Part of the Comtek Group

PROTECTING YOUR DATA

AES-256 Encryption Using the ODC2 Line Card

Author: Gene Norgard – Vice President, Sorrento Networks International

April 2015

INTRODUCTION

The abundance of hacking and espionage in the world has created an unsecure network environment. Despite ever increasing attempts by corporations and governments to protect its data, major attacks are still common. But with the volume of critical data transfer in today's world – from government agencies to retailers, banks and corporations – the need for secure data sharing remains critical.

Sorrento Networks International has taken the steps to provide a solution for both public and private network structures, based on the most recent government standards. The Advanced Encryption Standard (AES) went into effect as a government standard on May 26, 2002. AES is the first publicly accessible and open cipher approved by the U.S. National Security Agency (NSA) for top secret information.

Sorrento's Optical Data Center (ODC2) line card provides a complete and automated solution to ensure data is transported reliably and fully secure. The main features of the ODC line card are:

- Use of standard 256-bit AES algorithm based on the National Institute of Standards and Technology (NIST) FIPS 140 publication.
- An additional safeguard that scrambles the shared encryption key as it is exchanged end-to-end over a public or private network.
- Automatic generation of a 256-bit seed used in the encryption and decryption key generation.
- Secure encryption handshaking using the recommended Diffie-Hellman key exchange.
- Constant altering of the encryption key synchronously at each end of the line without data disruption.

Diffie-Hellman

The first step for a secure encryption process is the establishment of a shared key used by the encryption and decryption algorithm. The important aspects of this operation is that it must be extremely difficult to decode and only shared with the two designated end points.

There are several recommended methods within the industry that provide a means for two parties to secretly handshake in order to negotiate a shared key over an insecure communications network. One of the earliest methods developed and now widely used is the Diffie-Hellman key exchange. The key part of the process is that a coded message is shared across the network and used to generate an identical key that is nearly impossible to reverse by another party that might be listening.

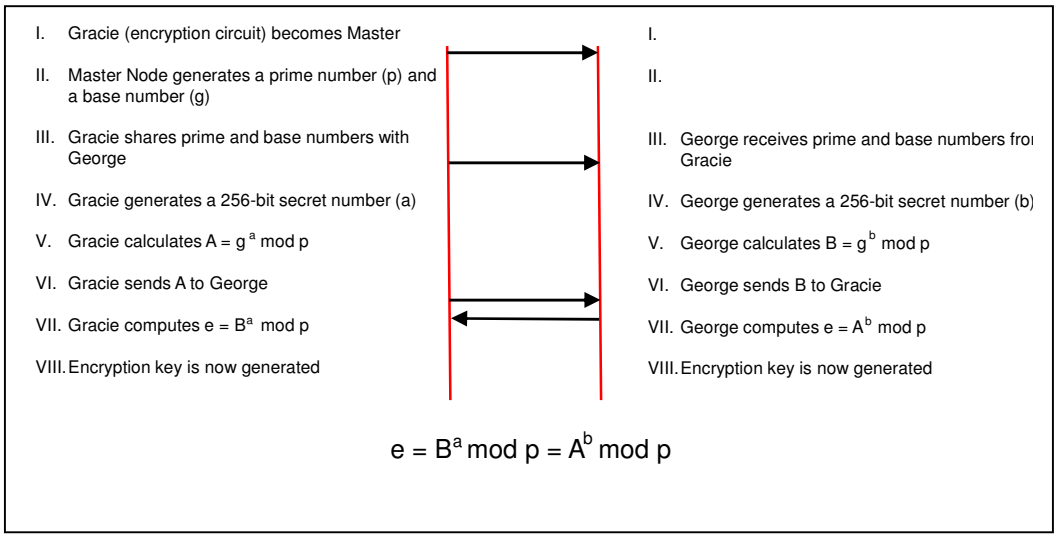
The original implementation of the Diffie-Hellman protocol used the multiplicative group of integers: modulo *p*, where *p* is prime and *g* is a primitive root modulo *p*. Since there isn't a reverse operation for modulus calculations, the security stems from the difficulty of calculating the discrete logarithms of very large numbers.

In order to start the Diffie-Hellman exchange, the two parties must agree on two non-secret numbers. The first number is *g*, the generator, and second number is *p*, the modulus. These numbers can be made public and are usually chosen from a table of known values. *g* is usually a very small number while *p* is a very large prime number. Next, each party generates its own secret value. Then, based on *g, p,* and the secret value, each party calculates its public value. The public value is computed according to the formula: $Y = g^x \bmod p$.

In this formula, x is the secret value and Y is the public value. After computing the public values, the two parties exchange their public values. Each party then exponentiates the received public value with its secret value to compute a common shared secret value. When the algorithm completes, both parties have the same shared secret which they have computed from their secret value and the public value of the other party.

The following explains the steps executed by the encryption algorithm (Gracie) and decryption algorithm (George).

1. Gracie and George agree to use a prime number *p=23* and base *g*=5 (which is a primitive root modulo *23)*.
2. Gracie chooses a secret integer *a=6*, then sends George *A*, where $A=g^a \bmod p$. For this example, $A = 5^6 \bmod 23 = 8$.
3. George chooses a secret integer *b=15*, then sends Gracie *B*, where $B=g^b \bmod p$. For this example, $B = 5^{15} \bmod 23 = 19$.
4. Gracie then computes $e = B^a \bmod p$, that is, $e = 19^6 \bmod 23 = 2$.
5. George computes $e = 8^{15} \bmod 23 = 2$.
6. Gracie and George now share a secret encryption number *e=2*.



Both the encryption and decryption circuits have arrived at the same value because $(g^a)^b$ and $(g^b)^a$ are equal mod *p*. Note that only *a*. *b*, and (g^{ab} *mod p* = g^{ba} *mod p*) are kept secret. All the other values are sent in the clear. Once the secret encryption number is computed and sent, the resultant encryption key can be used for sending messages across the same open communications channel.

Of course, much larger values of *a*, *b*, and *p* would be needed to make this example secure. However, if *p* is a prime number of at least 256 digits, and *a* and *b* are at least 100 digits long, the fastest modern computer could not determine *a* given any or all of the other variables.

The previous paragraphs describe the key exchange process, but this alone is not enough to provide a secure network. It's possible for a hacker to tap onto a communications channel with an equivalent decryption machine and override the destination node with its own. To prevent unwanted "listeners" to participate in the key exchange, further scrambling and handshaking needs to take place across the network to insure the correct endpoints are synchronized. The Sorrento ODC2 line card takes this extra step by requiring the user to enter a 32-character word that is password protected. The 32-character word will be identical at each end point and used to scramble the handshaking sequence and secret scrambling number shared between the encryption and decryption circuits.

AES Encryption Process

AES comprises three block ciphers, AES-128, AES-192, and AES-256. Each cipher encrypts and decrypts data blocks of 128 bits using cryptographic keys of 128-, 192-, and 256-bits, respectively. Symmetric or secret-key ciphers use the same key for encrypting and decrypting, so both the sender and the receiver must know and use the same secret key. All key lengths are deemed sufficient to protect classified information up to the "Secret" level with "Top Secret" information requiring 192- or 256-bit lengths. There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition, and mixing of the input plaintext and transform it into the final output of ciphertext.

Sorrento Networks' ODC2 Encryption Operation

The ODC2 line card performs encryption/decryption over standard optical traffic following the AES-256 specifications. The unique design allows encryption to be used at any of the line rates supported by the ODC2 line card. The Diffie-Hellman key exchange is used to share a secret key between the two end-points of the data channel.

The encryption process utilizes the AES-GCM core with a 64-bit word-based data interface to implement the block cipher confidentiality and authentication routines. The core can support various key sizes (GCM = 256 for this design), a fixed nonce/IV size of 128 bits, and a fixed length tag (Message Integrity Check or MIC).

The AES-256 Core integrates together all of the AES and Hash functions together with the counter mode logic, hash length counters, final block padding, and tag appending and checking features. External framing circuitry is also included. The frames establish a demarcation where frame overhead and

collateral system information such as Initial Vectors (IV), Key Update messages, and Message Integrity Codes are passed in the clear while the data itself is encrypted.

The hardware has a Crypto Layer Management interface which implements commands for encryption to be enabled or disabled, allows new keys to be loaded slightly ahead of time, coordinates key changes to be initiated at the transmit end, and provides status on MIC failures at the receive end.

The GigaMux system includes a key management task in the embedded software. This task is responsible for negotiating and computing keys at each end of the link, and communicates with its peer using an unencrypted low rate communications channel in the encrypted frame overhead. At power-up (or on demand), fresh master session keys for each direction are autonomously negotiated using the Diffie-Hellman key exchange.

A cryptographic grade deterministic pseudo-random number generator (PRNG) based on NIST SP800-90A which is iterated identically at each end of the link, provides a sequence of 256-bit AES link keys. These keys are used to calculate a shared value used to calculate a common encryption/decryption key. The lifetime of a link key is operator-selectable from 10 minutes through to 1 day, thus limiting the amount of data encrypted under a single key. Although the link key is automatically updated at least once per day, an internal counter is used to change the keys on every frame boundary.

Link key changes are seamless to the optical client, and are performed at multi-frame boundaries. The encryption module maintains operation with the current key, and has a holding register for the next key. Sometime ahead of each key change, the GigaMux Management Card (MPM2) calculates a new key. The Tx encryption module then takes the lead by setting a “key change” flag in the crypto overhead which is sent across the optical link just before a multi-frame boundary. The Rx encryption module acts on receipt of this flag and changes to the next key at the same point in the data stream.

The client frames are grouped into cryptographic messages covering 8 x 510 64-bit words. Each cryptographic message is processed by the hardware encryption module using AES-256 which provides confidentiality using counter mode (CTR) and integrity using a Galois hash calculation. Overhead is added to the proprietary frame format to allow transmission of a cryptographic Initialization Vector (12 bytes) and Message Integrity Check (16 bytes) for each message. The overhead also provides status flags for controlling key changes and reverse indication of MIC and IV replay failures from the receiving end.

The generation of IVs is autonomous in the Tx encryption module. The Tx end inserts the IV into the frame overhead before each message, and the Rx end extracts it from the frame for decryption. For GCM, the requirement for each message IV to be unique is met by using a simple counter for the IV, which is reset every time a new key is loaded. Note that IV exhaustion is not possible as the IV counter size (32-bits) can accommodate data rates up to 14.025G within the lifetime of the key.

Replay attacks can be detected at the Rx end very easily. Because a simple counter is used for the GCM IV, the ODC simply rejects any IVs numerically less than or equal to the last successfully received IV for the duration of a link key.

The GCM algorithm requires that data is not forwarded on until the MIC is verified. This requires a store & forward stage after the line-rate decryption for the entire cryptographic message, since the MIC validation cannot be completed until a short time after the last message data is received. This buffering adds a latency of approximately 17.3us at the Rx end at 14Gbps (60us at 4Gbps). A message in which the MIC fails has the frame payloads replaced with a NULL payload, and the appropriate alarms and statistics counters are updated.

Conclusion

Encryption is essential to insure the private transfer of sensitive data across a public or private network. AES-256 has been tested and approved by the US government and is the standard means for high security networks. Another aspect for secure networks is the ability to exchange encryption keys across the network in a secretive fashion. Diffie-Hellman is an accepted method but this alone is not enough as an unknown source could be listening and trying to synchronize with the encryption source. To prevent this, another layer of coordinated handshaking between the two end-points must be created. Sorrento Networks International has built these processes into its ODC2 line card to insure complete privacy of all encryption functions and most importantly, your private data.

For more information, please email us at info@sorrentonet.com or view our website at www.sorrentonet.com.