# Power Analysis Attacks Against IEEE 802.15.4 Nodes

Colin O'Flynn(✉) and Zhizhang Chen

Dalhousie University, Halifax, Canada
{coflynn,zchen}@dal.ca

**Abstract.** IEEE 802.15.4 is a wireless standard used by a variety of higher-level protocols, including many used in the Internet of Things (IoT). A number of system on a chip (SoC) devices that combine a radio transceiver with a microcontroller are available for use in IEEE 802.15.4 networks. IEEE 802.15.4 supports the use of AES-CCM* for encryption and authentication of messages, and a SoC normally includes an AES accelerator for this purpose. This work measures the leakage characteristics of the AES accelerator on the Atmel ATMega128RFA1, and then demonstrates how this allows recovery of the encryption key from nodes running an IEEE 802.15.4 stack. While this work demonstrates the attack on a specific SoC, the results are also applicable to similar wireless nodes and to protocols built on top of IEEE 802.15.4.

**Keywords:** AES · Side-channel power analysis · DPA · IEEE 802.15.4

## 1 Introduction

IEEE 802.15.4 is a low-power wireless standard which targets Internet of Things (IoT) or wireless sensor network (WSN) applications. Many protocols use IEEE 802.15.4 as a lower layer, including ZigBee (which encompasses many different protocols such as ZigBee IP and ZigBee Pro), WirelessHART, MiWi, ISA100.11a, 6LoWPAN, Nest Weave, JenNet, IEEE 802.15.5, Thread, Atmel Lightweight Mesh, and DigiMesh. As part of the IEEE 802.15.4 standard a security suite based on AES is included, which allows encrypting and adding an authentication code on the wireless messages.

Protocols using IEEE 802.15.4 as a lower layer often include security at layers above IEEE 802.15.4, but many of them also use the same AES primitive as the lower layer (with a different key and possibly encryption mode). An attack against the AES peripheral in an embedded device may be useful in attacking both the lower and higher layers depending on network specifics. Even if acquiring the 802.15.4-layer key is not directly useful, because for example each link uses a different key, an attacker may practically benefit from the ability of sending arbitrary messages which will be accepted as valid and passed to the higher-layer protocol decoder logic. With this ability an attacker can exploit security flaws in higher-layer protocol decoding logic, since the lower-layer messages will be successfully decrypted and presented to higher layers.


F.-X. Standaert and E. Oswald (Eds.): COSADE 2016, LNCS 9689, pp. 55–70, 2016.
DOI: 10.1007/978-3-319-43283-0_4

This paper presents an attack against a wireless node that uses the IEEE 802.15.4 protocol. We present the following important results from developing this attack: (1) an attack against the hardware AES engine in the Atmel ATMega128RFA1, (2) an attack on AES-128 in CCM* mode as used in IEEE 802.15.4 [1], (3) a method of causing the AES engine in the target device to perform the desired encryption, and (4) a shunt-based measurement method for devices with internal voltage regulators. This attack is validated with a hardware environment (shown in Fig. 1).

The attack demonstrated here uses side-channel power analysis [2], specifically a correlation-based attack [3]. We obtained the power measurements in this work by physically capturing a node and inserting a shunt resistor. In general, side-channel attacks can be performed with a noncontact electromagnetic (EM) probe instead, which does not require modification to the device [4]. The EM measurement typically achieves similar results to the resistive shunt [5,6].

It has previously been demonstrated that wireless nodes are vulnerable to side-channel power analysis when running AES-ECB in software [7]. This type of attack does not destroy the node under attack, and the node will continue to function during the attack. This makes detection more difficult: although a node is captured, it still appears on the network. Our work extends this by attacking the actual AES-CCM* mode used in IEEE 802.15.4, attacking the hardware AES accelerators typically used in wireless stack implementations, and demonstrating how to force many encryption operations to occur for rapid collection of traces.

We begin by describing the attack on the ATMega128RFA1 AES hardware peripheral in Sect. 2. Next, we look at specifics of the use of AES encryption on the IEEE 802.15.4 wireless protocol in Sect. 3. This outlines the challenges of applying the side-channel attack to the AES-CCM* mode of operation, which is solved for the case of IEEE 802.15.4 in Sect. 4. Our application of this to a real IEEE 802.15.4 node is discussed in Sect. 5, and our conclusions follow.

An extended version of this paper is available which contains additional details and discussion of this attack[1].

## 2 ATMega128RFA1 Attack

The Atmel ATMega128RFA1 is a low-power 8-bit microcontroller with an integrated IEEE 802.15.4 radio, designed as a single-chip solution for Internet of Things (IoT) or wireless sensor network (WSN) applications [8]. As part of the IEEE 802.15.4 radio module a hardware AES-128 block is available, designed to work with the AES security specification of IEEE 802.15.4. Other vendors such as Freescale (MC13233), Silicon Laboratories (EM35x), STMicroelectronics (STM32W108), and Texas Instruments (CC2530) provide similar chips integrating an IEEE 802.15.4 radio and microcontroller in a single device.

To perform a side-channel power analysis attack, we evaluate a method of physically measuring power on the ATMega128RFA1 in Sect. 2.1. We then determine an appropriate power model in Sect. 2.2, and we present the results of the

[1] The extended version is published at https://eprint.iacr.org/2015/529.

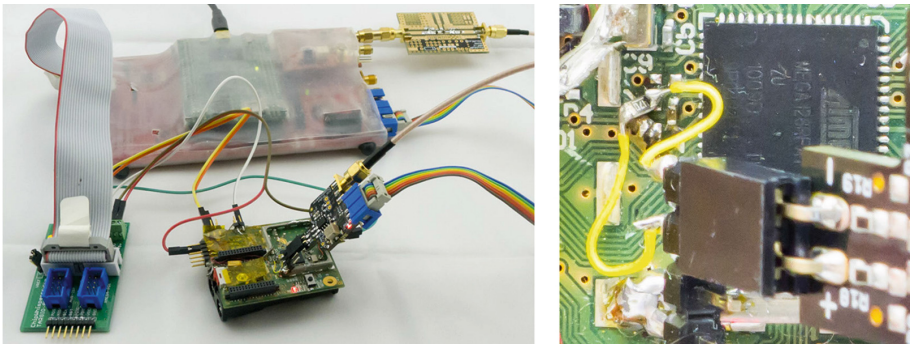

**Fig. 1.** The ChipWhisperer capture hardware is used in this attack, along with details of the measurement point.

CPA attack [3] in Sect. 2.3. We present additional considerations for attacking intermediate rounds (i.e., beyond the first round) of the AES algorithm in Sect. 2.4; these intermediate-round attacks are required for the AES-CCM* attack.

### 2.1 Power Measurement

Power measurements can be performed by inserting a resistive shunt into the power supply of the target device, and measuring the voltage drop across the shunt. Because devices often have multiple power supplies (such as $VCC_{core}$, $VCC_{IO}$, $VCC_{RF}$), the shunt must be inserted into the power supply powering the cryptographic core. As with many similar IEEE 802.15.4 chips, the core voltage of the ATMega128RFA1 is lower (1.8 V) than the IO voltage (typically 2.8–3.3 V) [8].

To avoid requiring an external voltage regulator for the lower core voltage, most of these devices also contain an integrated 1.8 V voltage regulator. Some devices require an external connection from the regulator output pin to the $VCC_{core}$ pin. With this type of device we could perform the power measurements by either (a) inserting a shunt resistor between the output and input, or (b) using an external low-noise power supply with a shunt resistor (as in [7]). The ATMega128RFA1 is not such a device – it internally connects the regulator to the $VCC_{core}$ pin, but does require a decoupling capacitor placed on the $VCC_{core}$ pin (which also serves as the output capacitor for the voltage regulator).

By inserting a shunt resistor into the path of the decoupling capacitor, we can measure high-frequency current flowing into the $VCC_{core}$ pin. Note that this measurement will be fairly noisy, as we will also have noise from current flowing out of the voltage regulator. The right side of Fig. 1 shows the implementation of this arrangement. Externally powering this pin with a voltage slightly higher than 1.8 V may disable the internal regulator, giving a lower-noise signal from the shunt resistor. This is dependent on regulator design.

### 2.2 Related Hardware Attack

We based our work on Kizhvatov's attack on the XMEGA device [9]. Kizhvatov determined that for a CPA attack on the XMEGA, the Hamming distance between successive S-box input values leaked. These input values are the XOR of the plaintext with the secret key that occurs during the first `AddRoundKey`.

Our notation considers $p_i$ and $k_i$ to be a byte of the plaintext and encryption key respectively, where $0 \leq i \leq 15$. To determine an unknown byte $k_i$, we first assume we know a priori the value of $p_i$, $p_{i-1}$, and $k_{i-1}$.

This allows us to perform a standard CPA attack, where the sensitive value is given by the Hamming weight of (1). That is to say the leakage for unknown encryption key byte $i$ is: $l_i = HW(b_i)$. Provided $k_0$ is known, this attack can proceed as a standard CPA attack, with only $2^8$ guesses required to determine each byte.

$$b_i = (p_{i-1} \oplus k_{i-1}) \oplus (p_i \oplus k_i), \quad 1 \leq i \leq 15 \tag{1}$$

For the specific case of $k_0$, the Hamming distance from the fixed value `0x00` is used as a leakage model[2], as in (2). This allows the entire encryption key to be attacked with a total of $16 \times 2^8$ guesses.

$$l_0 = HW(b_0) = HW(p_0 \oplus k_0) \tag{2}$$

### 2.3 Application to ATMega128RFA1

Our experimental platform was a Dresden Elektronik radio board, model number RCB128RFA1 V6.3.1. To sample the power measurements, we used an open-source platform called the ChipWhisperer Capture Rev2 [10]. This capture hardware synchronizes its sampling clock to the device clock, and we configured it to sample at 64 MS/s (which is 4 times the ATMega128RFA1 clock frequency of 16 MHz). The differential probe is connected across a shunt in the $VCC_{core}$ power pin as described previously. A filter with a passband of 3–14 MHz was inserted between the output of the differential probe and the low-noise amplifier input of the ChipWhisperer.

We implemented a test program in the ATMega128RFA1 that encrypts data received over the serial port. This encryption can be done via either a software AES-128 implementation or the hardware AES-128 peripheral in the ATMega128RFA1. When using the hardware peripheral, the encryption takes 25 µs to complete, or about 400 clock cycles.

We used a CPA attack, ranking the most likely byte as the one with the highest correlation values [3]. We use a plot of the partial guessing entropy (PGE) compared to number of traces in order to measure attack success [11]. The PGE indicates where the correct value of the encryption subkey byte falls within a list ordered from most to least likely based on CPA attack results.

[2] This is not published in [9], but was described in private communication from the author.

Thus when the PGE falls to zero the specific subkey byte is perfectly known, and a PGE of 128 would be expected for a completely unsuccessful attack that is equivalent to a random guess.

To evaluate our measurement toolchain, we performed this attack against a software AES implementation on the ATMega128RFA1, which recovered the complete key in under 60 traces.

We then recorded a total of 50 000 power traces, where the ATMega128RFA1 was performing AES-128 ECB encryptions using random input data during the time each power trace was recorded. For each trace, 600 data points were recorded at a sampling rate[3] of 64 MS/s. Each trace therefore covered about the first third of the AES encryption.

Our initial CPA attack was repeated five times over groups of 10 000 traces. The resulting average partial guessing entropy for each byte is shown in Fig. 2. The first byte (which uses the leakage assumption of (2)) has the worst performance, as the guessing entropy does not reach zero with 10 000 traces.

**Guessing of $k_{i-1}$.** This attack used the leakage (2) of the first byte $i = 0$ to bootstrap the key recovery. Once we know this byte, we can use (1) to recover successive bytes.

Practically, we may have a situation where $i-1$ is not recoverable. Previous work assumed either some additional correlation peak allowing us to determine $i-1$, or the use of a brute-force search across all possibilities of the byte $i-1$ [9]. We can improve on this with a more efficient search algorithm, described next.

The leakage function (1) could be rewritten to show more clearly that the leaked value depends not on the byte values, but on the XOR between the two successive bytes, as in (3).

$$b_i = (k_{i-1} \oplus k_i) \oplus (p_{i-1} \oplus p_i), \quad 1 \leq i \leq 15 \tag{3}$$

The side-channel attack can be performed with the unknown byte $k_{i-1}$ set to `0x00`, and the remaining bytes are recovered by the CPA attack described previously. These recovered bytes are not the correct value, but instead provide the value that has to be XOR'd with the previous byte to generate the correct byte.

The 256 candidate keys can then be generated with almost no computational work, by iterating through each possibility for the unknown byte $k_{i-1}$, and using the XOR values recovered from the CPA attack to generate the remaining byte values $k_i, k_{i+1}, \cdots, k_I$.

This assumes we are able to directly test those candidate keys to determine which is the correct value. As is described in the next section, we can instead use a CPA attack on the next-round key to determine the correct value of $k_{i-1}$.

[3] Note that this 64 MS/s sample rate is successful because the capture hardware samples synchronously with the device clock. If using a regular oscilloscope with an asynchronous timebase we expect a much higher sample rate to be required, similar to that reported in the XMEGA attack.

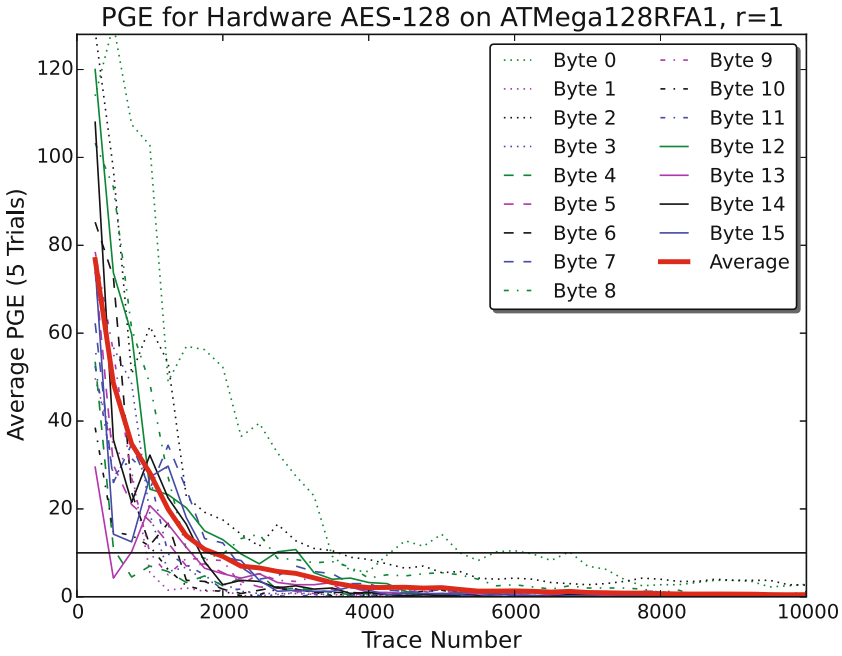


**Fig. 2.** The CPA attack on the hardware AES peripheral reduces the guessing entropy to reasonable levels in under 5000 traces, and is makes key recovery trivial in 10 000 traces. (Color figure online)

### 2.4 Intermediate-Round Attacks

Whereas our work so far has been concerned with determining the first-round encryption key, we will see in Sect. 4 that information on the round keys used during intermediate rounds is also required.

We determined that for intermediate rounds the leakage assumption of (1) and (2) still holds, where the unknown byte $k_i$ is a byte of the round key, and the known plain-text byte $p_i$ is the output of the previous round. We can extend our notation such that the leakage from round $r$ becomes $l_i^r = HW(b_i^r)$, where each byte of the round key is $k_i^r$, and the input data to that round is $p_i^r$.

Examples of the PGE when attacking the start of the third round ($r = 3$) are given in Fig. 3. The entropy change for all rounds tested ($r = 1, 2, 3, 4$) was similar.

For details of the execution time of the hardware AES implementation, refer to Table 1. This table shows the samples used for each byte in determining the most likely encryption key for the first four rounds. For byte 0 (the first byte), (2) is the sensitive operation. For later bytes (1) is the sensitive operation.

Note the sample rate is four times the device clock, and in Table 1 the sample delta from start to end of the sensitive operations within each round is about 64 samples, or 16 device clock cycles. This suggests that a sensitive operation

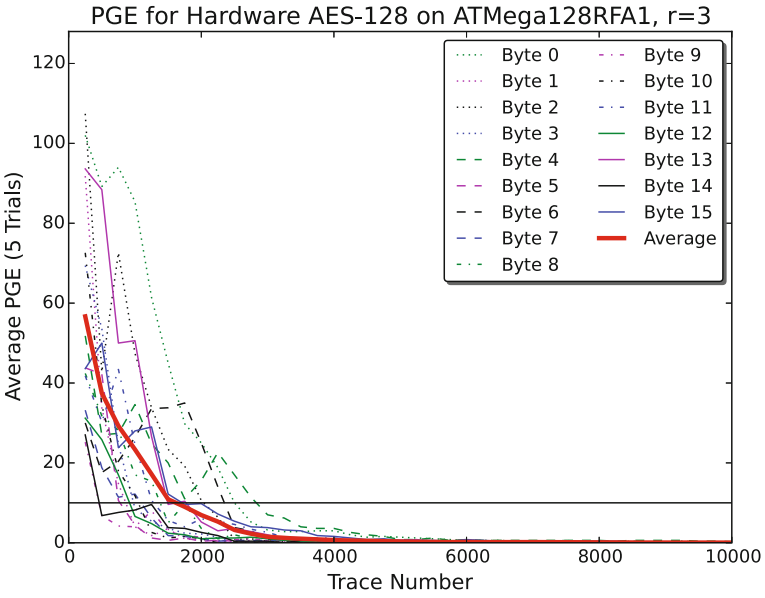


**Fig. 3.** Attacking intermediate rounds in the AES peripheral is also successful using the same leakage assumptions as the first-round attack. (Color figure online)

is occurring on each clock cycle. Each round takes approximately 32–34 cycles based on the repeating nature of the leakages in intermediate rounds.

**Determining $k_{i-1}$ Using Intermediate Rounds.** As described in Sect. 2.3, we can perform the CPA attack on byte $k_i$ where $k_{i-1}$ is unknown by determining not the value of the byte, but the XOR of each successive byte with the previous key. This means performing the attack first where $k_{i-1}$ is assumed to be `0x00`.

By then enumerating all 256 possibilities for $k_{i-1}$, we can quickly generate 256 candidate keys to test. But if we are unable to test those keys, we need another way of validating the most likely value of $k_{i-1}$.

If we knew the initial (first-round) key, we could determine the input to the second round, and thus perform a CPA attack on the second-round key. Instead we have 256 candidates for the first round ($r = 1$), and want to determine which of those keys is correct before proceeding.

To determine which of the keys is correct, we can perform a CPA attack on the first byte of the second round, $k_0^2$, repeating the CPA attack 256 times, once for each candidate first-round key.

The correlation output of the CPA attack will be low for all guesses of $k_0^2$ where $\mathbf{k^1}$ is wrong, and only for the correct guess of $k_0^2$ and $\mathbf{k^1}$ will there be a peak. This technique will be used in Sect. 4.1, where we cannot test candidate keys as we are not recovering the complete key.

**Table 1.** A small range of points is selected from each trace, corresponding to the location of the device performing (2) for $i = 0$, or (1) for $i \geq 1$. The variable $r$ corresponds to the AES round being attacked, and $i$ is the byte number.

| $i$ | $r = 1$ | $r = 2$ | $r = 3$ | $r = 4$ | $i$ | $r = 1$ | $r = 2$ | $r = 3$ | $r = 4$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 66–70 | 198–204 | 336–342 | 474–478 | 8 | 98–102 | 233–237 | 370–374 | 506–508 |
| 1 | 70–75 | 205–210 | 340–345 | 478–481 | 9 | 101–106 | 237–241 | 373–377 | 510–513 |
| 2 | 73–78 | 208–215 | 345–348 | 482–489 | 10 | 106–111 | 240–247 | 378–383 | 514–519 |
| 3 | 79–83 | 213–216 | 350–355 | 486–490 | 11 | 110–114 | 245–250 | 382–385 | 518–521 |
| 4 | 81–88 | 218–221 | 355–368 | 490–494 | 12 | 114–119 | 248–254 | 385–390 | 522–524 |
| 5 | 85–90 | 220–225 | 358–361 | 494–498 | 13 | 118–123 | 253–258 | 390–394 | 525–529 |
| 6 | 89–95 | 225–233 | 362–365 | 498–501 | 14 | 121–126 | 258–265 | 394–398 | 530–534 |
| 7 | 93–98 | 230–235 | 366–370 | 502–505 | 15 | 126–129 | 262–268 | 398–402 | 534–538 |

## 3 IEEE 802.15.4 Security

IEEE 802.15.4 is a low-power wireless standard, sending short data packets of up to 127 bytes at bit-rate of 250 kbit/s. The IEEE 802.15.4 standard uses AES-128 as the basic building block for both encryption and authentication of messages. The standard defines a mode of operation called CCM*, which modifies the regular CCM mode by allowing the use of encryption without authentication [1,12].

The underlying encryption uses AES-CTR mode, with an input format as shown in Fig. 4. The first 14 bytes are the nonce, and the last two bytes are the AES-CTR mode counter. Each received frame must use a new nonce, as the counter only counts the number of 16-byte blocks within the frame.

To ensure nonce freshness, a field called `FrameCounter` is included with each transmitted message and used as part of the nonce. The receiver verifies that the value of `FrameCounter` is larger than any previously used value, avoiding the reuse of a nonce.

On receiving a packet, the IEEE 802.15.4 layer first returns an acknowledgment to the sender. If the packet has security enabled (it is encrypted or has an authentication code appended) the node performs the following steps: (1) validates headers, (2) check the new received frame counter is numerically greater

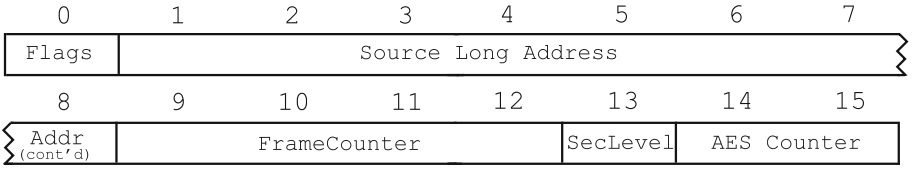


**Fig. 4.** The following data is used as the input to AES-128 when a frame is decrypted by an IEEE 802.15.4 stack. The `FrameCounter` can be controlled by the attacker.

than the last valid frame count, (3) looks up the secret key based on addressing, (4) decrypts the payload and authentication code (if present), (5) validates the authentication code (if present), and (6) stores the frame counter.

For our side-channel attack we only care that step 4 is performed; this means our packet must successfully pass through steps 1–3. This requires that the packet is properly addressed and has an acceptable security configuration, i.e. using a valid key identifier and address. An example of such a packet is available in the extended version of this paper.

## 4 Application to AES-CCM* Mode

For a standard CPA attack, we require the ability to cause a number of encryption operations to occur with known plaintext or ciphertext material. In addition, the data being encrypted must vary between operations, as otherwise each trace will generate the same hypothetical intermediate values during the search operation of the CPA attack.

From Sect. 3 and Fig. 4, we know that a number of the bytes are fixed during the AES encryption operation. Practically *all* the bytes except for the `FrameCounter` are considered fixed in this attack. The `Flags` and `SecLevel` bytes will have constant (and known) values. Initially it would appear that the `Source Long Address` and `AES Counter` fields may vary, but as we discuss next, this is not the case.

The `Source Long Address` field comes from internal tables in the 802.15.4 stack, and is not simply copied from the incoming packet fields. The `AES Counter` field changes during operation, as it increases for each 16-byte block encrypted in AES-CCM* mode. But as the IEEE 802.15.4 packet is limited to a total of 127 bytes, the `AES Counter` field could never exceed `0x0007`. Thus, between these 10 bytes, at most 3 bits vary during operation.

We instead rely on the ability of the attacker to control the `FrameCounter` field to mount a successful attack on an IEEE 802.15.4 wireless node. For our work we will assume an attack on the first encryption operation when a packet is received, meaning the `AESCounter` field is also fixed. The sent value of `FrameCounter` must simply be higher than a previously accepted value, which can either be determined by passive listening, or the most significant bit(s) can simply be set high to guarantee values which are likely to be accepted.

### 4.1 Previous AES-CTR Attacks

The AES-CCM* mode used by IEEE 802.15.4 is a combination of CBC-MAC and CTR modes of operation. Our attack is on the AES-CTR portion of the algorithm, with some modifications to reflect the use of a frame counter for the nonce material.

Previous work on AES-CTR mode has focused on the assumption that we can cause a number of encryptions to occur in sequence (i.e., with increasing counter number), but with unknown but constant nonce material [13]. Our work

uses many of the constructs developed by Jaffe in [13], but with different assumptions of inputs on the AES block and a different leakage model. These differences necessitate the development of new techniques to recover partial keying information, as we cannot directly apply the previously published attack.

In our case, we have the ability to change 4 bytes of the input plaintext (bytes 9, 10, 11, and 12). The CPA attack only allows us to recover these four bytes of the key, as the keying material associated with bytes 9–12 can be recovered by a standard CPA attack using the leakage model identified in Sect. 2. The remaining bytes cannot be recovered, as the input data is constant, and hence our leakage target of the difference between S-Box inputs is also constant.

For the $MixColumns()$ operation, we can represent the four input bytes – one column of the state matrix – with $s_0, \cdots, s_3$, and the resulting output bytes with $S_0, \cdots, S_3$. The $MixColumns()$ operation uses multiplication over the Galois field GF($2^8$), where we represent this multiplication operation with the symbol "$\circ$". The $MixColumns()$ operation then becomes:

$$S_0 = (2 \circ s_0) \oplus (3 \circ s_1) \oplus s_2 \oplus s_3 \tag{4}$$

$$S_1 = s_0 \oplus (2 \circ s_1) \oplus (3 \circ s_2) \oplus s_3 \tag{5}$$

$$S_2 = s_0 \oplus s_1 \oplus (2 \circ s_2) \oplus (3 \circ s_3) \tag{6}$$

$$S_3 = (3 \circ s_0) \oplus s_1 \oplus s_2 \oplus (2 \circ s_3) \tag{7}$$

Using the method from [13], we use our partial knowledge of the current round key to recover information about the next round key. Performing the attack with partial knowledge is possible as if some of the input bytes to $MixColumns()$ are fixed but unknown, we set those fixed bytes to 0, and use the linear property of $MixColumns()$ to introduce a correction constant. Assuming the true output of one $MixColumns()$ is $S_0$, we define the output that results by setting constant bytes to 0 as $S'_0 = S_0 \oplus E_0$, where $E_0$ is an unknown correction constant.

Performing the CPA attack using the assumed output $S'_0$, we would recover a version of this round key byte (we will refer to it as $k'_0$) XOR'd with the unknown constant $E_0$, that is $k'_0 = k_0 \oplus E_0$. The output of $AddRoundKey()$ will be equivalent to the case where we had the true key and true input, as:

$$AddRoundKey(k'_0, S'_0) = k'_0 \oplus S'_0 = (k_0 \oplus E_0) \oplus (S_0 \oplus E_0) = k_0 \oplus S_0 \tag{8}$$

This is sufficient information to perform the attack on the next round of the AES algorithm. Thus, if the entire modified version of a key can be recovered for a given encryption round, we can recover the entire *unmodified* key by attacking the next encryption round. This unmodified key can then be rolled backwards using the AES key schedule.

**Description of Attack.** We describe the attack by working through a symbolic example, using the following variables:

$p_i^r$ : "text" input to $AddRoundKey()$
$k_i^r$ : "key"input to $AddRoundKey()$
$E_i^r$ : a constant, see Sect. 4.1
$n_i^r$ : the modified round key, $k_i^r \oplus E_i^r$
$s_i^r$ : the output of $SubBytes()$
$v_i^r$ : the output of $ShiftRows()$
$m_i^r$ : the output of $MixColumns()$
X : variable and known inputs
Y : variable and known intermediates
Z : variable and known intermediates
c : constant values
? : variable and unknown values
N : known modified round-key values ($n_i^r$)
K : known key or round-key values ($k_i^r$)
X* : group of variables which has a small set of candidates for the correct value

Initially, we have the known input plaintext, where 12 of the bytes are constant, and the 4 variable bytes are under attacker control (`FrameCounter`). From this, we can perform a CPA attack to recover 4 bytes of the key. Note that in practice the byte $k_9^1$ cannot be recovered because $k_8^1$ is unknown. Instead we use the technique detailed in Sect. 2.4 to generate 256 candidate keys for $k_9^1, \cdots, k_{12}^1$, and test them at a later step. This means we can assume the following is the state of our initial-round key:

$\mathbf{k^1}$ = `[c c c c c c c c K*K*K*K* c c c]`

This can be used to calculate the output of the $SubBytes()$ and $ShiftRows()$ functions, where the majority of bytes are constant (but unknown):

$\mathbf{s^1}$ = `[c c c c c c c c Y*Y*Y*Y* c c c ]`
$\mathbf{v^1}$ = `[c c Y* c c Y* c c c c c c Y* c c Y*]`

At this point we need to symbolically deal with the $MixColumns(\mathbf{v^1})$ output, as we will be working with the modified output that has been XOR'd with the constant $E$. As in [13], this is accomplished in practice by setting unknown constants `c` to zero, and calculating the output of the $MixColumns(\mathbf{v^1})$ function. The unknown constants are all pushed into the variable $E$, which we never need to determine the true value of. This means our output of round $r = 1$ becomes:

$\mathbf{m^1}$ = `[Z*Z*Z*Z*Z*Z*Z*Z* c c c c Z*Z*Z*Z*]`

Note that 4 bytes of this output are constant. We again set these constant bytes to zero to simplify our further manipulation of them. This means our input to the next round becomes:

$\mathbf{p^2}$ = `[Z*Z*Z*Z*Z*Z*Z*Z* 0 0 0 0 Z*Z*Z*Z*]`

We are not able to recover $n_8^2, \cdots, n_{11}^2$ yet, as the inputs associated with those key bytes are constant.

We first attempt to recover $n_0^2$, which is performed for all 256 candidates for $k_9^1, \cdots, k_{12}^1$. As mentioned in Sect. 2.4, the highest correlation peak determines both $k_9^1, \cdots, k_{12}^1$ and $n_0^2$. This means we no longer have a group of candidates for the input, but a single value:

$\mathbf{p^2}$= `[Z Z Z Z Z Z Z Z 0 0 0 0 Z Z Z Z]`

We can then proceed with the CPA attack on the remaining bytes of $\mathbf{n^2}$. Bytes $n_1^2, \cdots, n_6^2$ can be recovered by application of the CPA attack from Sect. 2.3.

Recovery of $n_7^2$ using the same process is not possible, as $MixColumns(\mathbf{v^1})$ interacts with the leakage model. The inputs to this round $p_6^2$ and $p_7^2$, are generated by the previous-round $MixColumns(\mathbf{v^1})$ outputs $m_6^1$ and $m_7^1$.

When attacking $n_7^2$, we apply (1) to (6) and (7). This means our leakage is:

$$HW\left((n_6^2 \oplus (6)) \oplus (n_7^2 \oplus (7))\right) \tag{9}$$

The XOR cancels common terms in (6) and (7), and in this case that cancels term $s_1$. As $s_1$ is the variable and known input to the $MixColumns(\mathbf{v^1})$, the leakage appears constant and the attack fails. Instead, we can recover this value using a CPA attack on the next round, which is described later.

Returning to our CPA attack on the modified round key, we are unable to recover $n_8^2, \cdots, n_{11}^2$ as the associated inputs are constant. As $n_{11}^2$ is unknown, we cannot directly recover $n_{12}^2, \cdots, n_{15}^2$. Instead we again use the method of Sect. 2.4 to generate 256 candidates for $n_{12}^2, \cdots, n_{15}^2$.

At this point we assume the CPA attack has succeeded, meaning we have recovered the following bytes of the *modified* round key, where the final 4 bytes are partially known – we have 256 candidates for this group, as we know the relationship between each byte, but simply don't know the starting byte to define the group:

$\mathbf{n^2}$ = `[N N N N N N N c c c c c N*N*N*N*]`

Remember, once we apply $AddRoundKey(\mathbf{n^2}, \mathbf{p^2})$, the constant $E$ will be removed – $E$ is included in both the output of $MixColumns(\mathbf{v^1})$ and the modified key – meaning we can determine the true value of the input to $SubBytes()$.

The outputs $8, \cdots, 11$ of $MixColumns(\mathbf{v^1})$ from the first round are constant, so we also know the four unknown modified bytes $n_8^2, \cdots, n_{11}^2$ can be ignored at this point. The result of $AddRoundKey(\mathbf{n^2}, \mathbf{p^2})$ for these bytes will be another constant.

The unknown byte $n_7^2$ is associated with variable input data, meaning this output will be unknown and variable, which cannot be ignored. At this point we can represent the known outputs of $SubBytes()$ and $ShiftRows()$:

$\mathbf{s^2}$ = `[Y Y Y Y Y Y Y ? c c c c Y* Y*Y*Y*]`

$\mathbf{v^2}$ = `[Y Y c Y* Y c Y* Y c Y* Y ? Y* Y Y c]`

As before, we can set unknown constant values to zero to determine the modified output $\mathbf{m^2} = MixColumns(\mathbf{v^2})$. The unknown variable byte means 4 bytes of the $MixColumns(\mathbf{v^2})$ output are currently unknown. In addition, we have 256 candidates for the remaining known values, since the four modified bytes $n_{12}^2, \cdots, n_{15}^2$ have been mixed into all output bytes by $ShiftRows(\mathbf{p^2})$ and $MixColumns(\mathbf{v^2})$:

$\mathbf{m^2}$ = `[Z*Z*Z*Z*Z*Z*Z*Z* ? ? ? ? Z*Z*Z*Z]`

This becomes the input to the next round:

$\mathbf{p^3}$ = `[Z*Z*Z*Z*Z*Z*Z*Z* ? ? ? ? Z*Z*Z*Z]`

We again apply the CPA attack on $n_0^3$ across all values for $n_0^3$ and the 256 candidates for the previous modified round key (a total of $2^{16}$ guesses), the peak telling us the value of $n_0^3$ and $n_{12}^2, \cdots, n_{15}^2$. We now know which of the candidates to select for further processing:

$\mathbf{p^3}$= [Z Z Z Z Z Z Z Z ? ? ? ? Z Z Z Z]

We can apply a CPA attack to discover the modified key values $n_1^3, \cdots, n_7^3$. The unknown plaintext byte ? represents a changing value. We cannot ignore it as we can constant values in the $MixColumns(\mathbf{v^2})$, and thus cannot apply the CPA attack on the remaining bytes.

Instead we enumerate all possibilities for $n_7^2$, and apply a CPA attack against $n_8^3$, similarly to previously described attacks from Sect. 2.4. We verified experimentally that the correlation value with the highest peak for $n_8^3$ resulted only when $n_7^2$ was the correct value. This means we now have the entire modified output of $MixColumns(\mathbf{v^2})$, and thus the complete modified input plaintext to round 3:

$\mathbf{p^3}$ = [Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z]

With $n_7^2$ and $n_8^3$ now known, we can continue with the CPA attack against $n_9^3, \cdots, n_{15}^3$. At this point we have an entire modified key:

$\mathbf{n^3}$ = [N N N N N N N N N N N N N N N N]

We can again apply the modified key $\mathbf{n^3}$ to the modified output of the previous round $\mathbf{m^2}$ to recover the complete output of round $r = 3$, which will be the actual input to round $r = 4$. This allows us to perform a CPA attack and recover the true round key $\mathbf{k^4}$. This round key can then be rolled backwards using the AES key schedule to determine the original encryption key.

We have now attacked an AES-CCM* implementation as specified in the IEEE 802.15.4 standard. This attack requires only the control of the four bytes of `FrameCounter`, which are sent as plaintext over the air.

The computational load of the attack is minimal: performing these steps on an Intel i5-2540M laptop using a single thread program written in C++ takes under ten minutes with 20 000 traces, using only the subset of points in each trace from Table 1. Note when performing the hypothetical value calculation for intermediate rounds, the calculation was accelerated using the Intel AES-NI instruction set for performing the $SubBytes()$, $ShiftRows()$, and $MixColumns()$ operations, which form part of a single AES round executed by this instruction [14].

## 5 Attacking Wireless Nodes

In the previous sections, we demonstrated the vulnerability of an IEEE 802.15.4 SoC device to power analysis, and how the AES-CCM* mode used during reception of an encrypted IEEE 802.15.4 packet can be attacked when the underlying hardware is vulnerable to power analysis. The last two aspects of this attack

are to (1) demonstrate how we can trigger that encryption operation, and (2) determine where in the power signature the encryption occurred.

Details of the required packet format for reception are detailed in the extended version of this paper. The packet must simply conform to IEEE 802.15.4 requirements and have valid addressing information. The attacker controls the `FrameCounter` field as part of the attack.

In order for the side-channel attack to be successful, the attacker needs to determine *when* the AES encryption is occurring. As a starting point, the attacker can use information on when the frame should have been received by the target node. Practically, this would be either the attacker's transmitter node toggling an IO line when the packet goes over the air, or the attacker could use another node that also receives the transmitted messages to toggle an IO line.

To determine the reliability of such a trigger, we measured the time between the frame being received and the actual start of AES encryption on the target node. Over 100 transmitted frames the delay varied between 311 and 338 µs. The mean value of the delay was 325 µs (5200 clock cycles), with a standard deviation of 7 µs (112 clock cycles). The jitter in the delay is assumed to be from the software architecture, which uses an event queue process the frames. Solutions for aligning or resynchronizing power traces before applying power analysis is well known [15–18].

To test the ability of an attacker to realign captured power traces, we used a simple normalized cross-correlation algorithm [19] to match a feature across multiple power traces for realignment, performing a simple static alignment [20].

The selected feature was a window at 9.2–29.2 µs after the start of the AES encryption in one reference trace, meaning the matched feature extended slightly beyond the actual AES encryption. We confirmed that a high correlation peak was generated only for a single sample around the AES algorithm with many sample power traces. A threshold of 0.965 on the correlation output (determined empirically) was used; if a power trace had no correlation peak higher than this level, the trace was dropped.

Future work on this IEEE 802.15.4 attack can include applying more advanced preprocessing techniques (such as differential frequency analysis or principal component analysis). But such preprocessing techniques are not required to fundamentally prove that (a) the AES core is leaking, and (b) the AES operation has some unique signature allowing realignment to succeed.

## 6 Conclusions

The IEEE 802.15.4 wireless standard is a popular lower layer for many protocols being used in or marketed for the coming "Internet of Things" (see Sect. 1 for an enumeration of some of these). Such protocols often use the same underlying AES primitive as the IEEE 802.15.4 layer for security purposes.

This paper has demonstrated vulnerabilities in a real IEEE 802.15.4 wireless node. A successful attack against the AES peripheral present in the ATMega128RFA1 device was demonstrated. This attack was demonstrated

against AES-ECB; as electronic code book (ECB) is not the operating mode of AES used in the network, we extended a previous attack on AES-CTR mode [13] to work against the AES-CCM* mode used in IEEE 802.15.4. This demonstrated that it is possible to recover the encryption key of a wireless node using side-channel power attacks and valid IEEE 802.15.4 messages sent to the node.

An extended version of this conference paper with additional details of the attack is available at https://eprint.iacr.org/2015/529.

**Acknowledgments.** The authors would like to thank the anonymous reviewers at COSADE 2016 for their insightful comments. Colin O'Flynn is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the CGS program.

## References

1. IEEE: Standard 802.15.4-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) (2006)
2. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
5. Agrawal, D., Rao, J.R., Rohatgi, P.: Multi-channel attacks. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 2–16. Springer, Heidelberg (2003)
6. O'Flynn, C., Chen, Z.: A case study of side-channel analysis using decoupling capacitor power measurement with the OpenADC. In: Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Miri, A., Tawbi, N. (eds.) FPS 2012. LNCS, vol. 7743, pp. 341–356. Springer, Heidelberg (2013)
7. de Meulenaer, G., Standaert, F.-X.: Stealthy compromise of wireless sensor nodes with power analysis attacks. In: Chatzimisios, P., Verikoukis, C., Santamaría, I., Laddomada, M., Hoffmann, O. (eds.) MOBILIGHT 2010. LNICST, vol. 45, pp. 229–242. Springer, Heidelberg (2010)
8. Atmel Corporation: ATmega128RFA1 Datasheet (2014)
9. Kizhvatov, I.: Side channel analysis of AVR XMEGA crypto engine. In: Proceedings of the 4th Workshop on Embedded Systems Security, WESS 2009, pp. 8:1–8:7. ACM, New York (2009)
10. O'Flynn, C., Chen, Z.D.: ChipWhisperer: an open-source platform for hardware embedded security research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Heidelberg (2014)
11. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
12. Whiting, D., Ferguson, N., Housley, R.: Counter with CBC-MAC (CCM). https://tools.ietf.org/html/rfc3610

13. Jaffe, J.: A first-order DPA attack against AES in counter mode with unknown initial counter. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 1–13. Springer, Heidelberg (2007)
14. Gueron, S.: Intel Advanced Encryption Standard (AES) new instructions set. Whitepaper Doc. No. 323641-001 (2012)
15. Clavier, C., Coron, J.-S., Dabbous, N.: Differential power analysis in the presence of hardware countermeasures. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 252–263. Springer, Heidelberg (2000)
16. Gebotys, C.H., Ho, S., Tiu, C.C.: EM analysis of Rijndael and ECC on a wireless Java-based PDA. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 250–264. Springer, Heidelberg (2005)
17. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving differential power analysis by elastic alignment. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 104–119. Springer, Heidelberg (2011)
18. Batina, L., Hogenboom, J., van Woudenberg, J.G.J.: Getting more from PCA: first results of using principal component analysis for extensive power analysis. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 383–397. Springer, Heidelberg (2012)
19. Lewis, J.P.: Fast template matching. In: Canadian Conference on Vision Interface – VI 1995, pp. 120–123 (1995)
20. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, New York (2007)