The Helion basic guide to AES encryption in hardware

What is AES?

During September 1997, the National Institute of Standards and Technology (NIST), the main standards body in the US, issued an open request for submissions for a new Advanced Encryption Standard (AES). The idea was that AES would be a new encryption standard to replace the existing Data Encryption Standard (DES), which had been in place for more than two decades. In August 1998, a number of potential candidates were selected from the submissions, and then in August 1999, these were further selected to become a shortlist of five finalists. In October 2000, one of these five algorithms, “Rijndael”, was chosen as the new AES. This decision was then fully ratified by the NIST in November 2001, at which point AES became the encryption algorithm of choice for all new commercial developments requiring a high degree of data security.

What is AES like from the outside?

Rijndael is what is known as a “Block Cipher” with multiple options for its block and key size. The NIST approved AES is a subset of these options; the block size is fixed at 128-bits, but the key may be either 128, 192 or 256-bits in length. As their name suggests, Block Ciphers only process data in blocks. In the case of AES this means that it is capable of encrypting plaintext data in blocks of 128-bits using any of the specified key sizes; in broad terms, the bigger the keysize, the higher the security. It is generally accepted that 128-bit keys will provide a sufficient level of security for most commercial applications, though longer keys seem to be specified more and more for the latest applications. Don’t worry if your data doesn’t naturally fall into neat 128-bit blocks; we look at how to deal with this later in this document.

And from the inside?

Internally, in common with many block ciphers, AES consists of a complex non-linear core function, which is iterated multiple times on the incoming plaintext data block. The number of times this iteration is needed, or more correctly, the number of “rounds” required, depends on the selected key size. For 128-bit key AES, there are 10 rounds, for 192-bit key AES there are 12 rounds, and for 256-bit key AES there are 14 rounds. This being the case, the longer key sizes take more time to process, and hence the maximum data bandwidth available necessarily goes down with increasing key size. Note that the round function is slightly different for the final round, and that an initial pre-processing function is also required at the start. For a more in-depth explanation of the AES algorithm, please refer to the official AES specification documentation (NIST FIPS Publication 197), which is available for download at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

What are Roundkeys?

Each round of AES requires a unique 128-bit Roundkey to be fed in to the complex round function. This series of 128-bit Roundkeys are generated from the supplied 128-bit, 192-bit or 256-bit AES key using a specified key expansion algorithm. This expansion yields exactly the right number of Roundkeys to feed the single pre-process step and the multiple rounds. So for 128-bit keysize AES you require 11 Roundkeys, for 192-bit you require 13 Roundkeys, and for 256-bit you require 15 Roundkeys.

How can we generate Roundkeys?

In a hardware implementation of AES, the key expansion process can be accomplished in one of two ways; essentially using hardware in real-time, or offline using software.

The AES key expansion algorithm was designed to be usable “on the fly”, such that the Roundkeys can be expanded iteratively in real-time as and when they are required by the encryption algorithm. This is especially useful if the AES keys need to change on a regular basis. The only penalty here is that additional Roundkey expansion hardware is required. This approach is the most commonly used, as the solution is completely implemented in hardware, with no external support required.

If AES keys do not get changed too often, or absolute throughput is not the limiting factor, then Roundkeys may be expanded off-line in software, in which case the Roundkeys would be stored in memory for subsequent use. This can save a significant number of gates and reduce the total power consumption, and is especially appropriate in low resource implementations in FPGA, where suitable Roundkey buffer RAM is readily available at low cost.

Helion AES cores were the first (and still one of the few) to seamlessly support either approach to Roundkey expansion, as they will happily address external Roundkey memory, or interface easily to matching Helion Roundkey expansion logic. If Helion hardware expansion is used, the user need have no visibility or concern for Roundkeys at all! Most Helion customers decide to go this route, as the efficiently designed hardware typically costs little by way of additional resources.

What about AES Decryption?

Once data has been encrypted using AES, it is obviously necessary to be able to decrypt it. AES decryption is essentially the “mirror image” of AES encryption, and putting AES encrypted ciphertext into an AES decryptor will yield the original plaintext. The internal complex round function for encryption is effectively inverted for use in the decryptor. The Roundkeys are identical, but are required in reverse order.

Unfortunately, this mirroring makes AES encryption and decryption implementations quite different, so creating one hardware block to handle both functions is often not very efficient. Moreover, the inverse round function for decryption is significantly more complex than that for encryption, and so an AES decryptor will always be both bigger and slower in hardware than its matching AES encryptor. However, all is not lost; depending on the AES mode you are using, this mismatch in performance and complexity may not turn out to be a problem (see later).

What is different about Decryption Roundkey generation?

As mentioned above, the roundkeys required by the Decryptor are the same as those used for encryption, but in the reverse order. Both the Encryptor and Decryptor use the identical expanded AES key, but take their roundkeys from opposing ends of the expanded list.

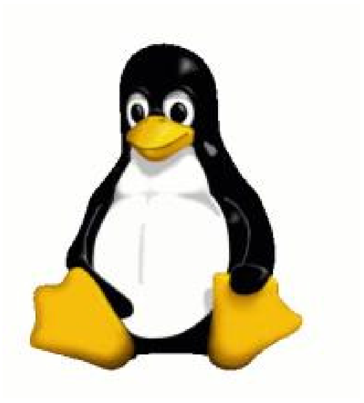
The key expansion algorithm inherently expands the AES key in the order that the Encryptor requires the roundkeys. In this way, a roundkey generation block can output encryption roundkeys as they are required, in the correct order in real-time; no buffering of these roundkeys is necessary. However, for decryption, where roundkeys are required in reverse order, there is no way of algorithmically producing the inverse roundkeys in the correct order *directly* from the supplied AES encryption key.

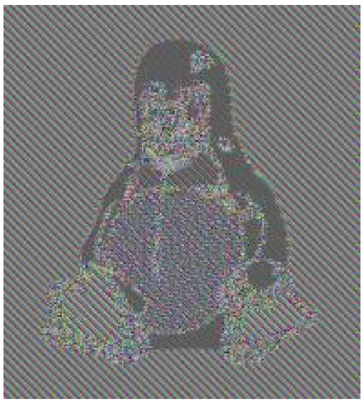
This is not an issue where roundkeys are being generated off-line in software, as the order they are read by the decryption core from a memory is fully controllable; it is more a concern when generating the roundkeys on request in the hardware. Helion do however offer some unique ways to overcome this issue, to allow decryption keys to be changed frequently and expanded in real time as required. These techniques are outside the scope of this overview document, but can be described in more detail on request.

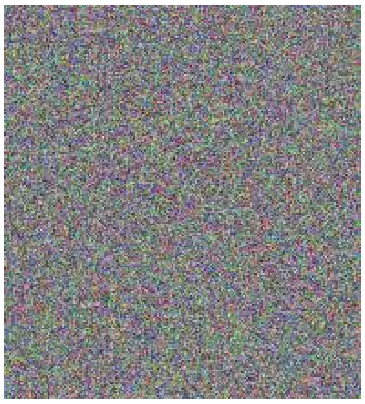
What are AES modes?

The most obvious way to encrypt data using AES is to split it up into 128-bit blocks, and put each block in turn directly through the algorithm. This is called Electronic Codebook mode (ECB), and has the benefits that it is simple and that data does not have to be encrypted in any particular order or context; ideal for databases and the like. However, one of its major shortcomings is that the same plaintext data will always yield the same ciphertext data with the same key; this can affect the security of ECB mode under cryptanalysis, as repeated data may be easily seen in the encrypted result.

A fantastic example of this can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform colour. Although the colour of each individual pixel is encrypted, the structure of the image may still be discerned in the encrypted version. This is because the pattern of identically coloured pixels in the original comes right through the encryption process. For this reason, ECB mode is rarely used in real applications, and deciding to adopt it should be done with caution!


Original image


Encrypted using ECB mode


Encrypted using a feedback mode

Other “Block Cipher Modes” are available which try to overcome this issue. They generally rely on feedback of some kind, so that the ciphertext of a data block is made dependant on the previous ciphertext blocks in a message, and not just its corresponding plaintext data block. In this way, the same plaintext blocks repeated in a message will be converted into radically different ciphertext blocks, thus making cryptanalysis much harder. Common modes of this type include Cipher Block Chaining (CBC), Cipher Feedback mode (CFB), Output Feedback mode (OFB), and Counter mode (CTR), the differences between them being exactly how or where the feedback is applied.

How can the various AES modes be implemented?

All common block cipher modes are usually implemented using a basic ECB mode encrypt or decrypt engine, along with some simple additional feedback logic. The additional mode logic necessary often consists only of XOR functions and storage registers. For this reason, the bare Helion AES cores have been designed to implement ECB mode, and optionally come bundled with ready-made and pre-tested “wrapper” logic reference designs, which may be used along with the cores to implement most of the basic modes with no further effort required. This means you can have exactly the mode coverage you require, without the overhead of modes you don’t need!

What is an Initialisation Vector (IV)?

Any mode which includes feedback or some kind of internal "state" to cover up repeated data, generally requires an Initialisation Vector (IV) to be provided. This can be viewed as being the initial value of the feedback or state before any data has been processed - kind of a "pre-history". To start the encryption of a new stream or message, you will need to feed an associated IV into the encryption process, and the same IV will need to be made available to the decryptor when that same message is decrypted.

Typically the IV need not be kept secret, as knowledge of it does not compromise security, so it is often transmitted unencrypted along with the encrypted message, or it may be derived locally from some information known to both ends of the link, for example a timestamp or message ID.

The various modes have differing requirements for their IVs; for example CTR mode requires that the same IV is never used twice with the same key, so steps must be taken to ensure that repeated IVs never get used.

Some modes require an IV for encryption, but you can choose not to bother transmitting that IV to the decryptor, if you can live with losing the first block of decrypted data. Modes like CFB which self-synchronise have this characteristic, often needing no framing for cryptographic synchronisation or additional bandwidth to transmit explicit IVs.

How can AES modes help the decryption overheads?

Some of the common AES modes do not require an ECB decryptor in order to perform decryption! For example, due to the way in which the feedback is structured, CFB, OFB and CTR modes use an AES ECB Encryptor in both their encrypt and decrypt functions. This is especially useful since the AES ECB decryptor, as detailed previously, has several fundamental disadvantages.

Using an encryptor at both ends of the system, allows a perfectly balanced system to be attained, running at the highest possible rate, without suffering the bottleneck of using a decryptor with its potential speed, size and key update issues. This is something the original designers of Rijndael were fully aware of, and cite as a reason for accepting the complications of their ECB decryption scheme.

What is CCM mode?

The modes mentioned so far have been standard modes that have been around for many years with earlier algorithms like DES. New modes are also being developed, and AES-CCM is one of those. It is slightly more complicated than the more established modes, but has the advantage that it offers both security and authentication in a single algorithm.

Until recently, it has been common to use a completely separate algorithm to provide this authentication function; examples include SHA-1 and MD5, as used in such protocols as SSL and IPSec. CCM allows a block cipher encryption algorithm such as AES to cover both requirements, and can be more efficient in hardware than using the separate algorithms.

CCM also has the advantage that it only ever uses AES Encryptors, even for decryption, and so avoids some of the downsides of the AES decryption algorithm. CCM is used in standards such as IEEE 802.11 Wireless LAN, IEEE 802.15 Personal Area Network and IEEE 1619.1, which is the standard for securing data-at-rest stored on tape media. More details on CCM can be obtained from http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf, which is the NIST specification for this mode.

Helion offer a suite of fully tested and highly efficient solutions to implement CCM based on mature and product proven AES technology; please feel free to contact us for more details.

What is GCM mode?

AES-GCM is another recent authenticated encryption algorithm designed to provide both authentication and privacy. Developed by David A McGrew and John Viega, it uses universal hashing over a binary Galois field to provide authenticated encryption. It was designed specifically to support very high data rates as it can take advantage of pipelining and parallel processing techniques.

Before GCM, it was quite possible to encrypt at very high data rates using Counter mode. This mode uses no feedback, so multiple encryption engines can be used in parallel to offer enhanced aggregate throughputs. However, authentication of that data was then a problem.

Most authentication algorithms like CBC-MAC (as used in CCM mode), CMAC (an AES based authentication scheme) and the various hashing algorithms (like SHA-1 and SHA-2) require block-by-block feedback, making pipelining and parallel operation impossible. AES-OCB mode was a potential way around this, but was held back by multiple intellectual property claims. CWC was another option, but this has implementational complexities which make it less attractive.

So although encryption at very high rates was possible, it was the authentication which could not keep pace. This issue was especially concerning since Counter mode without any authentication provides no protection against bit-flipping attacks. With the advent of GCM, authenticated encryption at data rates of many Gbps is now practical, permitting high grade encryption and authentication on systems which previously could not be fully protected.

AES-GCM is specified for use in a number of recent standards; for example it is one of the options specified by the IEEE 1619 group for securing data-at-rest stored on tape media, and it is also the algorithm specified for use in MACsec (802.1AE) for protecting data traversing Ethernet LANs. It is also part of the NSA's "Suite B" cryptographic suite.

In recent times its simplicity and strength has made it the mode of choice for many applications not necessarily requiring its high rate capabilities, in many cases displacing AES-CCM as the standard for lower rate authenticated encryption.

Helion offer a selection of GCM solutions both for very-high, high and lower rate requirements, again based on mature and product proven AES technology; please feel free to contact us for more details.

What is XTS mode?

The AES-XTS algorithm is in a class known as "Tweakable" block ciphers, more specifically an application specific version of a mode called AES-XEX, specified by the IEEE 1619 working group in their standard for disk encryption, taking over from the previously specified LRW-AES algorithm. In this application, AES-XTS is used to encrypt data at the disk sector level, where it addresses threats such as copy-and-paste attacks and dictionary attacks, while allowing the option of parallel processing for enhanced performance.

When used for disk encryption, AES-XTS encrypts and decrypts blocks of 16-bytes at a time under the control of a secret AES key, and a "tweak" value derived from the logical position of the block on the disk. This algorithm fulfils the fundamental requirements for disk encryption, in that the data can be independently encrypted and decrypted at the sector level as it arrives in arbitrary order, and that the encryption process does not change the data size. In addition, the location of the data on the disk will vary the encrypted result, so that identical plaintext sectors stored at different places on the disk will be different after encryption.

Helion offer a selection of XTS solutions both for very high, high and lower rate requirements, again based on mature and product proven AES technology; please feel free to contact us for more details.

Why are there so many modes to choose from?

As well as the basic requirement of obscuring repeated data patterns, the mode used also changes the way the system responds to events like errors in the encrypted data, or loss of synchronisation between the encryptor and the decryptor. Choice of mode can also allow you to encrypt arbitrary length messages without having to split them into a whole number of 128-bit AES blocks, or can give you authentication as described above. Modes are therefore very powerful, and an important part of the specification process when deciding how to secure a system.

What mode is best for my application?

The choice of mode may not be immediately obvious, and each mode has its own advantages, disadvantages and implications for security. If your application does not dictate a mode to use, such as for compliance to a standard, then an informed choice should be made. Choice of mode may be dictated by numerous requirements; for example, the type of data you are handling, susceptibility to data error propagation or synchronisation loss, or whether ultimate decryption performance is required.

Where can I learn more about all this?

The Helion website covers this area in some preliminary detail; take a quick look at http://www.heliontech.com/aes_modes_backgnd.htm as a starting point. Beyond this, we are always very happy to advise on mode choice for your specific application; please feel free to contact Helion for a more detailed discussion.

There are also numerous good books available on the subject, but a well known book we can recommend as being both readable and informative is called “Applied Cryptography” by Bruce Schneier, published by John Wiley and Sons, Inc. The latest edition does not cover AES, but if you do want to know more about the broader subject of data encryption, including block cipher modes, this is an excellent place to start.

About Helion

Established in 1992 and based in Cambridge, England, Helion Technology is a specialist provider of data security and encryption IP, with a particular emphasis on high performance solutions in ASIC and FPGA. We offer a range of product-proven Data Security and Lossless Compression IP cores, backed by a team of highly experienced engineers, proudly developing and supporting a world-class portfolio. We pride ourselves on our flexibility and friendliness, and willingness to go the extra mile to get the very best solution for our customers.

Version : V1.2.0