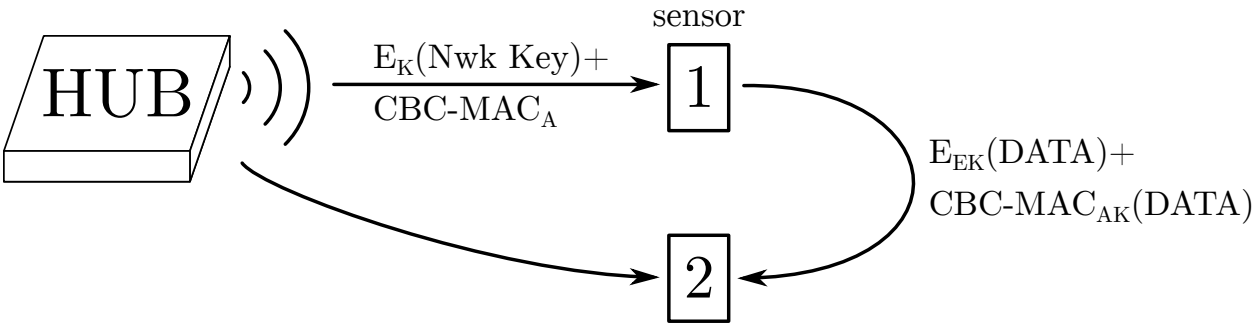
# 3 Carols of the Z-Wave Security Layer; or, Robbing Keys from Peter to Unlock Paul

*by Chris Badenhop and Ben Ramsey*




## 3.1 Adeste Fideles

Z-Wave is a physical, network, and application layer protocol for home automation. It also allows members of the disposable income class to feed their zeal for domestic gadgetry, irrespective of genuine utility. Z-Wave devices sit in their homes, quietly exchanging sensor reports and actuating in response to user commands or the environment.

The curious reader may use an SDR to learn how, when, and what they communicate. Tools like Scapy-radio (Picod, Lebrun, and Demay) and EZ-Wave (Hall and Ramsey) demodulate Z-Wave frames for inspection and analysis. The C++ source code for OpenZwave is a great place to examine characteristics of the Z-Wave application layer. Others may still prefer to cross-compile OpenZwave to their favorite target and examine the binary using a custom disassembler built from ROP gadgets found in the old shareware binary WOLF3D.EXE.

After tinkering with Z-Wave devices and an SDR, the stimulated readers will quickly realize that they can send arbitrary application layer commands to devices where they are executed. To combat this, some devices utilize the Z-Wave security layer, which provides both integrity and confidentiality services to prevent forgery, eavesdropping, and replay.

The first gospel of the Z-Wave security layer was presented by Fouladi and Ghanoun at Black Hat 2013. In it they identified and exploited a remote rekeying vulnerability. In this second gospel of the Z-Wave security layer, we validate and extend their analysis of the security layer, identify a hardware key extraction vulnerability, and provide open source PoC tools to inject authenticated and encrypted commands to sleeping Z-Wave devices.

## 3.2 Deck the Home with Boughs of Z-Wave

This Christmas, Billy Peltzer invests heavily in Z-Wave home automation. The view of his festive front porch reveals several of these gadgets. Billy is a little paranoid after having to defend himself from hordes of gremlins every Christmas, so he installs a Z-Wave door lock, which both Gizmo and he are able to open using a smart phone or tablet. Billy uses a Z-Wave smart plug to control Christmas lights around his front window. He programs the strand of lights to turn on when a Z-Wave PIR (passive infrared) sensor detects darkness and turn off again at daylight. This provides a modest amount of energy savings, which will pay for itself and his Mogwai-themed ornament investment after approximately 20 years.

The inquisitive reader may wonder if Billy's front door is secure. Could a gremlin covertly enter his home using the Z-Wave application layer protocol, or must it instead cannonball through a window, alerting his dog Barney? Fortunately, sniffing, replaying, or injecting wireless door commands is fruitless because the door command class implements the Z-Wave security layer, which is rooted in cryptography.

Z-Wave cryptography uses symmetric keys to provide encryption and authentication services to the application layer. It stores a form of these keys in nonvolatile memory, so that the device does not require rekeying upon power loss. Of the five locks we have examined, the nonvolatile memory is always located in the inner-facing module, so a gremlin would have to destroy a large portion of the Z-

Wave door lock to extract the key. At that point it would have physical access to the lock spindle anyway, making the cryptographic system moot.

Wireless security is enabled on the 5th generation (i.e., Z-Wave Plus) devices on Billy's front porch. Thus, their memory contains the same keys that keep gremlins from wirelessly unlocking his door. A gremlin may crack open the outdoor smart plug or PIR sensor, locate and extract the keys, and send an authenticated unlock command to the door. Billy has figuratively left a key under the doormat!

### 3.3 We Three Keys of AES Are

Since Z-Wave security hinges on the security of the keys, it is important to know how they are stored and used. Z-Wave encryption and authentication services are provided by three 128-bit AES keys; however, the security of an entire Z-Wave network converges to a single key in the set. Like the three wise men, only one of them was necessary to deliver the gifts to Brian of Nazareth. The other two could have just as well stayed home and added a few extra camels to haul the gifts. A card would also have been nice.

The key of keys in this system is the network key. This key is generated by the Z-Wave network controller device and is shared with every device requiring cryptographic services. It is used to derive both the encrypting and signing keys. When a new device is added to a Z-Wave network, the device may declare a set of command classes that will be using security (e.g., the door lock command class) to the Z-Wave network controller. In turn, the controller sends the network key to the new device. To provide a razor-thin margin of opaqueness, this message is encrypted and signed using a set of three default keys known by all Z-Wave devices. The default encryption and authentication keys are derived from a default 128-bit network key of all zeros. If the adherent reader recovers the encryption key from their device, decrypts sniffed frames, and finds that the plaintext is not correct, then they should attempt to use the encryption key derived from the null network key instead.[8]

An authentication key is derived from a network key as follows. Using an AES cipher in ECB-mode, a 16-byte authentication seed is encrypted using the network key to derive the authentication key. The derivation process for the encryption key is identical, except that a different 16-byte seed value is used. A curious reader may want to know what these seeds are, and any fortuitous reader in possession of a MiCasaVerde controller will be able to tell you.

The MiCasaVerde controller uses an embedded Linux OS and provides two mechanisms for extracting a keyfile from its filesystem, located at `/etc/cmh/keys`. Using the web interface, one may download a compressed archive of the controller state. The archive contains the `/etc` directory of the filesystem. Alternatively, a secure shell interface is also provided to remotely explore the filesystem. The MiCasaVerde binary key file (`keys`) is exactly 48 bytes and contains all three keys. The file is ordered with the network key first, the authentication key second, and the encryption key last. Billy Peltzer's Z-Wave network controller is a MiCasaVerde-Edge. In Figure 1, we show the resulting key file and dump the values of the keys for his network (i.e., `0xe97a5631cb5686fa24450eba103f-945c`).

To find the seeds, one must simply decrypt the authentication and encryption keys using an AES cipher in ECB mode loaded with the network key, and the resulting gifts will be the authentication and encryption seeds respectively. From our own observations, the same seed values are recovered from both 3rd and 5th generation Z-Wave devices. Billy's keys are used in Figure 2 to recover the seeds. Given the seed values and a network key, we have a method for deriving the encryption key and the authentication key from an extracted network key.

### 3.4 Away in an EEPROM, No ROM for Three Keys

Z-Wave devices other than MiCasaVerde controllers may not have an embedded Linux OS, so where are the keys stored in these devices? Extracting and analyzing the nonvolatile memory of Billy's PIR sensor and doorlock reveal that the network key is stored in a lowly, unprotected 8-pin SPI EEPROM, which is external to the proprietary Z-Wave transceiver chip. In fact, only the network key is stored in the EEPROM, implying that the encryption key and the authentication key are derived upon startup and stored in RAM.

Unless the device designers hoped to obscure the key derivation process, the decision to store only the network key in nonvolatile memory is unclear.

[8] `unzip pocorgtfo12.pdf zwave.tar.bz2`

Moreover, it is not clear why the key is found in the EEPROM rather than somewhere in the recesses of the proprietary ZW0X01 Z-Wave transceiver module, whose implementation details are protected by an NDA. The transceiver certainly has available flash memory, and there does not appear to be anyone who has dumped the ZW0501 5th generation flash memory yet. Until this issue is fixed, anyone with an EEPROM programmer and physical access can acquire this key, derive the other two keys, and issue authenticated commands to devices. We extract Billy's network key by desoldering the EEPROM from the main board of his PIR sensor and use an inexpensive USB EEPROM programmer (Signstek MiniPRO) to dump the memory to a file.

The circuit board from the PIR sensor is shown in Figure 3. The ZW0501 transceiver is the large chip located on the right side of the board (a 3rd generation system would have a ZW0301). In general, the SPI EEPROM is the 8-pin package closest to the transceiver. The reader may validate that the SPI pins are shared between the EEPROM and transceiver package to be sure. In fact, the ATMLH436 EEPROM used in a 3rd generation door lock is not in the MiniPRO schematics library, so we trace the SPI pin outs of the ZM3102 (i.e., the postage-stamp transceiver package) to the SPI EEPROM to identify its pin layout. We use this information to select a compatible SOIC8 ATMEL memory chip that is available in the MiniPRO library.

We are unable to provide a fixed memory address of the network key, as it varies among device types. Even so, because the memory is so empty (>99% zeros), the key is always easy to find. In all three of Billy's Z-Wave devices, the key is within the only string of at least 16 bytes in memory. The region of the EEPROM memory of Billy's PIR sensor containing the same network key follows, with the key itself starting at address `0x60A0`.

```
~/Downloads/etc/cmh $ ls
alerts.json          HW_Key                  user_data.json.lzo.1
cmh.conf             HW_Key2                 user_data.json.lzo.2
devices              keys                    user_data.json.lzo.3
dongle.3.83.dump.0   last_report             user_data.json.lzo.4
dongle.3.83.dump.1   PK_AccessPoint          user_data.json.lzo.5
dongle.3.83.dump.2   servers.conf.default    vera_model
dongle.3.83.dump.3   sync_kit                wan_failover
dongle.3.83.dump.4   sync_rediscover         zwave_locale
ergy_key             user_data.json.luup.lzo
first_boot           user_data.json.lzo
~/Downloads/etc/cmh $ xxd ./keys
0000000: e97a 5631 cb56 86fa 2445 0eba 103f 945c  .zV1.V..$E...?.\
0000010: 620d 486c 6a65 2122 afe1 086c 79e6 3740  b.Hlje!"...ly.7@
0000020: eec9 ef96 a155 a3d3 02a1 8441 f5f3 7ea0  .....U.....A..~.
```

Figure 1 – Keys found in Billy's MiCasaVerde Edge Controller

```
~/POCs $ ./getSeeds ../keys/veraedge_keyFile
gcry_cipher_open worked
gcry_cipher_setkey worked
gcry_cipher_decrypt worked
A_K: : 62 0d 48 6c 6a 65 21 22 af e1 8 6c 79 e6 37 40
A_Seed: : 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
gcry_cipher_decrypt worked
E_K: : ee c9 ef 96 a1 55 a3 d3 2 a1 84 41 f5 f3 7e a0
E_Seed: : aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
```

Figure 2 – The seeds for the Encryption and Authentication Keys

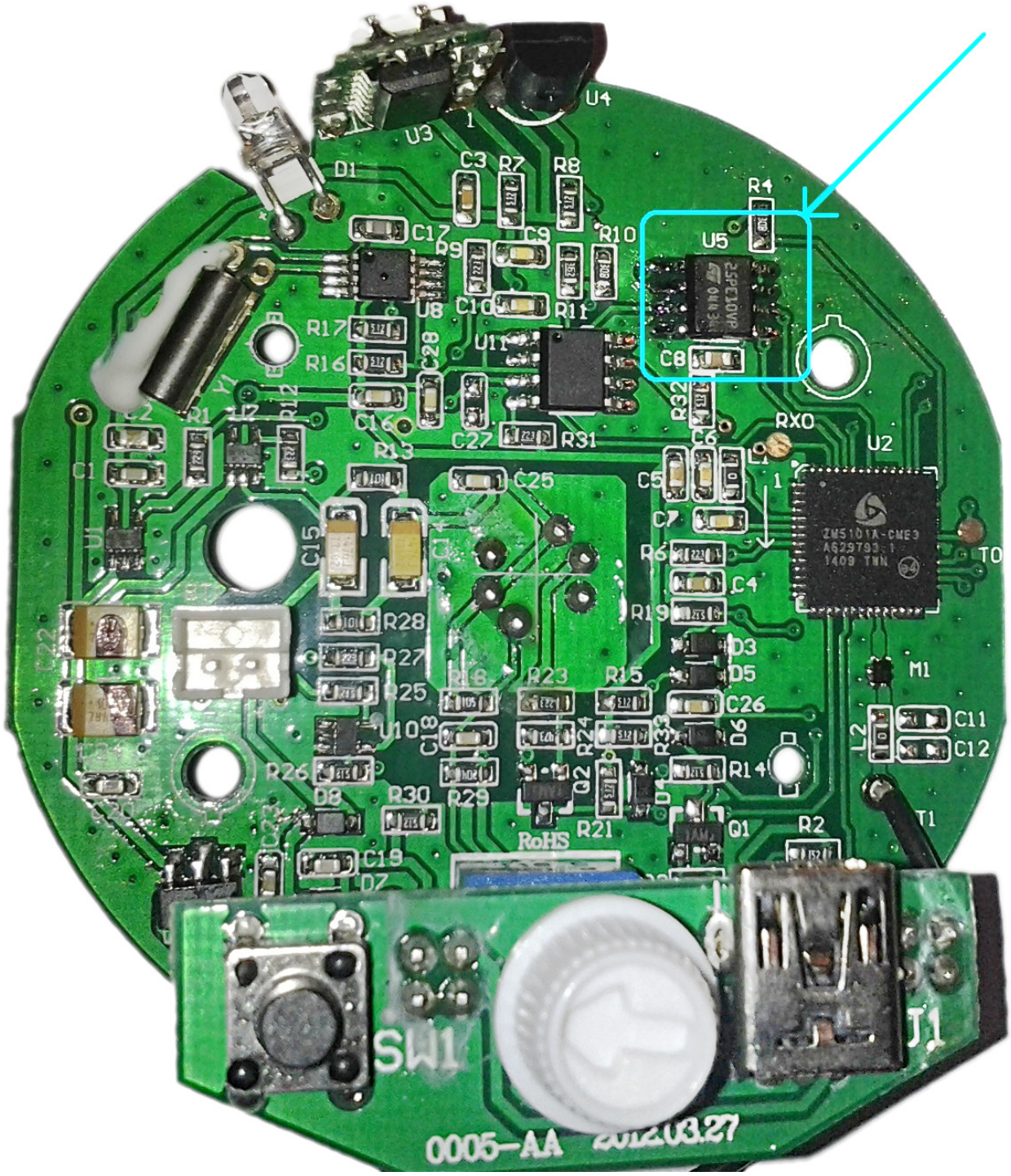



Figure 3 – Location of the EEPROM DIP on a 5th gen Z-Wave PIR sensor (Aeotec Multisensor 4)

```
1 6090: 00000000 00000000 00000000 ff000001
  60a0: e97a5631 cb5686fa 24450eba 103f945c
3 60b0: 56001498 eff17275 13cc4201 00000000
  60c0: 42326402 a8010000 00000000 00000000
```

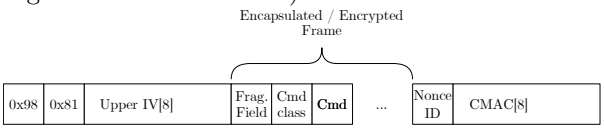
For reference, the segment of memory in Billy's door lock containing the network key follows. The network key starts at address `0x012D`.

```
  0110: 00000000 00000000 00000000 00000000
2 0120: 00000000 00420100 00000000 81e97a56
  0130: 31cb5686 fa24450e ba103f94 5c560000
4 0140: 00000000 00000000 00000000 00000000
```

To summarize the above, each device contains a network key, an authentication key, and an encryption key. The network key is common throughout the network and is shared with the devices by using default authentication and encryption keys that are the same for all 3rd and 5th generation Z-Wave devices in the world. The authentication and the encryption key on the device are derived from the network key and the nonces of all 5s and all As respectively.

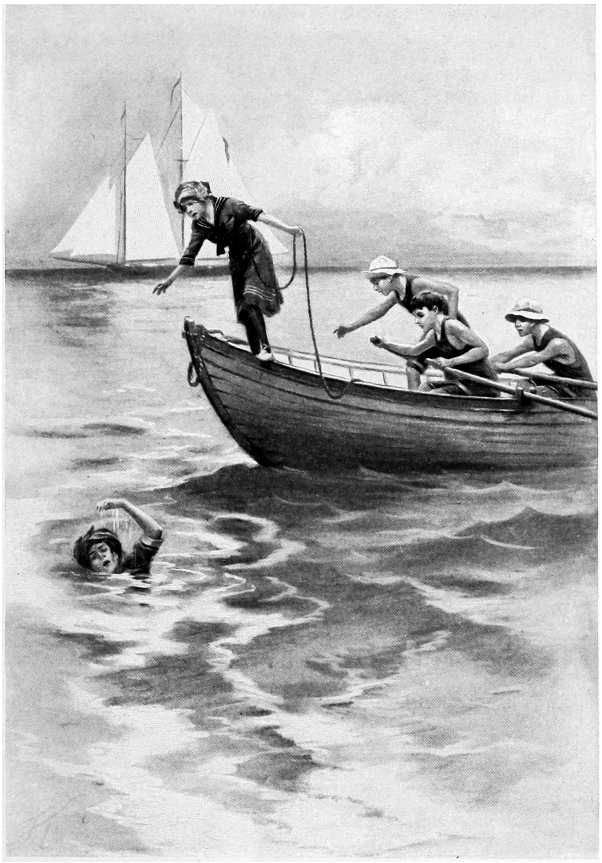
### 3.5 Do You Hear What I Hear? A Frame, a Frame, Encapsulated in a Frame, Is Encrypted

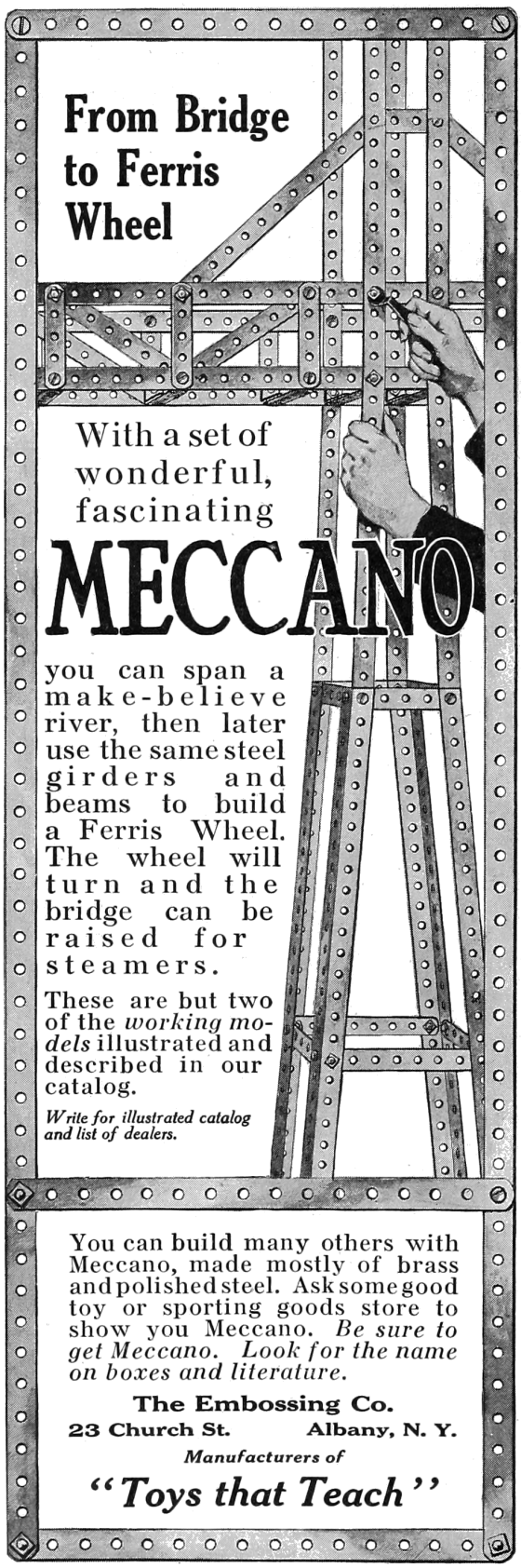
Even armed with the keys, the patient reader still needs to know how to use them. The Z-Wave security service provides immutable encryption and authentication through the use of an encapsulation frame. The encapsulation security frame (shown below) is identified in the first two bytes of the application layer payload. The first byte specifies the command class, and the second provides the command, where an encapsulated security frame has byte values of `0x98` and `0x81`, respectively. The remainder of the frame contains the eight upper bytes of the IV, used for both encryption and signing, the variable length encapsulated and encrypted payload, the nonce ID, and an 8-byte CMAC (cipher-based message authentication code).




At a minimum, the frame encapsulated in the security frame is three bytes. The first byte is used for fragmentation; however, we have yet to observe a value other than `0x00` in this field. The second byte provides the command class and, like the application layer, is followed by a single command byte and zero or more bytes of arguments.

The application payload is encrypted using the encryption key and an AES cipher in OFB mode with a 16-byte block size. OFB mode requires a 16-byte IV, which is established cooperatively between the source and destination. The lower 8 bytes of the IV are generated on request by the destination, which OpenZwave calls a nonce, and are reported to the requestor before the encapsulation frame is sent. The first byte of this 8-byte nonce is what we referred to as the nonce ID. The upper eight bytes of the IV are generated by the sender and included in the encapsulation security frame. When the destination receives the encapsulated frame, it decrypts the frame using the same cipher setting and key. It is able to reconstruct the IV using the IV field of the encapsulated frame and by using the nonce ID field to search its cache of generated nonces.






### 3.6 Joy to the Home, Encrypted Traffic is Revealed

Some cautious readers may become anxious when two automations are having a private conversation within their dwelling. This is especially true when one of them is a sensor, and the other is connected to the Internet. Fear not! Armed with knowledge of the encapsulation security frame and possession of the network or encryption key, the triumphant reader can readily decrypt frames formerly hidden from them. They will hopefully discover, as we have, that Z-Wave messages are devoid of sensitive user information. However, may the vigilant reader be a sentry to warn us if any future transgressions do occur in the name of commercialism and Orwellianism.

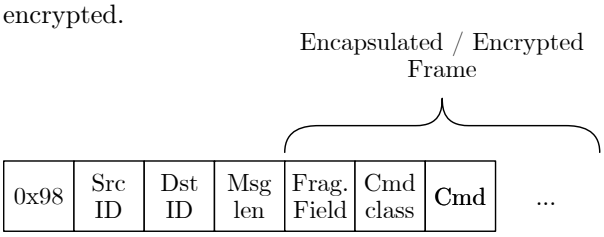
To aid the holy sentry, we provide the PoC `decryptPCAPNG` tool to decrypt Z-Wave encapsulated Z-Wave frames. The user provides the network or encryption key. The tool assumes the user is capturing Z-Wave frames using either Scapy-radio or EZ-Wave with an SDR, which sends observed frames to Wireshark for capture and saving to PCAPNG files.

### 3.7 What Frame Is This, Who Laid to Rest, upon Receiver's Antenna, Did Originate?

Secure Z-Wave devices do not act upon a command issued in an encapsulation frame unless its CMAC is validated. Thus, the active reader wishing to do more than observe encrypted messages requires further discourse. Certainly, the gremlin wishing to open Billy's front door desires the ability to generate an authenticated unlock-door command.

The Z-Wave CMAC is derived using the CBC-MAC algorithm, which encrypts a message using an AES cipher in CBC mode using a block size of 16 bytes. It uses the same IV as the encryption cipher, and only the first eight bytes of the resulting 16-byte digest are sent in the encapsulation frame to be used for authentication. Instead of creating the digest from the entire security encapsulation frame, a subset of fields are composed into a variable-length message. The first four bytes of this message are always the security command class ID, source ID, destination ID, and length of the message. The remaining portion of the message is the variable length

encapsulated frame (e.g., an unlock-door command, including the fragmentation byte) after it has been encrypted.




The recipient of the encapsulation security frame validates the integrity of the frame using the included 8-byte CMAC. It is able to generate its own CMAC by reconstructing the message to generate the digest using the available fields in the frame, the IV, and the authentication key. If the generated CMAC matches the declared value in the frame, then the source ID, destination ID, length, and content of the encapsulated frame are validated. Note that, since the other fields in the frame are not part of the CMAC message, they are not validated. If the generated digest does not match the CMAC in the frame, the frame is silently discarded.


## 3.8 Bring a Heavy Flamer of Sanctified Promethium, Jeanette, Isabella

Knock! Knock! Knock! Open the door for us!
Knock! Knock! Knock! Let's celebrate!

We wrote `OpenBarley` as a PoC tool to demonstrate how Z-Wave security works. Its default encapsulated command is to unlock a door lock, but the user may specify alternative, arbitrary commands. The tool works with the GNURadio Z-Wave transceiver available in Scapy-radio or EZ-Wave to inject authenticated and encrypted frames.

The reader must note that battery operated Z-Wave devices conserve power by minimizing the time the transceiver is active. When in low-power mode, a beam frame is required to bring the remote device into a state where it may receive the application layer frame and transmit an acknowledgment. Scapy-radio and EZ-Wave did not previously support waking devices with beam frames, so we have contributed the respective GNURadio Z-Wave blocks to EZ-Wave to allow this.

## 3.9 It Came! Somehow or Other, It Came Just the Same!

This Christmas, as we have done, may you, the blessed reader, extract the network key from the EEPROM of a Z-Wave device. May you use our PoCs to send authenticated commands to any other secured device on *your* network. May you enlighten your friends and neighbors, affording them the opportunity to sanctify by fire, or with lesser, more legal means, home automation lacking physical security in the name of Manion Butler and his holy mother. May you use our PoCs to watch the automation for privacy breaches and data mining in the time to come, and may you brew in peace.