

AES Encryption

This is an implementation of the Rijndael block cipher, including the Advanced Encryption Standard (AES) cipher with 128, 192, or 256-bit keys. This is useful for encrypting sensitive data to be sent over unsecured network paths.

Cipher feedback encryption is a method for using a block cipher like Rijndael to encrypt a stream of characters. It does this by maintaining a feedback register which is the same size as the cipher block size. The feedback register is loaded with a known initial set of data (the init vector) and then encrypted. Each byte in the feedback register is XORed with a plain text byte to produce the cipher text. When a feedback byte is used it is shifted out of the feedback register and the new ciphertext byte is shifted in. Each time a full block of new ciphertext has been shifted in to the feedback register, it is encrypted again. This scheme allows any number of bytes to be encrypted when needed. Since XORing the ciphertext with matching bytes from the feedback register will recover the plaintext, decryption is a similar process, where the decrypter's feedback register is kept synchronized with the encrypter.

See “Applied Cryptography, 2nd Edition” by Bruce Schneier (pg. 200) for a more complete explanation of the encryption/decryption process.

The Rabbit implementation of cipher feedback mode uses an AESstreamState structure to hold the state of the feedback register. Separate states must be used for encrypting and decrypting.

The specification for the Rijndael cipher contains a set of known answer tests (KATs) that are used to verify that an implementation of the cipher is correct. A known answer test consists of a block of plaintext, a key, and a corresponding block of cipher text. See the sample programs located in the `/Samples/AES_Encryption` directory for more details.

API Functions

The following functions are included with the AES encryption module.

AESdecrypt

```
void AESdecrypt( char * data, char * expandedkey, int nb, int nk );
```

DESCRIPTION

Decrypts a block of data in place.

PARAMETERS

data	Pointer to a block of data to be decrypted.
expandedkey	Pointer to a set of round keys (generated by <code>AESexpandKey()</code>).
nb	The block size to use. Block is $4 * nb$ bytes long.
nk	The key size to use. Cipher key is $4 * nk$ bytes long.

LIBRARY

`AES_CRYPT.LIB`

AESdecryptStream

```
void AESdecryptStream( AESstreamState * state, char * data,
    int count );
```

DESCRIPTION

Decrypts an array of bytes.

PARAMETERS

state	Pointer to the <code>AESstreamState</code> structure.
data	An array of bytes that will be decrypted in place.
count	Size of data array

LIBRARY

`AES_CRYPT.LIB`

AESdecryptStream_CBC

```
void AESdecryptStream_CBC( AESstreamState * state, char * data,
    int count );
```

DESCRIPTION

Decrypts an array of bytes using cipher-block chaining.

PARAMETERS

state	Pointer to the <code>AESstreamState</code> structure
data	An array of bytes that will be decrypted in place. Must be padded to be an integer multiple of 16-byte blocks.
count	Size of data array. Must be a multiple of <code>_AES_CBC_BLK_SZ_</code> .

LIBRARY

`AES_CRYPT.LIB`

AESdecryptStream_CBC_XMEM

```
int AESdecryptStream_CBC_XMEM( AESstreamState * state, long message,  
    long output, unsigned int count );
```

DESCRIPTION

Perform an AES-CBC decryption operation.

PARAMETERS

state	Pointer to an AES stream state structure.
message	The cipher-text message (an xmem buffer)
output	The output buffer, for return of decrypted text (in xmem). Must be as large as the cipher-text buffer. May be the same as the cipher-text buffer.
count	Length of the message. Must be a multiple of <code>_AES_CBC_BLK_SZ_</code> .

RETURN VALUE

0 on success
non-zero on failure

LIBRARY

AES_CRYPT.LIB

AEEncrypt

```
void AEEncrypt( char * data, char * expandedkey, int nb, int nk );
```

DESCRIPTION

Encrypts a block of data in place. Note that AES (as such) uses only nb=4 and nk=4,6 or 8. Rijndael allows different block and key sizes than AES, however performance will be slower if nb is not 4.

You can #define AES_ONLY before #use AES_CRYPT.LIB if you are adhering to AES. This saves some code space.

PARAMETERS

data	Pointer to a block of data to be encrypted; the block of data must be nb*4 bytes.
expandedkey	Pointer to a set of round keys (generated by AESexpandKey () , with the same nb and nk parameters!)
nb	The block size to use. Block is 4 * nb bytes long. Must be “4” for AES.
nk	The key size to use. Cipher key is 4 * nk bytes long. Must be 4, 6 or 8 for AES.

RETURN VALUE

None.

LIBRARY

AES_CRYPT.LIB

AEEncryptStream

```
void AEEncryptStream( AESstreamState * state, char * data,
    int count );
```

DESCRIPTION

Encrypts an array of bytes.

PARAMETERS

state	Pointer to the AESstreamState structure.
data	An array of bytes that will be encrypted in place.
count	Size of data array.

LIBRARY

AES_CRYPT.LIB

AEEncryptStream_CBC

```
void AEEncryptStream_CBC( AESstreamState *state, char *data,
    int count );
```

DESCRIPTION

Encrypts an array of bytes using cipher-block chaining.

PARAMETERS

state	Pointer to the AESstreamState structure
data	Pointer to an array of bytes that will be encrypted in place. Must be padded to be an integer multiple of 16-byte blocks.
count	Size of data array. Must a multiple of <code>_AES_CBC_BLK_SZ_</code> .

LIBRARY

AES_CRYPT.LIB

AESencryptStream_CBC_XMEM

```
int AESencryptStream_CBC_XMEM( AESstreamState * state, long message,
    long output, unsigned int count );
```

DESCRIPTION

Perform an AES-CBC encryption operation on XMEM data. Encryption is not “in-place.” Call `AESinitStream()` before using this function. Enable `AES_DEBUG` while developing to detect block-size errors.

PARAMETERS

state	Pointer to an AES stream state structure, initialized
message	The message in plaintext (an xmem buffer)
output	The output buffer, for return of encrypted text (in xmem), must be as large as the plaintext buffer, and may be the same as the plaintext buffer.
count	The length of the message. Must a multiple of <code>_AES_CBC_BLK_SZ_</code> .

RETURN VALUE

0 on success
non-zero on failure

LIBRARY

`AES_CRYPT.LIB`

AESexpandKey

```
void AESexpandKey( char *expanded, char *key, int nb, int nk,
    int rounds );
```

DESCRIPTION

Prepares a key for use by expanding it into a set of round keys. A key is a “password” to decipher encoded data.

PARAMETERS

expanded	A buffer for storing the expanded key. The size of the expanded key is $4 * nb * (rounds + 1)$.
key	The cipher key, the size should be $4 * nk$
nb	The block size will be $4 * nb$ bytes long.
nk	The key size will be $4 * nk$ bytes long.
rounds	The number of cipher rounds to use.

RETURN VALUE

None.

LIBRARY

AES_CRYPT.LIB

AESinitStream

```
void AESinitStream( AESstreamState * state, char * key,  
    char * init_vector );
```

DESCRIPTION

Sets up a stream state structure to begin encrypting or decrypting a stream. A particular stream state can only be used for one direction.

See `Samples\AES_Encryption\AES_STREAMTEST.C` for a sample program and a detailed explanation of the encryption/decryption process.

PARAMETERS

state	Pointer to a <code>AESstreamState</code> structure to be initialized.
key	Pointer to the 16-byte cipher key; using a null pointer will prevent an existing key from being recalculated.
init_vector	A 16-byte array representing the initial state of the feedback registers. Both ends of the stream must begin with the same initialization vector and key.

RETURN VALUE

None.

LIBRARY

`AES_CRYPT.LIB`

