AES Encryption and CAST's AES IP Cores

Meredith Lucky, VP Sales, CAST, Inc.
December, 2008

Introduction

Governments, companies, and even individuals worldwide rely on encryption to protect confidential data for transmission, processing, and storage.

The ideal encryption algorithm would provide unbreakable security with no significant impact on the normal flow of data. For twenty years the Data Encryption Standard (DES) was the state of the art, but it became unsafe once it was determined how to identify its secret key and decipher the encrypted data. A variant, Triple DES, is more secure, but requires three times the processing time to encrypt or decrypt data.

In September 1997, the National Institute of Standards and Technology (NIST) issued a request for a new Advanced Encryption Standard (AES) to replace DES. Several algorithms were proposed and considered, and in October 2000 the Rijndael block cipher algorithm was chosen as the new AES.

NIST mandated AES for civilian agency use in November 2001 as Federal Information Processing Standard Publication 197 (FIPS 197). In 2003 AES was approved for use with classified information by the US National Security Agency (NSA). Today AES is the encryption standard used around the world.

This paper gives an introduction to the AES algorithm, explaining its encryption and decryption processes and describing its various options. The paper then discusses how AES is implemented in CAST's IP cores. These register transfer level (RTL) cores can be used to physically implement the AES algorithm for fast hardware encryption and decryption in ASIC or FPGA technologies.

The AES Algorithm

AES is a block cipher, meaning that it operates on an *input block* of data of a known size and outputs a block of data which is the same size. An *input key* is also required as input to the AES algorithm. A *mode of operation* is selected which selects a specific implementation of the AES algorithm.

The input block and output block data are each a fixed length size of 128 bits. The unencrypted data is referred to as *Plaintext*, and the encrypted data is referred to as *CipherText*.

The input key can be 128, 192 or 256 bits. The same key is used for both encryption and decryption. In general, the longer the key, the higher the security level obtained with the encryption.

AES Processing Steps

The AES algorithm consists of a series of steps. With the exception of the initial key expansion operation, all the steps are repeated a number of times; each such repetition of steps is called a *round*.

The number of rounds used in the algorithm is dependent on the size of the input key. In Figure 1, this is represented by variable n, and n=10 for a 128-bit key, 12 for a 192-bit key, and 14 for a 256-bit key.

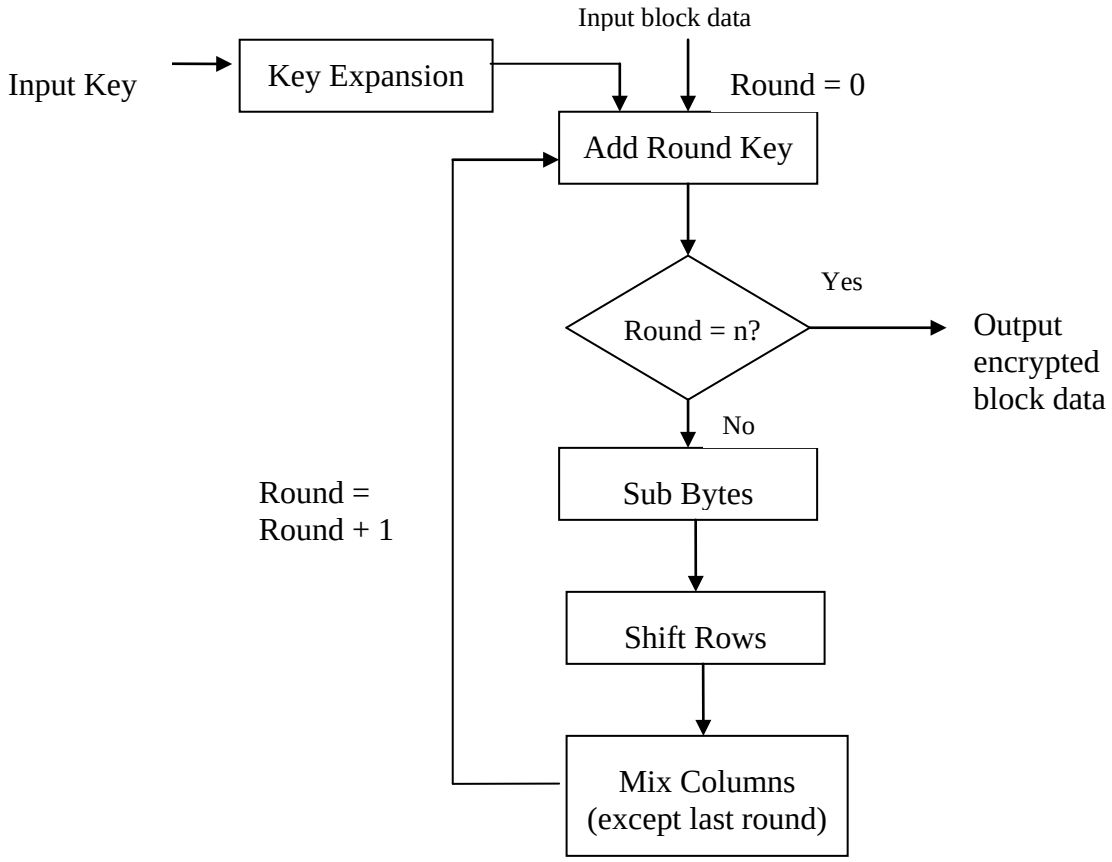

Figure 1. AES Encryption Flow and Processing Steps

The initial input plaintext data is used to create a matrix of 4x4 bytes of data. The transformed data of this matrix is stored as *state data* during processing of each step (see Figure 2).

0,0	0,1	0,2	0,3
1,0	1,2	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Figure 2. State Data

In decryption, the reverse processing steps are used to transform encrypted ciphertext to unencrypted plaintext data. This decryption process uses the same key as the encryption process.

The encryption steps are briefly described below.

The **Key Expansion** step expands and transforms a 128-, 192- or 256-bit key to 11, 13, or 15 sub-keys, each 128-bits long, using the Rijndael key expansion algorithm. One sub-key corresponds to each AES processing round, thus each sub-key is referred to as a *round key*. The set of 11, 13, or 15 round keys comprises the *key schedule*.

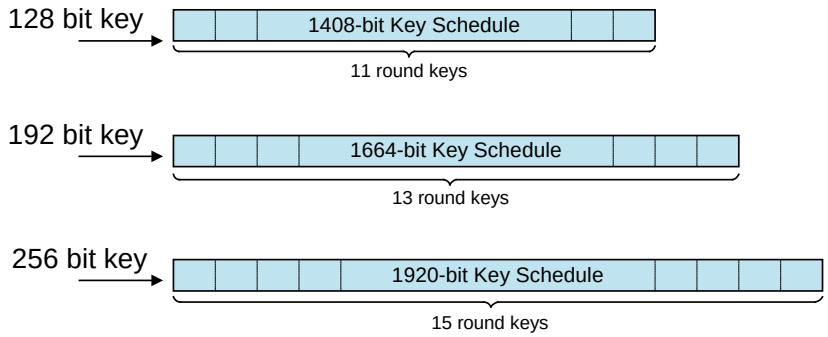

Figure 3. Encryption Key lengths and resulting Round Keys

The **Add Round Key** step is a transformation that combines the current state data block and the round key corresponding to the specific round using an XOR function.

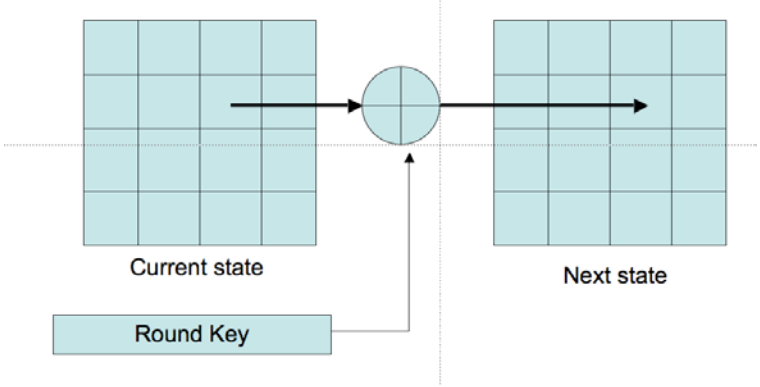

Figure 4. AES Steps: Add Round Key

The **Sub Bytes** step replaces each state data byte with an entry in a fixed lookup table.

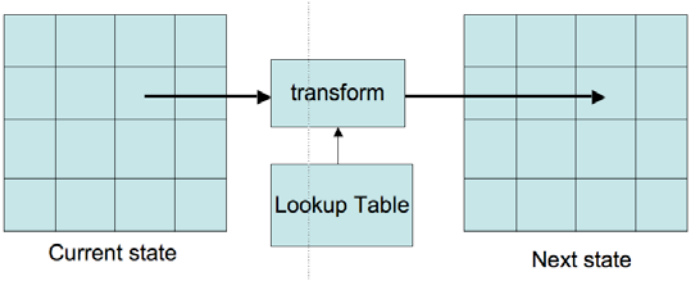

Figure 5. AES Steps: Sub-Bytes

The **Shift Rows** step rotates the four bytes of state data in each row in the state data matrix.

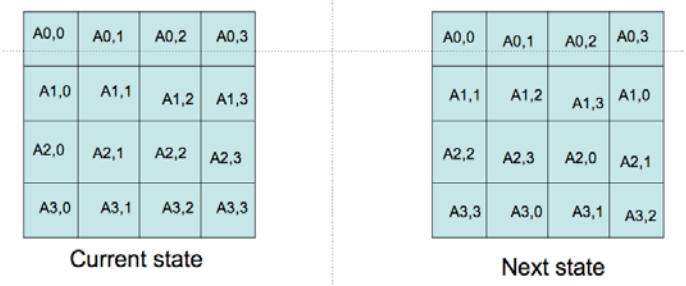

Figure 6. AES Steps: Shift Rows

The **Mix Columns** step performs a transformation on the four bytes of state data in each column in the state data matrix.

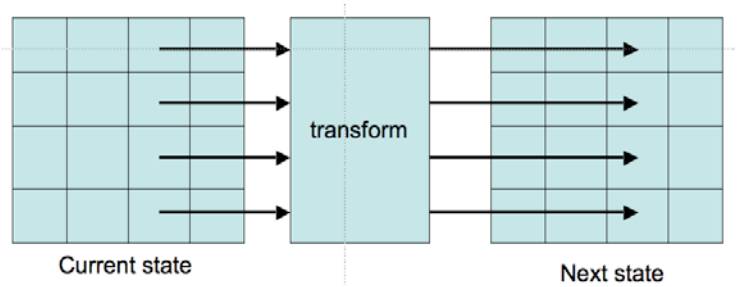

Figure 7. Mix Columns

AES Encryption Modes of Operation

There are several different encryption modes. One of these modes must be chosen for encryption or decryption to be performed.

ECB, **E**lectronic **C**ode**B**ook mode, uses the AES encryption and decryption algorithm previously discussed on blocks of plaintext. It is the most simple mode of operation and very predictable: identical input plaintext always results in identical encrypted ciphertext. This leads to code breaking because the ciphertext can be deciphered by applying pattern matching techniques.

CBC, **C**ipher **B**lock **C**haining mode, provides more security than ECB by adding a feedback method to encryption and decryption. This means the same block of plaintext data will yield different ciphertext results each time the CBC encryption is run.

With CBC mode, an XOR is performed on the input plaintext and the previously encrypted (or decrypted) ciphertext. Since previously encrypted or decrypted data is not available for the first operation, an *initialization vector* must be provided. CBC works on complete 128-bit blocks of plaintext, so if the plaintext is less than 128-bits long, the data must be padded.

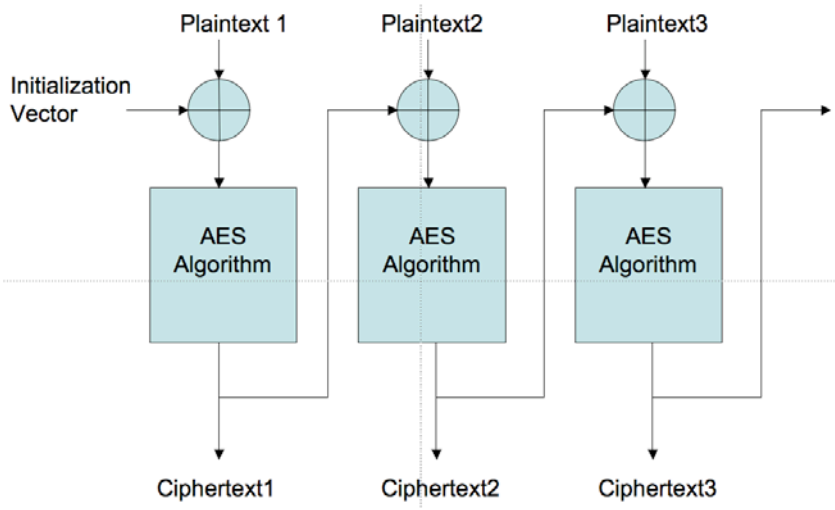

Figure 8. CBC Mode Encryption

CFB, **C**ipher **F**eed**B**ack mode, is similar to CBC but may use only a portion of the previously encrypted (or decrypted) data. For example, if 8 bits are specified for CFB, then only 8 bits of the previously encrypted (or decrypted) data is used to perform an XOR with the next input data. This has an advantage over ECB and CBC in that the entire 128-bit input data block need not be available to start encryption if the number of bits being used for the feedback is less than 128. As in CBC, an initialization vector must be provided for the first operation.

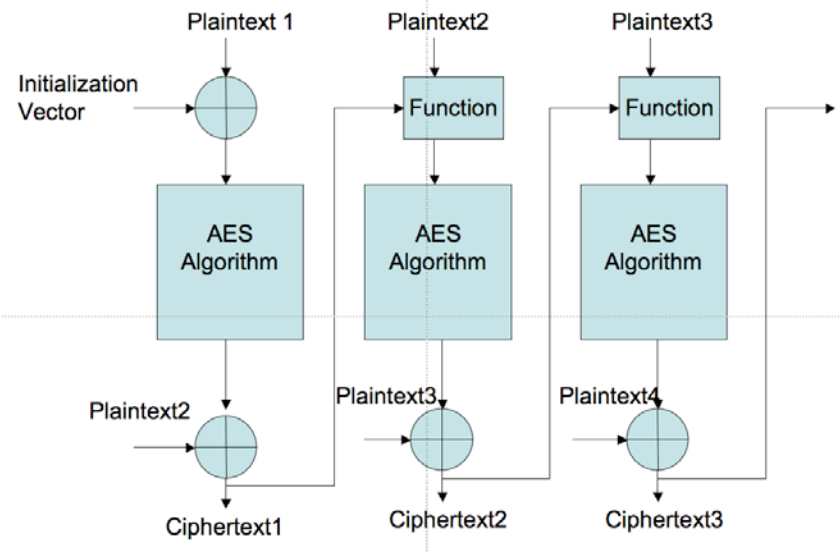

Figure 9. CFB Mode Encryption

OFB, **O**utput **F**eed**B**ack mode, is similar to CFB, except that the encrypted (or decrypted) data is used as an input to an XOR function with it and the previously encrypted data.

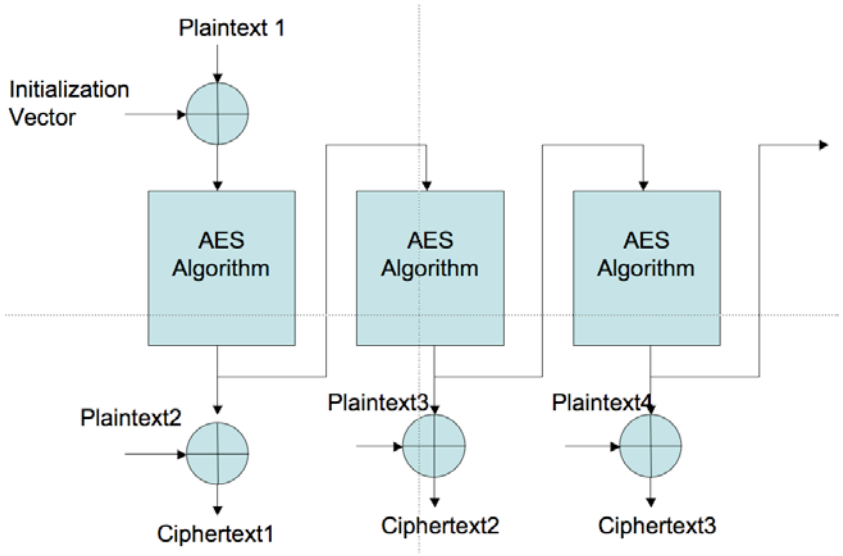

Figure 10. OFB Mode Encryption

CTR, **C**oun**T**e**R** mode, is similar to OFB with the addition of a counter that is incremented after each encryption (or decryption) operation. (The counter can actually be any function that produces a series of bit sequences that do not repeat for a long period of time.)

The modes just described — EBC, CBC, CFB, OFB, and CTR — are the basic block cipher modes described in the FIPS 197 specification. These serve well for the majority of applications for which AES IP might be employed. Additional modes have been more recently developed, such as GCM.

GCM, **G**alois/**C**ounter **M**ode, combines the CTR mode with a 128-bit Galois Field multiplication and a 128-bit Galois Field addition for authentication. The multiplication is easily implemented in fast hardware logic, allowing for higher throughput than AES with the common modes. An initialization vector is required, and authenticated data (non-encrypted) must also be provided in addition to the plaintext input. GCM produces both ciphertext and an authentication tag of up to 128 bits. GCM mode is used in Ethernet security (IEEE 802.1AE), Fibre Channel security (INCITS) , tape storage (IEEE P1619.1), and the IETF IPSec standard.

AES Mode Observations

It is important to understand trade-offs beyond the degree of security when selecting a particular AES mode.

Error resiliency is one factor affected by mode. When the encryption (or decryption) operation uses the previously encrypted (or decrypted) output, the data becomes non-recoverable in the event of a data transmission error. EBC mode does not have this characteristic, and is more error-resilient than the other basic modes. CBC and CFB will propagate a single-bit error to corrupt the current and next block. OFB and CTR will not propagate a single-bit error, but instead will corrupt the entire current block.

In the event of a block synchronization error between the encryptor and decryptor, all modes except CTR will self-sync after one block. CTR must be resynchronized with a new initial vector.

Data padding is another factor. As mentioned, the ECB and CBC modes require complete 128-bit data blocks, and so data less than 128-bits in length must be padded. This is not the case for the CFB, OFB or CTR modes, which can work with shorter data bit lengths.

CAST's AES Implementation

CAST's AES IP offerings are designed to be some of the most versatile yet fastest and most compact cores available.

Architectural Design

The speed of the AES algorithm in hardware is a function of IP implementation details. CAST's IP is architecturally-optimized for speed by combining the Sub Bytes and part of the Mix Columns steps into a single look-up table operation.

The result of this and other design details is that with a 128-bit data bus, only one clock cycle is required to load or unload each 128-bit block of data for encryption or decryption. This enables a high AES data transmission bit rate, with ASIC implementations achieving throughout rates of 5 Gbps or more.

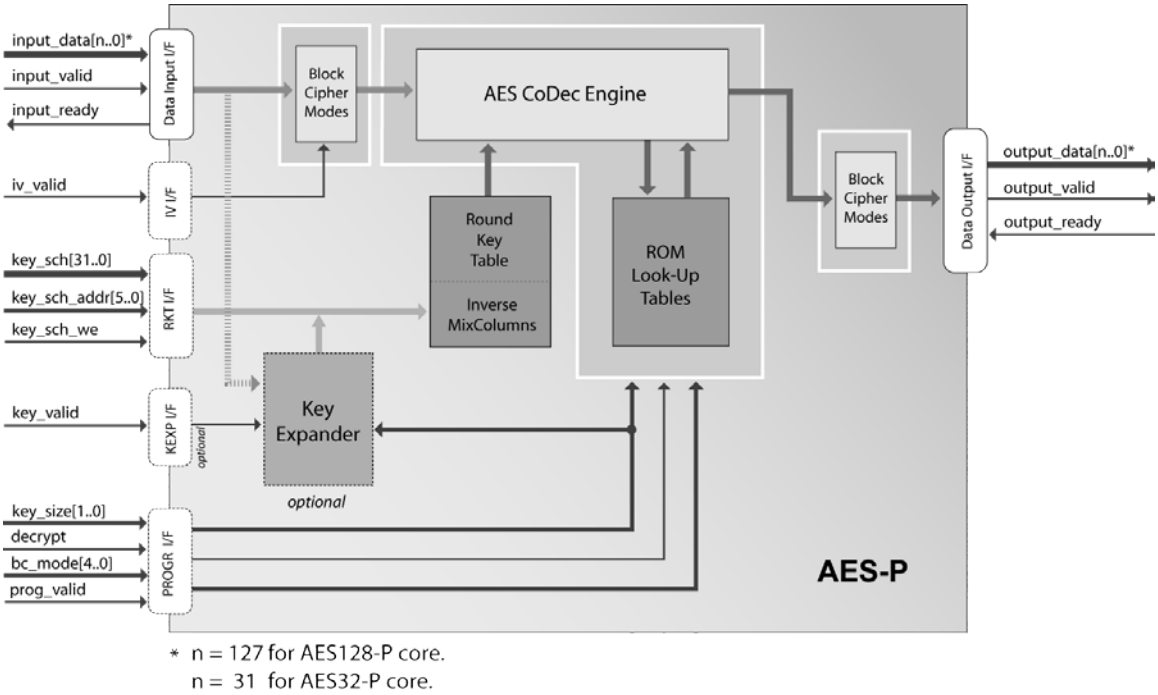

Figure 11. Block diagram for the AES-P IP core

For application where such a high bit rate is less important, an available variation of the CAST AES cores uses a 32-bit data bus, requiring 4 clock cycles to load/unload each

128-bit data block. Allowing the option of 32-bit or 128-bit data bus allows for size/performance trade-offs within the system design.

The efficiency of CAST's AES IP implementation enables allows each core to perform both encryption and decryption, simplifying system design and integration. Input control signals specify which operation to perform, and the cores can be switched between encrypt and decrypt at the beginning of each data block without any performance penalty.

The key size and cipher modes are also controlled by input control signals, and may be changed at the beginning of each data block without any performance penalty.

An extremely useful and unusual feature in CAST's AES IP is fully-stallable input and output interfaces. This allows the application receiving the core's output to arbitrarily pause the output data generation. In a similar way, the application that feeds the input data to the AES IP can arbitrarily pause the input data stream. This allows the CAST AES IP to be responsive to the speed of the input and output applications.

Additionally, the AES IP core can stall the application if the core is busy processing, so that the input is stopped until the core frees up to receive more data for processing. This allows the IP core to be integrated in any application regardless of the input data rate.

Multi-Purpose and Optimized Cores

CAST's family of AES IP cores consists of variations of two basic products.

The AES-P is a multi-purpose, programmable encrypt/decrypt core. It supports all the common block cipher modes — ECB, CBC, CFB, OFB and CTR — and application software can switch between them in real-time.

The AES-C encrypt/decrypt core can also support all the same AES modes, but one specific mode must be selected prior to synthesis, and the resulting implementation is optimized for that mode.

Get more details and see implementation results In the AES-P and AES-C datasheets.

Optional Key Expander Module

The round key values for AES processing are stored in the IP's Round Key Table memory. For applications where the cipher key changes often or fast processing is important, it can be best to use CASTs optional Key Expander to automatically generate and store the round key values from the cipher key. Otherwise, the key expansion can be performed by system software (and the round keys still stored in the internal Round Key Table memory). This can reduce overall hardware area of the ASIC or FPGA, but is only suitable for slower applications, and those where the cipher key tends not to change.

Note that the decryption process uses the expanded round key in the inverse order from encryption. Also, the round keys must be transformed for decryption using an inverse mix columns function. CAST's AES IP takes care of this inversion process automatically, using minimal resources and no performance penalty.

FIPS Validation

The validation of the AES CAST IP core includes the use of Federal Information Processing Standards (FIPS) vectors supplied by the National Institute of Standards and Technology (NIST) and the Communications Security Establishment (CSE) of Canada. The test vectors provide the correct encryption for a given input and key. The set of FIPs validation vectors are part of the deliverables with the AES IP cores.

An automatic test vector generation capability is included with the deliverables. The test vector generation uses a Bit Accurate Model of the AES IP core to determine the expected output vectors. Input vectors are quickly generated using simple ASCII text files.

Applications

The employment of the AES encryption and decryption algorithms are suitable for a variety of applications, such as secure networking routers, wireless communications, encrypted data storage including secure Smart Cards, secure video surveillance systems, secure RFID and electronic financial transactions.