ALERT2 Encryption and Authentication

David Van Wie, Adam Torgerson, R Chris Roark

June 2, 2017 / Revised May 2, 2018

Introduction

With the increase is usage of the ALERT2 protocol, it has become clear that there is a need to support encryption and authentication at the protocol level. Encryption is valuable to protect any sensitive information transmitted via the ALERT2 protocol, and as a means to perform authentication. Authentication -- verifying that a message is genuine -- is an essential part of any command and control system. Recent attacks [1] have shown that even public safety systems are not immune from intentional abuse and hacking.

There are several immediate use cases for encryption and authentication in ALERT2:

- Command and Control - control of flashers, gates, and warning sirens, for example.
- Sensor data near hydroelectric sites - This information must be protected for some duration before it is released to the public.
- Remote IND configuration and management - "Over-the-air" management of an ALERT2 device should require authentication.

In this document, we propose adding industry-standard encryption techniques to the ALERT2 protocol, at the MANT layer, to provide both authentication and encryption.

Discussion

MANT Layer

We propose implementing the encryption changes at the MANT layer, leaving the MANT header in plain text. Repeaters, then, need not know how to decrypt a message in order to process it, and plain text and encrypted messages can coexist on the same system.

Providing encryption at the MANT layer ensures that secure service is available to both IND and APD devices. It is our hope that, by providing encryption at the IND, it will be adopted rapidly and widely throughout the community for command and control applications.

Encryption Algorithm

We propose the use of AES-128 as the core encryption algorithm for use in ALERT2. The Advanced Encryption Standard (AES) is an encryption algorithm adopted as the standard by National Institute of Standards and Technology (NIST) in 2001 [2], and has been approved for encryption of classified data by the NSA [3].

Essential attributes of an encryption algorithm for use in the ALERT2 protocol include:

- Performance - the algorithm must be suitable for use on low-power embedded systems.
- Efficient operation on small blocks of data - the chosen encryption technology must not dramatically increase the size of the message.
- Symmetric - because we only want known sites to be able to transmit to other sites, a public/private key system is not required for this application.
- Broadly accepted - selecting an algorithm with broad acceptance and a high level of security will make it easier for the protocol to gain acceptance in different applications.

The AES algorithm performs well on all of the attributes described above. There are a large number of freely available open source implementations of AES [4] optimized for a variety of different embedded platforms -- including the widely-used AVR32 processor -- in addition to variants optimized for PCs. AES is a block cipher (operating on fixed blocks of 128 bits), but it can also be used as a stream cipher, where the number of output bits matches exactly the number of input bits.

AES Modes

Block ciphers, such as AES, do not provide a complete encryption solution in isolation. The crux of the issue is that the same 128-bit input sequence produces the the same 128-bit output sequence. This can cause information to "leak" through the encryption if, for example, the cipher is being used to encrypt data that has long strings of repeated values. The industry has established different "modes of operation" that describe a process for using a block cipher where this problem is addressed.[5, 6]

Of the different modes of operation, counter mode (CTR) is particularly well suited to low-bandwidth applications because it introduces no message-size overhead. In CTR mode, the sender and the receiver both need to know the key and one additional piece of shared information, called a *nonce*. The *nonce* need not be secret, but should never be reused with the same key.

Authentication

This proposal defines a simple authentication scheme: if a message can be successfully decrypted using a shared encryption key, it is considered to be genuine. In order for an encrypted message to be transmitted and received successfully, both the sender and the receiver must use the same secret key.

In short, anyone possessing the encryption key is authenticated. This means that system maintainers must have a process for retiring keys in the case that a key is compromised, and should cycle keys on a regular schedule.

A word of caution is warranted here: it is difficult to prevent an attacker with physical access to a programmed ALERT2 modem from recovering an encryption key. However, ALERT2 IND manufacturers should take reasonable precautions to secure encryption keys.

Methods of Attack

In addition to traditional attacks on the encryption algorithm or the key, a security solution for ALERT2 needs to be concerned with *replay attacks* and *forgeries*.

In a replay attack, the attack need not know how to decrypt a message. Instead, the attack simply records the message and plays it back later. For example, an attacker might record the command used to turn on a warning siren during a planned test, and then attempt to broadcast that same message, unaltered, at a later time.

In a forgery, an attacker attempts something similar - leaving the encrypted payload of a message intact, but modifying the message metadata. For example, taking an "open the gate" message intended for site A, changing the destination address to site B, and then transmitting it.

By using an ever-increasing message ID and a cryptographic hash, we can protect against both types of attacks. In CTR mode, we combine this message ID with the source address of the MANT PDU and use that as this *nonce*.

Proposed Updates

API Updates

The IND shall support the following API methods:

Name	Type	Length	Min	Max	Default	Description
Encrypt Outgoing Transmissions	130	1	0	1	0	0=off, 1=on
Encryption Source Address To Configure	131	2	0	65534	0	source address to apply API commands 132 to 135 against, 0=global
Encryption Key Rotation Time	132	4	0	2^32-1	0	Time at which encryption key change takes effect, in seconds since Jan 1, 1970, UTC. 0=immediate
Set Encryption Key	133	16	N/A	N/A	N/A	[16 Byte Key]
Remove Encryption Key	134	0	N/A	N/A	N/A	remove encryption key
EMID	135	3	0	2^24-1	0	[3 Byte EMID Value]
Encryption Address List	136	0	N/A	N/A	N/A	Useful on decoders, returns a TLV containing two-byte address for which

						an encryption key is set or pending.

The Encryption Key and EMID methods (124-125) use the values specified in the *Encryption Source Address To Configure* and *Encryption Key Rotation Time* to determine if the settings are to be applied site-wide, or to a specific source address. It is recommended that methods 131 and 132 be called immediately before calling methods 133-135.

If the *Encryption Source Address To Configure* is set to 0, the resulting encryption key is said to be a general purpose key; if the *Encryption Source Address To Configure* is set to another value, the key is said to be site-specific.

For each source address with an encryption key, the IND must maintain both an active key and a pending key with a rotation time. When setting encryption keys, the IND shall compare the *Encryption Key Rotation Time* configuration variable to the current time. If this value is set to a time in the future, the change does not take effect immediately. Rather, the active key is left intact, and the pending key and rotation time are updated. If the time specified in the *Encryption Key Rotation Time* variable is less than or equal to the current time, the pending key is cleared and the active key is set to the specified value. When an active key is set, the EMID values for all sites associated with that key are reset. For a source-specific key, this is just the EMID associated with that key's source address. For a general purpose key, this is all EMID's not associated with a source-specific key.

As a security measure, the IND device shall not return encryption keys via the API.

Outgoing transmissions shall be encrypted when the *Encrypt Transmissions* API flag is set. If the source address of the message matches an active site-specific key known to the IND, that key is used for encryption; otherwise, the general-purpose key is used. If the *Encrypt Transmissions* flag is set but no matching source-specific key and no general-purpose key is present, the IND shall not transmit the MANT.

Upon receipt of an encrypted message, the receiving IND shall first check to see if a matching source-specific encryption key is known and active. If so, that key shall be used for decryption. If not, the general-purpose key shall be used.

The *EMID* command sets or gets the value of the EMID associated with an encryption key. Received messages must have an EMID greater than or equal to the EMID value set/returned by this API method, and the next outgoing message will be encoded using this EMID value.

Because the IND updates EMID values in non-volatile memory during normal operations, EMID and Encryption Key commands are written directly to non-volatile memory, rather than when the API save command is given.

Binary and ASCII Output Updates

A critical part of the Encryption and Authorization protocol extension is communicating to downstream consumers of the ALERT2 data that a message was successfully decrypted and should be considered genuine. This information needs to be communicated using both the ASCII and the Binary output protocols.

For binary communication, the IND shall send an additional TLV with ID 1028, length 1, and value of 1 if a message was successfully decrypted on receipt or 0 if it was sent unencrypted. This TLV should be added to the MANT PDU section of output described in the “ALERT2 Demodulator & Decoder Binary Asynchronous Serial Interface” section of the ALERT2 IND API Specification document.

For ASCII communication, the IND shall send an additional metadata message prior to each “N” message with additional metadata. The format of the M message is:

M,	Message Type (P/N),	Year,	Month,	Day,	Hour ,	Minute,	Second,	ENC=Value

The Value field shall contain 0 if the message was sent in plain text, or 1 if it was sent encrypted. The M message may be omitted if the value is 0.

The timestamp of the M message shall be the same as the N or P message with which it is associated.

MANT Updates

The MANT header will remain in "clear text", regardless of whether or not a message is encrypted. In order to maintain backwards compatibility with existing installations, the encryption implementation does not change the length of the MANT header, instead inserting the extra data required for encryption into the MANT payload.

The first of the three reserved bits in the MANT Header will become an "encrypted payload" flag.

The MANT header will then look like:

Byte	Bits	Field	Hash
0	7-6	Version	include
0	5-3	Protocol ID	include
0	2	Timestamp Service Request Flag	include
0	1	Add Path Service Request Flag	include
0	0	Destination Address in Header	include
1	7-4	Port	include
1	3	Encrypted Payload	include
1	2-1	Reserved	include
1	0	ACK Flag	include
2	7	Added Header Flag	include
2	6-4	Hop Limit	mask
2	3-0	Payload Length	include
3	7-0	Payload Length	include

4	7-0	Source Address	include
5	7-0	Source Address	include
[6]	7-0	Destination Address	include if present
[7]	7-0	Destination Address	include if present
[8]	7-0	MANT PDU ID	include if present
[9]	7-0	Number of Added Source Addresses	exclude
[10]	7-0	[Source Address List]	exclude
[11]	7-0	[Source Address List]	exclude

If encryption is enabled, before transmitting a MANT PDU, the IND shall prepend a 3-byte *Encrypted Message ID*(EMID) to the beginning of the MANT payload. The EMID value shall begin at zero, and shall increment by one each time the IND creates a MANT PDU. The value of the EMID should be stored to nonvolatile memory after it has been incremented. Users may set the next EMID value using the *EMID* API command, or reset the EMID to zero by changing the encryption keys. The same Source Address and Encryption Key should never be reused with the same EMID, and the receiving device will refuse to decrypt messages with an EMID that is less than or equal to the last valid EMID received. In the event that a device needs to be replaced, the EMID must be set to a value greater than the last transmitted message, but it is not necessary to set the EMID to exactly the next value in the sequence; so long as it is greater than the last EMID sent by the device and it is not too close to the maximum value, any number will work. Alternatively, if the encryption keys are changed, the EMID can be reset to 0. This is preferable if the replaced device was compromised (i.e. stolen).

In order to ensure message integrity, the IND shall compute the SHA-1 hash of the MANT header, first masking the the hop limit bits so they are always zero, and excluding the number of added source addresses and the source address list, concatenated with the MANT payload, including the EMID. The IND shall truncate this hash, retaining only the most significant 4 bytes, and append it, most significant byte first, to the MANT payload. The payload length field in the header shall be updated to reflect the additional 7 bytes of payload.

The IND shall then encrypt the MANT payload, starting after the EMID and including the hash, using AES-128 in CTR mode. The *Initialization Vector* used for the AES-CTR mode encryption consists of the two-byte source address and the three-byte EMID value in the high bits, and zeros in the low bits (CTR mode will increment the lower- bits of the IV with every block). If a source-address specific encryption key is set for an outgoing MANT message, the IND will use that for encryption. Otherwise, the global one, will be used

Upon receipt of a MANT message with the encryption flag set, the receiving IND compares the EMID at the start of the payload block to the EMID of the last valid message received from the same source address (stored in nonvolatile memory). If the received EMID is not strictly greater than the last valid EMID received, the message shall be discarded. Note: regardless of whether the transmitters in the system are using a *general purpose* encryption key or a *site specific* encryption key, the transmitter maintains a single, monotonically incrementing EMID value, while the receiver maintains an EMID value for each source address from which it has received encrypted messages. Further, it is possible for the transmitter to be configured to send MANT PDUs using different source addresses -- either through independent source address configuration or through the IND API -- but the transmitter still retains a single EMID and encryption key used across these addresses. EMID values are reset when encryption keys are changed.

If the EMID transmitted with the MANT payload is greater than the EMID stored on the device (from the last successfully decoded message), the IND should decrypt the MANT payload using either the General Purpose key or a Source Address Specific key, if one is present. After decrypting the message, the receiving IND should compute the SHA-1 hash of the MANT header and the decoded payload, and compare the result to the received hash value. If the values do not match, the message shall be discarded.

If the message is found to be valid, the EMID associated with the Source Address of the message shall be updated in non-volatile memory, and the IND shall output the decrypted results. The Encrypted Payload bit shall only be set when the contents of the payload are actually encrypted. After successful decryption, this bit shall be cleared when the message is output.

For backwards compatibility, when using the API v1.0 or v1.1 ASCII format to display a MANT PDU, the encrypted bit shall remain combined with the other reserved bits.

Key Generation

To support a user-friendly and portable mechanism for key generation, it is recommended that user-facing applications implement a process where a 16-byte binary key can be generated from a variable length passphrase by using the first 16-bytes of the SHA256 hash of the passphrase.

For example, the passphrase “this is my passphrase” would generate the key “53:57:CE:17:87:33:55:2B:11:76:D4:30:6E:C2:B1:5D”.

Examples

API Configuration

Before encrypting a message, encryption must be enabled. The following API commands will accomplish that:

Type	Length	Value	Type	Length	Value
117	1	1	133	16	"keep me secret!!"
75	01	01	85	10	6B 65 65 70 20 6D 65 20 73 65 63 72 65 74 21 21

Type	Length	Value
130	1	1
82	01	01

Encryption

Given the following MANT input:

Header									Payload										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
03	00	10	0b	03	e8	06	40	00	70	01	08	12	12	03	24	13	22	02	76

Along with an EMID of 850 and an encryption key of “keep me secret!!”, the encryption process will look like:

- Update the MANT

Header									EMID			Payload										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
03	08	10	12	03	e8	06	40	00	00	03	52	70	01	08	12	12	03	24	13	22	02	76

- Compute the SHA1 hash on the MANT as follows:

Header								EMID			Payload										
0	1	2	3	4	5	6	7	9	10	11	12	13	14	15	16	17	18	19	20	21	22
03	08	00	12	03	e8	06	40	00	03	52	70	01	08	12	12	03	24	13	22	02	76

Note that the hop limit bits is masked, and the source address list is removed. The resulting hash begins with "725db2b2".

- Encrypt the payload

The key and the IV should be:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	6b	65	65	70	20	6d	65	20	73	65	63	72	65	74	21	21
IV	03	e8	00	03	52	00	00	00	00	00	00	00	00	00	00	00

And the results should be:

	Header	EMID	Payload	Hash

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Plaintext	03	08	10	12	03	e8	06	40	00	00	03	52	70	01	08	12	12	03	24	13	22	02	76	72	5d	b2	b2
Encrypted	03	08	10	12	03	e8	06	40	00	00	03	52	11	46	df	21	a1	e0	fb	1e	42	5c	93	1a	7c	62	50

Interactions

Adding encryption enables secure use of the MANT Command and Control protocol. The Command and Control protocol should only accept messages that were encrypted.

Limitations

This scheme introduces 7-bytes of overhead per encrypted MANT PDU.

The Add Timestamp service will work if the originating IND knows the time, but it would not be possible for a downstream repeater to add a timestamp. In order to ensure backwards compatibility, the "Add Timestamp" flag shall be set to 0 on an encrypted MANT packet.

It is recommended that in future versions of the protocol, the EMID, the hash, and the added timestamp data all be moved out of the MANT payload and into their own fields in the MANT header. This would allow the MANT payload to remain intact and unchanged from source to sink, and would enable the use of the Add Timestamp service with encrypted messages in all cases.

This scheme only addresses encryption over an ALERT2 network. IND services that send information over the internet, APD applications, etc., are not covered by this proposal.

References

[1] https://www.nytimes.com/2017/04/08/us/dallas-emergency-sirens-hacking.html

[2] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[3] https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[4] https://en.wikipedia.org/wiki/AES_implementations

[5] Helger Lipmaa, Phillip Rogaway, and David Wagner. Comments to NIST concerning AES modes of operation: CTR-mode encryption. 2000

[6] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation