Power Analysis Attacks against IEEE 802.15.4 Nodes

Colin O'Flynn and Zhizhang Chen

Dalhousie University,
{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, Thread, Atmel Lightweight Mesh, IEEE 802.15.5, and DigiMesh (this list only includes networking stacks that target commercial or industrial applications). As part of the IEEE 802.15.4 standard a security suite based on AES is included.

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) a shunt-based measurement method for devices with internal voltage regulators, (2) an attack against the hardware AES engine in the Atmel ATMega128RFA1, (3) an attack on AES-128 in CCM* mode as used in IEEE 802.15.4 [18], and (4) a method of causing the AES engine in the target device to perform the desired encryption. This attack is validated with a hardware environment (shown in Fig. 1).

The attack demonstrated here uses side-channel power analysis [21], specifically a correlation-based attack [5]. 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 [14]. The EM measurement typically achieves similar results to the resistive shunt [1,?].

This 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. The feasibility of capturing wireless nodes and performing side-channel power analysis has previously been demonstrated against AES and ECC [9].

This previous demonstration was limited to software implementations of AES (i.e., not the actual hardware AES used by most nodes), and did not attack the AES-CCM* operating mode used by IEEE 802.15.4. Instead the attack in [9] assumed the encrypted data packet transmitted by the node allowed recovery of the last-round state of the AES algorithm. This is not the case in AES-CCM* used by IEEE 802.15.4 and most higher-layer protocols: recovering the last-round state would require a plaintext and ciphertext pair.

In practical scenarios the ability to capture a node, perform the attack, and return the node all within a short window reduces the risk of detection. The approach of [9] requires an attacker to passively wait for a transmissions to record power traces. While passively waiting is a reasonable approach for the 20–60 traces required by [9] to break a software AES implementation, this could entail an unreasonably long wait period for the thousands of traces typically required to break a hardware AES peripherals [20]. Our work allows an attacker to rapidly force the operation to occur, and collecting 20 000 traces can be accomplished in 15–60 minutes (depends on network stack and how much other traffic node must process).

The work presented here results in a practical attack against IEEE 802.15.4 wireless nodes that recovers the encryption key in use by the IEEE 802.15.4 layer. In addition the attack is demonstrated against a hardware AES peripheral as used by a standards-complaint IEEE 802.15.4 stack. This work is applicable to protocols that use IEEE 802.15.4 as a lower layer, even if these higher-layer protocols include additional security. The higher layer often uses the same vulnerable AES primitive as the IEEE 802.15.4 layer. Users of these protocols must carefully evaluate how the vulnerabilities detailed in this paper might apply to the higher-layer protocols.

We begin by describing the attack on the ATMega128RFA1 AES hardware peripheral in Section 2. Next, we look at specifics of the use of AES encryption on the IEEE 802.15.4 wireless protocol in Section 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 Section 4. Our application of this to a real IEEE 802.15.4 node is discussed in Section 5, and our conclusions follow.

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

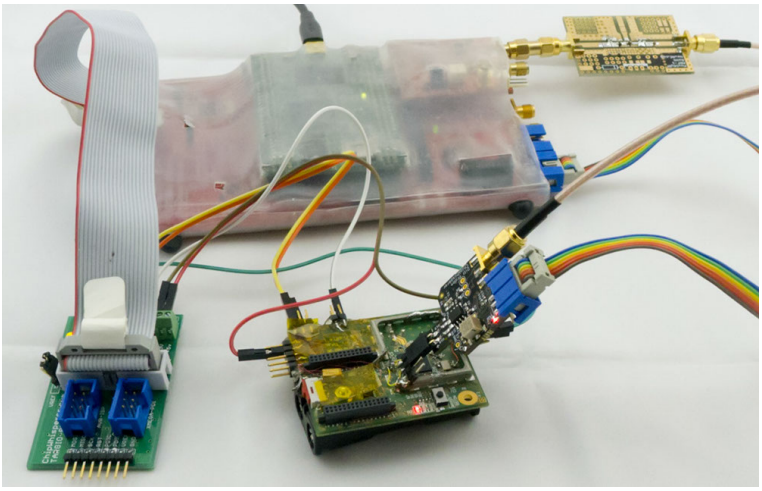

Fig. 1. The ChipWhisperer capture hardware is used in this attack, although a regular oscilloscope will also work.

Things (IoT) or wireless sensor network (WSN) applications [2]. As part of the IEEE 802.15.4 radio module, a hardware AES-128 block is present, designed to work with the AES security specification of IEEE 802.15.4. Other examples of such chips (chips that include an IEEE 802.15.4 radio, microcontroller, and AES block) include devices from Freescale [13], Silicon Laboratories [38], STMicroelectronics [41], and Texas Instruments [43].

Before detailing the specifics of the attack on the ATMega128RFA1, we present a brief background on side-channel power analysis.

2.1 Side-Channel Power Analysis

Side channel power analysis was first reported in 1998 by Kocher et al [21].

When performing a side-channel power analysis attack on AES-128, we will be attacking the 16-byte encryption key, denoted $\mathbf{k} = \{k_0, k_1, \cdots, k_{15}\}$. We assume the input plaintext value is known, the 16-byte plaintext being $\mathbf{p} = \{p_0, p_1, \cdots, p_{15}\}$. Finally, we also need to determine a "sensitive value" in the targeted operation.

For AES, one example of a sensitive value is the output of the S-box after a byte of the key has been XOR'd with a byte of the plaintext (i.e., $SBox(k_i \oplus p_i)$). We use a "leakage assumption" on the sensitive value, such as that the power consumption of the device depends on the Hamming weight (HW) of the sensitive value [5].

To perform the side-channel attack, we attack a single byte of the secret key, $\mathbf{k}$, at a time. We do this by enumerating all possibilities for the byte being attacked and calculating a hypothetical intermediate value along with a hypothetical leakage for each guess. Using a tool such as correlation power analysis

(CPA) [5], we can compare the measured power traces to our hypothetical power traces.

Hopefully, one of the hypothetical power traces will have a large correlation with the measured power used by the device. Knowing the input values to the leakage model that resulted in the power trace with the largest correlation gives us the most likely values the real system was operating on. This requires us to perform the comparison against a number of traces – if our model is very good and there is little system noise, it will take few traces for the output of the attack algorithm to determine the correct key. The CPA attack can break a software AES-128 implementation on an 8-bit microcontroller in fewer than 30 traces [31]. If our model is incorrect or the system noise is too high, the attack algorithm may not determine the correct key, even with millions of traces.

The CPA attack has parallels in other fields. Communications systems, for example, use a *matched filter* to match a received signal to one of several possible candidates (called symbols) [29]. In image processing, the *normalized cross-correlation* is used to match a feature of an image [11]. Both of these are mathematically equivalent to the CPA attack, demonstrating that the CPA attack is fundamentally an attempt to solve the same problem of matching a noisy signal with various possible candidates.

Side-channel power analysis has previously been used to break a variety of hardware devices. Table 1 summarizes published power analysis attacks against commercially available hardware cryptographic devices. This table does not include software implementations running on commercially available hardware or hardware implementations that are not commercial products (i.e., research projects).

Table 1. Power analysis attacks against commercially available *hardware* cryptographic implementations. Entries marked with † indicate firmware-based implementations, but still being commercially available.

Target	Cipher	Attack	Ref.
CryptoMemory	proprietary	CPA	[3]
DESFire MF3ICD40	3DES	CPA	[33]
DS2432, DS28E01	SHA-1	CPA	[32]
Microchip HCSXXX	KeeLoq	CPA	[12]
ProASIC3	AES	PEA	[39]
SimonsVoss†	proprietary	CPA	[35]
Spartan-6	AES	CPA	[26]
Stratix II	AES	CPA	[28]
Stratix III	AES	CPA	[42]
Virtex-II	3DES	CPA	[25]
Virtex-4, Virtex-5	AES	CPA	[27]
XMEGA	AES	CPA	[20]
Yubikey 2†	AES	CPA	[34]

From Table 1, it can be seen that the CPA attack is an extremely popular attack for targeting real systems. The only non-CPA attack in [39] was a new technique called pipeline emission analysis (PEA), used to break the ProASIC3 device. While each of the remaining attacks used CPA, additional work may be needed before applying the CPA attack. For example in the attack on the DESFire [33], a preprocessing technique realigned the traces before applying CPA.

There are more advanced attacks that use a *template* of the device leakage [6]. Such a template does not use a leakage model based on assumptions, but instead the leakage model is based on measurements of the target device as it performs known encryptions. This requires that the attacker has access to a device that closely matches the target device that they can manipulate or program.

While such template attacks are considerably more powerful – being able to recover the encryption key with less measurements – the CPA attack using a simple assumption is more versatile, since it only requires access to the single target device of interest. A device vulnerable to a CPA attack will always be vulnerable to a template attack, and it is almost certain that the template attack will improve the success rate further. For this reason, this work deals solely with the CPA attack, which also aligns this work with previous publications of successful CPA attacks against commercially available encryption hardware [12,20,25,26,27,28,33,34,32,35,42].

This paper uses the partial guessing entropy (PGE) to measure the attack success. PGE provides an indicator of the reduction in search space for each of the 16 key bytes. A PGE of zero for every key byte indicates that a full encryption key was recovered. Details of this metric are given in Appendix 7.

To perform this side-channel attack, we evaluate a method of physically measuring power on the ATMega128RFA1 in Section 2.2. We then determine an appropriate power model in Section 2.3, and we present the results of the CPA attack in Section 2.4. We present additional considerations for attacking later rounds of the AES algorithm in Section 2.5; these later-round attacks are required for the AES-CCM* attack.

2.2 Power Measurement

Power measurement is typically 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 [13,38,41,43], the core voltage of the ATMega128RFA1 is lower (1.8 V) than the IO voltage (typically 2.8–3.3 V) [2]. Since these chips are designed to operate from a single 3 V coin cell battery, the lower core voltage reduces power consumption, whereas the higher IO voltage allows the device to operate directly from the coin cell.

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

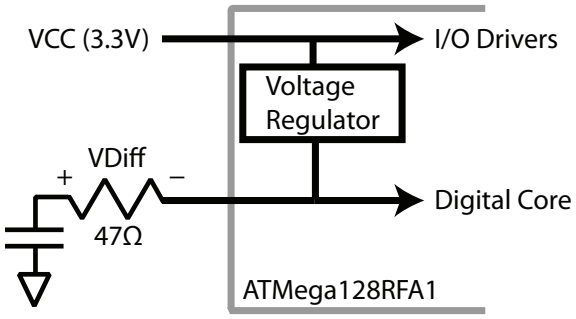

Fig. 2. Because of the internal connection of the voltage regulator for the core voltage, the measurement shunt resistor must be mounted in the decoupling capacitor path.

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 [9]). 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. This is shown schematically in Fig. 2.

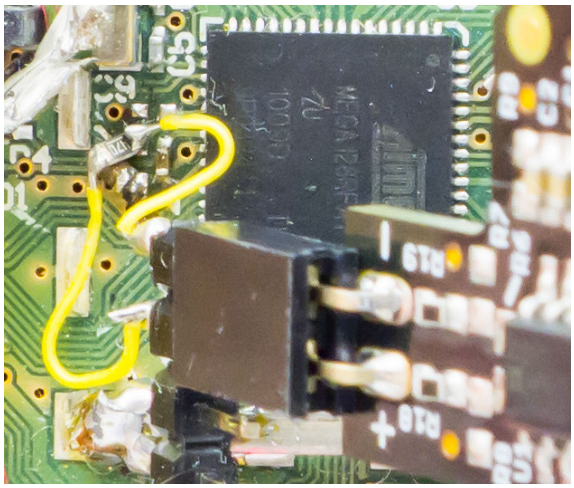

Fig. 3. A 0603-sized 47-ohm resistor was inserted into the VCC_{core} decoupling capacitor, and a differential probe is used to measure across this resistor.

Fig. 3 shows the implementation of this arrangement, where a differential probe is placed across the resistor. An example of the power measurement resulting from this probe is shown in Fig. 4. A number of measurements with a regular oscilloscope are overlaid to provide an indication of the repeatability of the measurement.

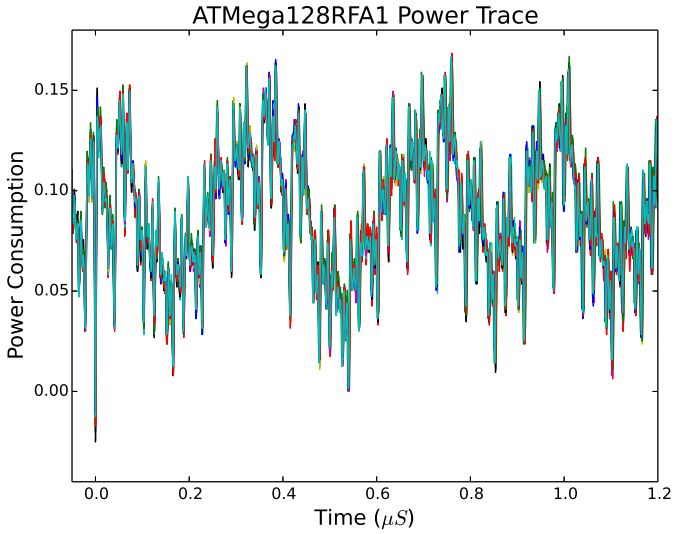

Fig. 4. This figure shows the power trace for the first 1.2 μS of the AES-128 encryption. A total of ten such traces have been overlaid to demonstrate the consistent nature of the signal.

2.3 Related Hardware Attack

To the authors' knowledge, the only previous published attack of an Atmel product with hardware AES acceleration was the XMEGA attack by Kizhvatov [20]. We used the XMEGA attack as a starting point, with the assumption that different Atmel products may use the same internal AES design.

Kizhvatov determined that for a CPA attack on the XMEGA device, a vulnerable sensitive value was the Hamming distance between successive S-box input values. These input values are the XOR of the plaintext with the secret key that occurs during the first `AddRoundKey`. This suggests a single S-box is implemented in hardware, with successive applications of the input values to the S-box.

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}. We discuss the determination of k_{i-1} later, but we can assume for now that byte k_{i-1} is known.

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.

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

As suggested in [20], if k_0 is unknown in practice, an attacker can simply proceed with an attack for all 2^8 possibilities of k_0. The attacker may then test each of the resulting 256 candidate keys to determine the correct value of k_0. This would entail a total of $2^8 \times (2^8 \times 15)$ guesses.

For the specific case of k_0, a more straightforward approach exists. The author of [20] later determined that k_0 can be determined directly by using a leakage assumption based on the Hamming distance from the fixed value `0x00`. This leakage function is shown in (2).

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

This allows the entire encryption key to be attacked with a total of 16×2^8 guesses.[1]

We now attempt to apply this attack to a different device, the ATMega128RFA1.

2.4 Application to ATMega128RFA1

Our experimental platform was a Dresden Elektronik radio board, model number RCB128RFA1 V6.3.1. As mentioned previously, power measurements were taken by inserting a resistor between the VCC_{core} power pin and decoupling capacitor. A differential probe was used to measure the voltage across this resistor. Fig. 1 shows the complete capture setup.

To sample the power measurements, we used an open-source platform called the ChipWhisperer Capture Rev2 [31]. 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).

To reduce noise in the power traces used for side-channel analysis, a band-pass 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 simple test program in the ATMega128RFA1 that encrypts data received over the serial port on the experimental platform. This encryption is 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 standard correlation power analysis (CPA) attack [5], ranking the most likely byte as the one with the highest correlation values. To evaluate our

[1] This is not published in their paper, but was described in private communication from the author.

measurement toolchain, we first performed an attack against a software AES implementation on the ATMega128RFA1.

Fig. 5 shows the results of the CPA attack against the software AES-128 implementation: we recovered the complete key in under 60 traces. These results can be compared to similar attacks using the ChipWhisperer hardware, where a software AES implementation on an AVR microcontroller is broken in around 30 traces [31].

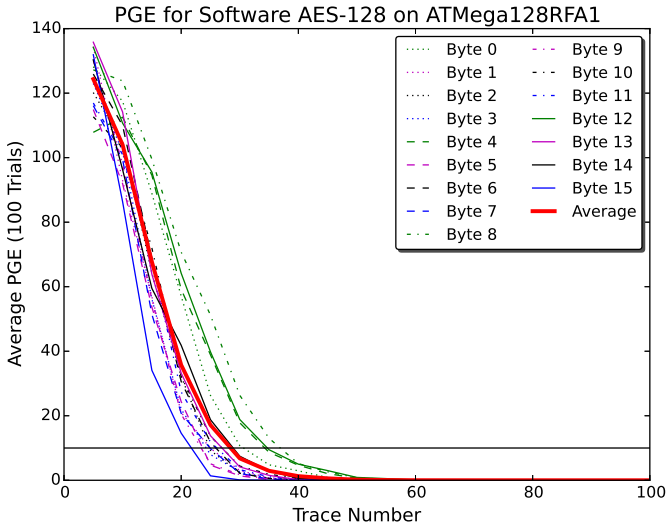

Fig. 5. Attacking a software AES algorithm on the ATMega128RFA1 is used to confirm that the measurement setup is a viable method of measuring the leakage. Here the PGE across all bytes falls to zero in under 60 traces, completely recovering the key.

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[2] 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. 6. 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.

[2] 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.

Examples of the correlation output vs. sample point are shown in Fig. 7, which shows the peaks at the output of the correlation function on the CPA attack for the "correct" key guess. The sign of the peak is not important – the sign will flip depending on probe polarity – but note that the correct key guess results in a larger magnitude correlation than the incorrect guess at certain points. These points are when the physical hardware is performing the operation in (1).

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 $i-1$ [20]. 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}.

2.5 Later-Round Attacks

Whereas previous work has been concerned with determining the first-round encryption key, we will see in Section 4 that information on later-round keys is also required.

We determined that for later 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.

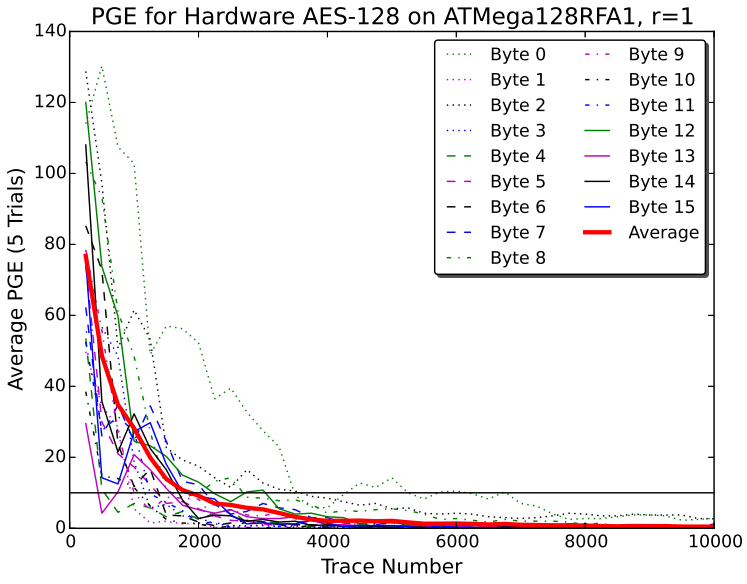

Fig. 6. The CPA attack on the hardware AES peripheral reduces the guessing entropy to reasonable levels in under 5000 traces, and is able to recover the key in around 10 000 traces.

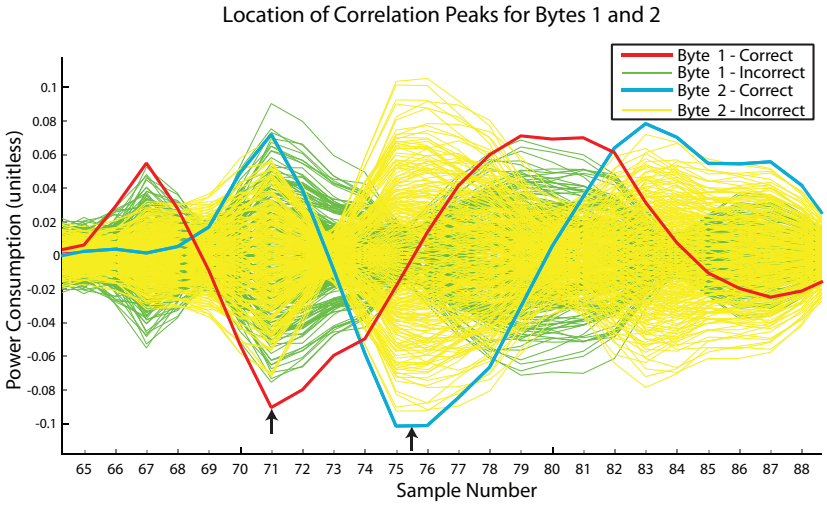

Fig. 7. Correlation peaks for byte $i = 1$ and $i = 2$. The "incorrect guess" means the $2^8 - 1$ guesses which are not the value of k_i. The sample number refers to the sample points since start of the encryption operation, again sampling at 64 MS/s.

Examples of the PGE when attacking the start of the third round ($r = 3$) are given in Fig. 8. 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 2. 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 2 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 is occurring on each clock cycle. Each round takes approximately 32–34 cycles based on the repeating nature of the leakages in later rounds.

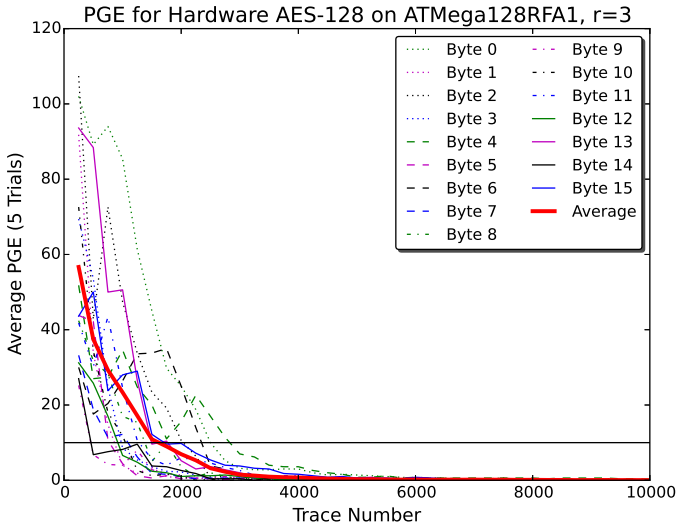

Fig. 8. Attacking later rounds in the AES peripheral is also successful using the same leakage assumptions as the first-round attack.

Determining k_{i-1} Using Later Rounds As described in Section 2.4, 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 2^8 possibilities for k_{i-1}, we can quickly generate 2^8 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}.

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

If we know the initial (first-round) key, we can 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.

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 Section 4.1, where we cannot test candidate keys as we are not recovering the complete key.

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. Devices running IEEE 802.15.4 can achieve extremely low power consumption, running for years on a small battery [17]. When we refer to "IEEE 802.15.4," we are specifically targeting the IEEE 802.15.4-2006 standard.

The IEEE 802.15.4 standard is generally used as a lower layer with another network on top, as IEEE 802.15.4 does not specify details such as routing or distribution of keying material. Examples of popular higher-layer protocols include ZigBee and ISA100.11a (see Section 1 for a full enumeration). These higher-layer

protocols often use IEEE 802.15.4 level security in combination with higher-level security – but this higher-layer security frequently uses the cryptographic primitives provided by IEEE 802.15.4. This paper demonstrates the vulnerabilities of these primitives, and those vulnerabilities may also exist in higher-layer protocols.

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 extends the regular CCM mode by allowing the use of encryption without authentication [18].

CCM itself is a combination of counter mode of AES with cipher block chaining message authentication code (CBC-MAC) [45]. For the side-channel attack, we are only concerned with the details of data passed to the AES-128 block, and not the further processing that occurs after this block.

The AES-128 block itself is used in AES-CTR mode, with an input format as shown in Fig. 9. 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 itself only counts the number of 16-byte blocks in the frame.

To ensure that the nonce is fresh, 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.

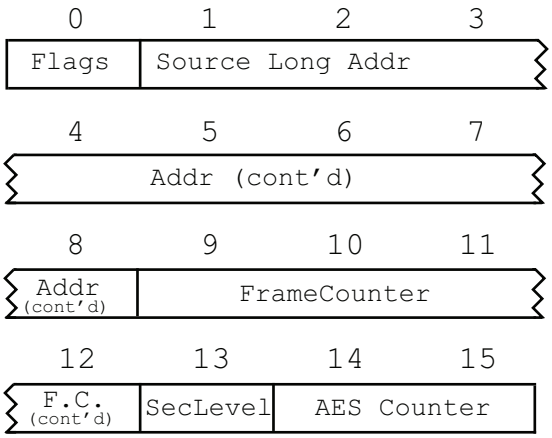

Fig. 9. The following data is used as the input to AES-128 when a frame is decrypted by an IEEE 802.15.4 stack. The `FrameCounter` can be controlled by the attacker.

A typical secure IEEE 802.15.4 wireless packet adds a message authentication code (MAC),[3] which ensures both the integrity and authenticity of this

[3] The name message integrity code (MIC) is used in place of message authentication code (MAC) within the IEEE 802.15.4 standard, as the acronym MAC already

message. The MAC and payload can optionally be encrypted. Although the MAC is optional, configuring nodes to require a MAC is generally recommended, since accepting encrypted but unauthenticated packets presents a serious security risk [37].

The address and header information are never encrypted. This is mostly because it significantly simplifies message filtering: otherwise nodes would need to decrypt every message to determine the address information.

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 simply has a MAC appended) the following operations are performed on the received packet, where processing stops if a step fails:

1. Validate headers and security options.
2. Check that the received frame counter is numerically greater than the last stored frame count.
3. Look up the secret key based on message address and/or key index.
4. Decrypt the payload (and MAC if present).
5. Validate the MAC (if present).
6. Store 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, such as using a valid key identifier and address. Generating such a message is discussed next.

3.1 Detailed Message Format

As mentioned, we need to ensure our 802.15.4 message is decrypted by the target device. Specific requirements vary depending on the network configuration, but as an example the frame used in our experiments was:

```
09 d8 01 ff ff ff ff ba ad 01 02 03 04 05 06 07 08 0d FC FC FC FC 01
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA 00 00
```

Where details of each portion of the frame are described as follows:

---

refers to the medium access control layer. This paper uses MAC to mean message authentication code, and medium access control is spelled out when required.

`09 d8`	Frame header
`01`	Sequence number
`ff ff ff ff`	Broadcast frame
`ba ad`	Source network ID
`01 02 ··· 08`	Source address
`0d`	Security level
`FC FC FC FC`	`FrameCounter` used as part of the AES nonce
`01`	Key ID
`AA AA ··· AA`	16 bytes of encrypted data and 4 bytes of MAC
`00 00`	Replace with CRC-16

The encrypted data and MAC are not used in the attack; only the value of the `FrameCounter` is used. The attacker can send a frame with random values inserted into the `FrameCounter`; they only need to ensure the random values are larger than the last *valid* received value.

Since the attacker's packets will be rejected as invalid once decrypted, the attacker will *not* update the internal frame counter field. An attacker could discover the approximate current `FrameCounter` value through sniffing valid packets sent to the device, since the frame counter is sent unencrypted.

Alternatively, an attacker can simply send very high values – the attacker could ensure the highest two bits are always `10`, `01`, or `11`. Provided the node has received fewer than a billion messages using the same key, this frame counter will be accepted.

The example message used here is sent as a broadcast frame, which ensures it will be received and processed by the target node. Using a broadcast message also hides the target of the attack, and someone sniffing the airwaves might simply assume that a device on a separate network is sending encrypted messages. They might assume either this node is malfunctioning, or the node is using some proprietary protocol that sends encrypted broadcast messages.[4]

3.2 Brute-Force Search

In Section 2.1, the side-channel attack is described as not always fully recovering the encryption key. This necessitates a key search, which requires a comparison function such as having a plaintext/ciphertext pair.

For IEEE 802.15.4, any message with a MAC present can be used as this plaintext/ciphertext pair. It is sufficient for the attacker to sniff a single wireless message secured with a MAC using the target key; the attacker can then use a given hypothetical key to calculate the hypothetical MAC, and compare the resulting MACs to determine if the hypothetical key matches the true key.

[4] Sending a data message that is broadcast across both the network ID and device address is unusual. However, it would work, and it is not unusual for devices to be deployed in the field after only basic testing, leaving in place bad practices such as flooding the network with encrypted messages.

This attack is trivial in practice: most networks using 802.15.4 security have message authentication enabled (i.e., messages have MACs) [37], so the attacker can simply capture a packet directed at or sent from the target. As the MAC may be shorter than the key, they may wish to capture several packets to confirm the true key was found, and not a collision for this particular message.

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 Section 3 and Fig. 9, 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 containing keying material, and is not simply copied blindly from the incoming packet address. This field can be considered effectively fixed. 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 128 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, i.e. we do not take advantage of the fact that each received packet causes more than one encryption operation. Practically this means that when our work refers to requiring N encryption traces, we only need to send a smaller number (in the range $N/4$) packets to the wireless node.

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 [19]. Our work uses many of the constructs developed by Jaffe in [19], 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, and we cannot simply apply the previous attack directly.

In this 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, so we push the attack into later AES rounds to recover the entire encryption key.

Initially, we can assume that the keying material associated with bytes 9–12 can be recovered by a standard CPA attack, as was shown in Section 2. The remaining bytes cannot be recovered, as the input data is constant.

Consider the steps in the AES-ECB encryption algorithm: $AddRoundKey()$, $SubBytes()$, $ShiftRows()$, and $MixColumns()$. All four functions can be assumed to operate on a 16-byte AES state matrix. The first three operate in a byte-wise manner – that is, a single-byte change of the state matrix before the operation results in a single-byte change after the operation. The $MixColumns()$ operation introduces diffusion, which operates on a column of the AES state – a single-byte change in the AES state results in 4 bytes changing after the $MixColumns()$ operation.

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:

$$S'_0 = (2 \circ S_0) \oplus (3 \circ S_1) \oplus S_2 \oplus S_3 \tag{4}$$
$$S'_1 = S_0 \oplus (2 \circ S_1) \oplus (3 \circ S_2) \oplus S_3 \tag{5}$$
$$S'_2 = S_0 \oplus S_1 \oplus (2 \circ S_2) \oplus (3 \circ S_3) \tag{6}$$
$$S'_3 = (3 \circ S_0) \oplus S_1 \oplus S_2 \oplus (2 \circ S_3) \tag{7}$$

Where the input state to the $MixColumns()$ operation was the AES state after the $ShiftRows()$ operation. The output of the $MixColumns()$ operation is used as the input state for the next round.

Constant Inputs For the $MixColumns()$ operation, we can lump all fixed input bytes into a single constant. As described in [19], if bytes S_1, S_2, S_3 of the input were fixed, we could rewrite (4)–(7) as:

$$S'_0 = (2 \circ S_0) \oplus E_0 \tag{8}$$
$$S'_1 = S_0 \oplus E_1 \tag{9}$$
$$S'_2 = S_0 \oplus E_2 \tag{10}$$
$$S'_3 = (3 \circ S_0) \oplus E_3 \tag{11}$$

Note that these bytes are constant but unknown. This occurs because although we do know the fixed plaintext input bytes (i.e., the nonce), we do not

know the keying material used for those bytes. These outputs will become the inputs to the next round of the AES algorithm.

Again using the method from [19], our objective is now to recover the next-round key. Consider that we have gone through one round of AES, and our objective is to recover the second-round key. We do not know the actual input data for this round, which is the output of the $MixColumns()$ step from the previous round. For example, to recover the first key byte k_0, we would need to know the output S_0 from $MixColumns()$. If some of the input bytes to $MixColumns()$ are fixed but unknown, we would instead recover the modified output S'_0.

Performing the CPA attack, we could recover instead a version of this key (we will refer to it as k'_0) XOR'd with the unknown constant E_0, that is $k'_0 = k_0 \oplus E_0$. We can use this modified key as one input to the $AddRoundKey()$ function, where the other input is our modified input to this round S'_0. Note that the output of $AddRoundKey()$ will be equivalent to the case where we had both the true key and true input:

$$\begin{aligned} AddRoundKey(k'_0, S'_0) &= k'_0 \oplus S'_0 \\ &= (k_0 \oplus E_0) \oplus (S_0 \oplus E_0) \\ &= k_0 \oplus S_0 \end{aligned}$$

This is sufficient information to perform the attack on the next round of the AES algorithm. 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.

This fundamental idea is used to attack AES-CCM* as used in IEEE 802.15.4, where many of the input bytes are fixed. By pushing the attack into later rounds, we can recover fixed bytes with a regular CPA attack.

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

p_i^r : "text" input to the $AddRoundKey()$
k_i^r : "round key" input to the $AddRoundKey()$
E_i^r : a constant, see Section 4.1
n_i^r : the modified round key, $k_i^r \oplus E_i^r$
s_i^r : the output of the $SubBytes()$ function
v_i^r : the output of the $ShiftRows()$ function

m_i^r : the output of the $MixColumns()$ function
X : variable and known input plaintext values
Y : variable and known intermediate values
Z : variable and known intermediate values
N : known modified round-key values (n_i^r)
K : known key or round-key values (k_i^r)
c : constant values (may be known or unknown)
? : variable and unknown values
X* : group of variables which has a small set of possible 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`):

$\mathbf{p^1}$ = [c c c c c c c c c X X X X c c c]

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 Section 2.5 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:

$\mathbf{r^1}$ = [c 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 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 described in Section 4.1. As in [19], 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*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*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 Section 2.5, 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 Section 2.4.

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, which are generated by (6) and (7) respectively as the previous-round $MixColumns(\mathbf{v^1})$ outputs m_6^1 and m_6^1. When attacking n_7^2, we apply (1) to (6) and (7). This means our leakage is:

$$HW\left((n_6^2 \oplus (6)) \oplus (n_7^2 \oplus (7))\right) \tag{12}$$

The XOR cancels common terms in (6) and (7), and in this case that term is S_1. Here S_1 is the variable and known input to the $MixColumns(\mathbf{v^1})$, the result being that 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 Section 2.5 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 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 this round.

The outputs $8, \cdots, 11$ of $MixColumns(\mathbf{v^1})$ from the first round are constant, so we also know these inputs are constant, and 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. 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*Z]

This becomes the input to the next round:

$\mathbf{p^3}$ = [Z*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 Section 2.5. 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, as in Fig. 10. 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

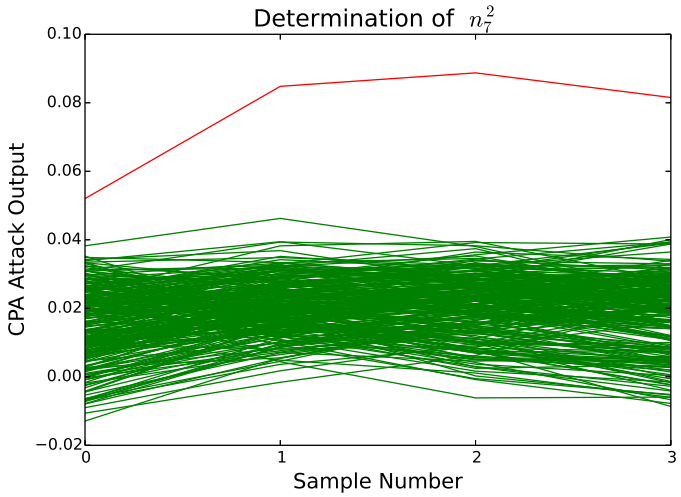

Fig. 10. Determining n_7^2 means generating 256 candidate keys, which are tested by attacking a byte in the next round. This figure shows the correlation output of the CPA attack when attacking n_8^3 for the correct value of n_7^2 and n_8^3 in red. The incorrect values are displayed in green.

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 `FrameCounter` bytes, which are sent as plaintext over the air, as detailed in Section 3.1.

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 the 20 000 traces, using only the subset of points in each trace from Table 2. Note when performing the hypothetical value calculation for later 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 [16].

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. This section

demonstrates the ability of an attacker to perform these operations, and thus gives a complete attack against an IEEE 802.15.4 wireless node.

In Section 3.1 we detailed the message format that would cause an IEEE 802.15.4 wireless node to automatically decrypt the message on receipt. To validate this we used the same Dresden Elektronik RCB128RFA1 V6.3.1 board as in Section 2, programmed with Atmel's IEEE 802.15.4 stack version 2.8.0. We used the "Secure Star Network" example application for this, which initializes a standard IEEE 802.15.4 networking using security.

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, with a standard deviation of 7 μs.

The jitter in the delay is assumed to be due to the software architecture, which uses an event queue process the frames. Practically, the issue of aligning or resynchronizing power traces before applying power analysis is well known, and a number of solutions have been proposed, such as comb filtering or windowing [7], differential frequency analysis [15], dynamic time warping [44], and principal component analysis [4].

To test the ability of an attacker to realign captured power traces, we used a simple normalized cross-correlation algorithm [22] to match a feature across multiple power traces for realignment, using the `scikit-image` implementation of this feature matching. This is an example of a *static alignment* method [23].

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. A plot of the output of the cross-correlation for different offsets of the template against another trace is shown in Fig. 11. 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. This eliminates problems with a particularly noisy trace being matched incorrectly.

The successful realignment of traces is an important step when attacking real systems. As an example of trace realignment on another platform, see the attack on DESFire [33] which demonstrated how differential frequency analysis was used as a preprocessing step for CPA attack.

In a similar manner, it was demonstrated in [9] that it was possible to detect the location of software AES encryption based on the transmitter output of a wireless node coupled with further signal processing.

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.

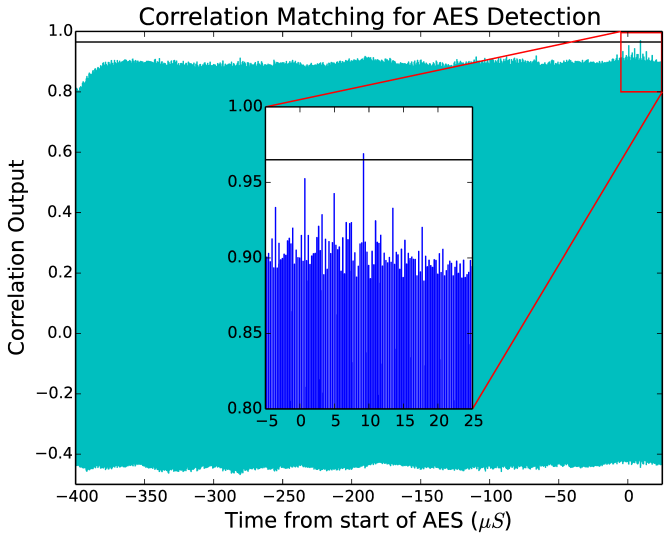

Fig. 11. Correlation can be used to match a template from the power signature to align traces in the time domain, here the correct match should be at 9.2 μs, the location of the largest correlation peak.

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 Section 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 significant vulnerabilities in a real IEEE 802.15.4 wireless node. A successful attack against the AES peripheral 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 [19] 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.

While this attack targeted a specific IEEE 802.15.4 SoC from Atmel, we believe similar attacks would be successful against AES peripherals on other devices. Users of IEEE 802.15.4 wireless networks, including users of protocols running on top of IEEE 802.15.4, need to seriously consider their security requirements in light of these attacks.

References

1. Agrawal, D., Rao, J., and Rohatgi, P. Multi-channel Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2003*, C. Walter, Ç. Koç, and C. Paar, Eds., vol. 2779 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 2–16.
2. Atmel Corporation. ATmega128RFA1 Datasheet, 2014.
3. Balasch, J., Gierlichs, B., Verdult, R., Batina, L., and Verbauwhede, I. Power Analysis of Atmel CryptoMemory – Recovering Keys from Secure EEPROMs. In *Topics in Cryptology, CT-RSA 2012*, O. Dunkelman, Ed., vol. 7178 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 19–34.
4. Batina, L., Hogenboom, J., and van Woudenberg, J. Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In *Topics in Cryptology – CT-RSA 2012*, O. Dunkelman, Ed., vol. 7178 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 383–397.
5. Brier, E., Clavier, C., and Olivier, F. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 16–29.
6. Chari, S., Rao, J., and Rohatgi, P. Template Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, B. Kaliski, Ç. Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 13–28.
7. Clavier, C., Coron, J.-S., and Dabbous, N. Differential Power Analysis in the Presence of Hardware Countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, Ç. Koç and C. Paar, Eds., vol. 1965 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pp. 252–263.
8. Clavier, C., Danger, J.-L., Duc, G., Elaabid, M., Gêrard, B., Guilley, S., Heuser, A., Kasper, M., Li, Y., Lomné, V., Nakatsu, D., Ohta, K., Sakiyama, K., Sauvage, L., Schindler, W., Stöttinger, M., Veyrat-Charvillon, N., Walle, M., and Wurcker, A. Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest. *Journal of Cryptographic Engineering 4*, 4 (2014), 259–274.
9. de Meulenaer, G., and Standaert, F.-X. Stealthy Compromise of Wireless Sensor Nodes with Power Analysis Attacks. In *Mobile Lightweight Wireless Systems*, P. Chatzimisios, C. Verikoukis, I. Santamaría, M. Laddomada, and O. Hoffmann, Eds., vol. 45 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, 2010, pp. 229–242.
10. Duc, G., Guilley, S., Sauvage, L., Flament, F., Nassar, M., Selmane, N., Danger, J.-L., Graba, T., Mathieu, Y., and Renaud, P. Results of the 2009-2010 'DPA contest v2'. In *International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)* (February 2011).

11. Duda, R., and Hart, P. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.
12. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., and Shalmani, M. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 203–220.
13. Freescale. MC1323x Datasheet, 2011.
14. Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic Analysis: Concrete Results. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, Ç. Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 251–261.
15. Gebotys, C., Ho, S., and Tiu, C. EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, J. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 250–264.
16. Gueron, S. Intel Advanced Encryption Standard (AES) New Instructions Set. Whitepaper, 2012. Doc No. 323641-001.
17. Gutierrez, J., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V., and Heile, B. IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *IEEE Network 15*, 5 (Sept 2001), 12–19.
18. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). *IEEE Std. 802.15.4-2006* (2006).
19. Jaffe, J. A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter. In *Cryptographic Hardware and Embedded Systems – CHES 2007*, P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 1–13.
20. Kizhvatov, I. Side Channel Analysis of AVR XMEGA Crypto Engine. In *Proceedings of the 4th Workshop on Embedded Systems Security* (New York, NY, USA, 2009), WESS '09, ACM, pp. 8:1–8:7.
21. Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology – CRYPTO' 99* (1999), M. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 388–397.
22. Lewis, J. P. Fast Template Matching. In *Canadian Conference on Vision Interface – VI 1995* (1995), pp. 120–123.
23. Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer, New York, 2007.
24. Massey, J. Guessing and entropy. In *IEEE International Symposium on Information Theory, 1994* (1994), p. 204.
25. Moradi, A., Barenghi, A., Kasper, T., and Paar, C. On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM Conference on Computer and Communications Security – CCS '11* (2011), ACM, pp. 111–124.
26. Moradi, A., Kasper, M., and Paar, C. On the Portability of Side-Channel Attacks – An Analysis of the Xilinx Virtex 4, Virtex 5, and Spartan 6 Bitstream Encryption Mechanism. Cryptology ePrint Archive, Report 2011/391, 2011. `http://eprint.iacr.org/`.
27. Moradi, A., Kasper, M., and Paar, C. Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures. In *Topics in Cryptology – CT-RSA 2012*,

O. Dunkelman, Ed., vol. 7178 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 1–18.

28. Moradi, A., Oswald, D., Paar, C., and Swierczynski, P. Side-channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II: Facilitating Black-box Analysis Using Software Reverse-engineering. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays – FPGA'13* (2013), ACM, pp. 91–100.
29. North, D. An analysis of the factors which determine signal/noise discrimination in pulsed-carrier systems. *RCA Labs.* (1943).
30. O'Flynn, C., and Chen, Z. A Case Study of Side-Channel Analysis Using Decoupling Capacitor Power Measurement with the OpenADC. In *Foundations and Practice of Security*, J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, A. Miri, and N. Tawbi, Eds., vol. 7743 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 341–356.
31. O'Flynn, C., and Chen, Z. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In *Constructive Side-Channel Analysis and Secure Design*, E. Prouff, Ed., vol. 8622 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 243–260.
32. Oswald, D. *Implementation Attacks: From Theory to Practice*. PhD thesis, Ruhr-Universit at Bochum, 2013.
33. Oswald, D., and Paar, C. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, B. Preneel and T. Takagi, Eds., vol. 6917 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 207–222.
34. Oswald, D., Richter, B., and Paar, C. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In *Research in Attacks, Intrusions, and Defenses*, S. Stolfo, A. Stavrou, and C. Wright, Eds., vol. 8145 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 204–222.
35. Oswald, D., Strobel, D., Schellenberg, F., Kasper, T., and Paar, C. When Reverse-Engineering Meets Side-Channel Analysis – Digital Lockpicking in Practice. In *Selected Areas in Cryptography – SAC '13*, T. Lange, K. Lauter, and P. Lisoněk, Eds., vol. 8282 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 571–588.
36. Rivain, M. On the Exact Success Rate of Side Channel Analysis in the Gaussian Model. In *Selected Areas in Cryptography*, R. Avanzi, L. Keliher, and F. Sica, Eds., vol. 5381 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 165–183.
37. Sastry, N., and Wagner, D. Security Considerations for IEEE 802.15.4 Networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security – WiSe '04* (2004), ACM, pp. 32–42.
38. Silicon Labratories. E35x Datasheet, 2013.
39. Skorobogatov, S., and Woods, C. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds., vol. 7428 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 23–40.
40. Standaert, F.-X., Malkin, T., and Yung, M. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 443–461.
41. STMicroelectronics. STM32W108xx Datasheet, 2013.

42. Swierczynski, P., Moradi, A., Oswald, D., and Paar, C. Physical Security Evaluation of the Bitstream Encryption Mechanism of Altera Stratix II and Stratix III FPGAs. *ACM Trans. Reconfigurable Technol. Syst. 7*, 4 (Dec. 2014), 7:1–7:23.
43. Texas Instruments. CC2530F Datasheet, 2015.
44. van Woudenberg, J. G. J., Witteman, M. F., and Bakker, B. Improving Differential Power Analysis by Elastic Alignment. In *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011* (Berlin, Heidelberg, 2011), CT-RSA'11, Springer-Verlag, pp. 104–119.
45. Whiting, D., Ferguson, N., and Housley, R. Counter with CBC-MAC (CCM). `https://tools.ietf.org/html/rfc3610`.

7 Appendix A: Meaning of PGE

The results of this work use the partial guessing entropy (PGE) as a metric of the attack success. A brief description of this within the scope of the side-channel analysis power attacks is presented here.

The "guessing entropy" is defined as the "average number of successive guesses required with an optimum strategy to determine the true value of a random variable X" [24]. The "optimum strategy" here is to rank the possible values of the byte from most to least likely based on the value of the correlation attack (higher correlation output is more likely).

The "partial" refers to the fact that we are finding the guessing entropy on each byte. This gives us a PGE for each of the 16 bytes. A PGE of 0 indicates the byte is perfectly known, and a PGE of 10 indicates that 10 guesses were (incorrectly) ranked higher than the correct guess.

The attack algorithm is given access to $1, 2, \cdots, N$ traces, and the PGE for each byte is calculated. To improve consistency, the PGE for each byte is averaged over several attacks (trials). Finally, we can average the PGE over all 16 bytes to generate a single "average PGE" for the attack.

The guessing entropy provides a simple measure of how a leakage reduces the required key search space. It is sufficient for an attacker to reduce the key search space to some reasonable size, compared to requiring the attacker to entirely derive the key through side-channel attacks. The guessing entropy can be related to the success rate, another frequently used metric in side-channel attacks [36].

Details of PGE and additional metrics are given in [40], and further examples of plots comparing different metrics are available as part of the DPAContest results [8,10].