all the necessary subkey information.

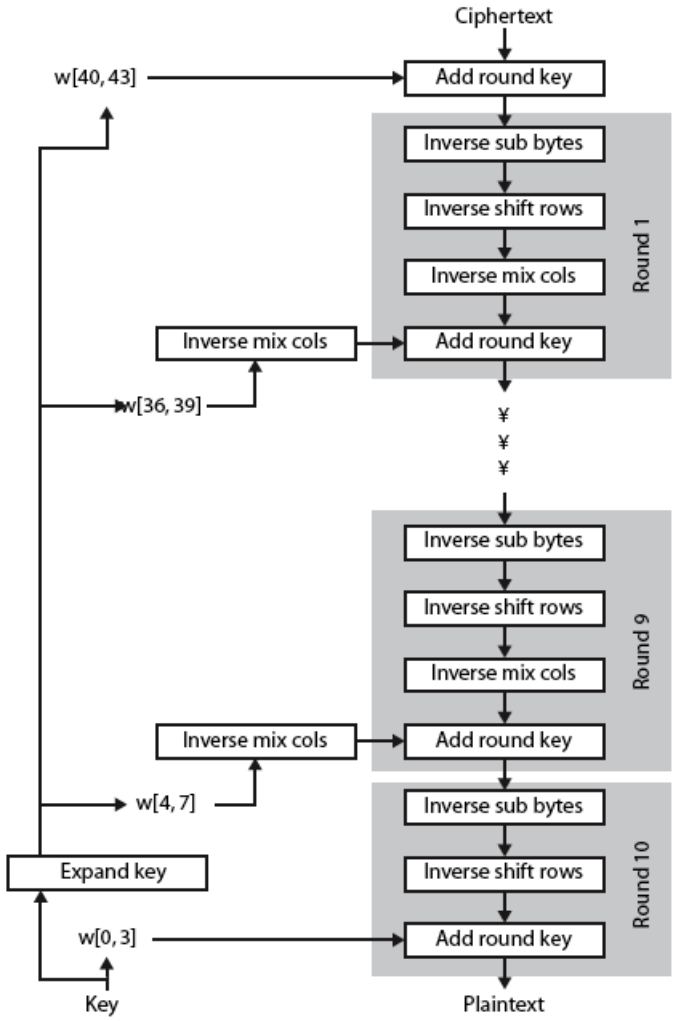


The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. It is designed to be simple to implement, but by using round constants break symmetries, and make it much harder to deduce other key bits if just some are known (but once have as many consecutive bits as are in key, can then easily recreate the full expansion). The design criteria used are listed above.

The AES decryption cipher is not identical to the encryption cipher (Stallings Figure 5.1). The sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm, with the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

By constructing an equivalent inverse cipher with steps in same order as for encryption, we can derive a more efficient implementation. Clearly swapping the byte substitutions and shift rows has no effect, since work just on bytes. Swapping the mix columns and add round key steps requires the inverse mix columns step be applied to the round keys first – this makes the decryption key schedule a little more complex with this construction, but allows the use of same h/w or s/w for the data en/decrypt computation.

Illustrate the equivalent inverse cipher with Stallings Figure 5.7.



AES can be implemented very efficiently on an 8-bit processor.

AddRoundKey is a bytewise XOR operation.

ShiftRows is a simple byte shifting operation.

SubBytes operates at the byte level and only requires a lookup of a 256 byte table S.

MixColumns (matrix multiply) can be implemented as byte XOR's & table lookups with a 2nd 256 byte table X2, using the formulae shown in Stallings equation 5.9.

AES can also be very efficiently implemented on an 32-bit processor, by rewriting the stage transformation to use 4 table lookups & 4 XOR's per column of state. These tables can be computed in advance using the formulae shown in the text, and need 4Kb to store.

The developers of Rijndael believe that this compact, efficient implementation was probably one of the most important factors in the selection of Rijndael for AES.