# Apple vs. EMA: Electromagnetic Side Channel Attacks on Apple CoreCrypto

Gregor Haas, Aydin Aysu
{ghaas,aaysu}@ncsu.edu
North Carolina State University
Raleigh, NC, USA

## ABSTRACT

Cryptographic instruction set extensions are commonly used for ciphers which would otherwise face unacceptable side channel risks. A prominent example of such an extension is the ARMv8 Cryptographic Extension, or ARM CE for short, which defines dedicated instructions to securely accelerate AES. However, while these extensions may be resistant to traditional "digital" side channel attacks, they may still vulnerable to physical side channel attacks.

In this work, we demonstrate the first such attack on a standard ARM CE AES implementation. We specifically focus on the implementation used by Apple's CoreCrypto library which we run on the Apple A10 Fusion SoC. To that end, we implement an optimized side channel acquisition infrastructure involving both custom iPhone software and accelerated analysis code. We find that an adversary which can observe 5-30 million known-ciphertext traces can reliably extract secret AES keys using electromagnetic (EM) radiation as a side channel. This corresponds to an encryption operation on less than half of a gigabyte of data, which could be acquired in less than 2 seconds on the iPhone 7 we examined. Our attack thus highlights the need for side channel defenses for real devices and production, industry-standard encryption software.

## 1 INTRODUCTION

Symmetric-key block ciphers are a pillar of modern cryptosystems used in healthcare, finance, education, and consumer electronics applications, among others [19]. The most common symmetric cipher is the Advanced Encryption Standard (AES) [7]. AES is based on a series of diffusing, confusing, and nonlinear operations repeated for several rounds. While AES is, so far, mathematically secure, its implementation can still be vulnerable. Side channel attacks, for example, correlate secret-key dependent operations with implementation behaviors such as execution time [3], power consumption [18] or electromagnetic (EM) radiation [9].

Software-based implementations of AES often rely on lookup tables to store various constants. In particular, efficient implementations rely on so-called t-tables [8], which collectively store about 4kB of constants used in the encryption process. These t-tables have been shown to be highly susceptible to cache-based timing side-channel attacks—even adversaries with limited capabilities [3] are able to extract full AES keys from such implementations given enough measurements. If adversaries are given additional capabilities, such as active cache manipulation, several attacks are possible which extract keys more efficiently [11, 23, 27].

In response to these cache attacks, CPU designers now include instruction set extensions which can implement AES without auxiliary memory accesses. Typically, these instructions execute in a processor's floating-point pipeline due to the availability of larger registers which can accommodate the entire cryptographic state. Both Intel [1] and ARM [13] have specified such instruction set extensions (AES-NI and ARM CE, respectively). Although these implementations can mitigate digital side-channels abusing cache timing, they can be vulnerable to physical side-channel attacks such as those that use power consumption or EM radiation. To the best of our knowledge, a physical side-channel attack on AES-NI has recently been shown [20], but the vulnerability of ARM CE implementations is, as of yet, unknown.

In this paper, we present the first physical side-channel attack on an ARM CE based AES implementation. We target the ARM CE code from Apple's CoreCrypto library, which is the standard provider of cryptographic functionality on iPhones [16]. We specifically use EM side channel measurements for our attack. To instrument our attack, we reverse-engineered the iPhone's printed circuit board (PCB) and developed a test infrastructure that allows sending inputs and observing outputs directly on the main processor. This hardware-based research is then coupled with a recent research toolkit [12] which exploits an iPhone boot vulnerability and allows us to execute arbitrary code on an iPhone 7's ARM processor. Finally, targeting Apple's own cryptography library [15], we acquire a high number of side channel traces and show that inputs and outputs can be observed using leakage assessment techniques [5, 25]. We next proceed to show an attack on AES intermediates, and describe how correlation for these can leak the secret key being used. Specifically, we find that *unprivileged* adversaries can extract secret keys from this AES implementation when observing encryption operations on less than half of a gigabyte of data.

### 1.1 Contributions

We specifically make the following contribution in this work:

(1) Extending a recently published research toolkit [12], we create a high-speed acquisition platform which allows us to collect and analyze >16000 side channel traces per second.
(2) As part of the acquisition platform, we make several novel modifications to an iPhone 7 printed circuit board (PCB).
(3) We target a common Apple SoC running at a high frequency rate (2.34GHz) with a realistic adversarial model.
(4) For the first time, we show that a RISC-style AES instruction set extension is vulnerable to EM-based side channel attacks. We do so by showing known-key correlations for both input/output and cipher intermediates, as well as analyzing the attack's performance in an unknown-key setting.

## 2 BACKGROUND

In this section, we present background information on the mathematical underpinnings of the cryptosystem under attack, as well as the concrete instantiation which we are targeting in this work. We then present a brief summary of the statistical techniques which we use in our attack. Finally, we conclude with a literature review of relevant prior work and identify some gaps in the research, which we aim to fill in this paper.

### 2.1 Advanced Encryption Standard (AES)

We consider AES128 in the ECB mode of operation. In this mode, AES consists of 10 rounds of encryption, as well as a preliminary key expansion phase—details of key expansion are out of scope for this work. Each encryption round is composed of the `AddRoundKey`, `SubBytes`, `ShiftRows`, and `MixColumns` operations. Specifically, the first 9 rounds consists of each of these operations in order, but the last round replaces `MixColumns` with another `AddRoundKey`. The AES state $A_{0-3,0-3}$ and key state $K_{0-3,0-3}$ are both indexed as 2D

$4 \times 4$ arrays. Each AES operation takes a 16-byte input state $A$ and produces a 16-byte output state $A'$, sometimes aided by a lookup table[1]. Updates to the output state are simultaneous for all bytes. Indices for all states are taken modulo 4—this notation is implied, and excluded for brevity. The AES operations are:

- `AddRoundKey`: $K^r$ is a round key: $A'_{y,x} = A_{y,x} \oplus K^r_{y,x}$
- `SubBytes`: $S$ is a 256-byte lookup table: $A'_{y,x} = S\left[A_{y,x}\right]$
- `ShiftRows`: $A'_{y,x} = A_{y,y+x}$
- `MixColumns`: $X$ is a 4-byte lookup table: $A'_{y,x} = \sum_{i=0}^{3} X_{i-y} A_{i,x}$

## 2.2 ARM-CE AES

ARM defines its cryptographic extensions (CE) as additional instructions which execute in the processor's NEON floating point unit (FPU) [13]. While various instructions are defined in the ISA extension, only two are relevant for this work: **`aese`**, which implements `AddRoundKey`, `SubBytes`, and `ShiftRows`, and **`aesmc`** which implements `MixColumns`. In typical AES implementations using ARM CE, these instructions are executed in a loop. Each iteration fetches the next round key $K^r$, executes **`aese`** and **`aesmc`**, and then branches conditionally if more rounds must be computed. In this work, we specifically attack Apple's implementation (downloaded from [15]) in the file `acceleratecrypto/Source/aes/arm64/encrypt.s`.

## 2.3 Correlation Power Analysis (CPA)

CPA [5] is a statistical technique for extracting secret information from side channel traces. CPA uses such traces defined as $T = \{D, s_0, s_1, \ldots, s_{n-1}\}$, consisting of $n$ evenly spaced time-domain samples as well as some associated data $D$—for our attack, $D$ contains a plaintext $P$ and corresponding AES-encrypted ciphertext $C$. Then, the goal of a CPA attack is to extract evidence for secret values of some secret-derived intermediate. To do so, we must define a leakage model $\mathcal{L}(D, \mathcal{K})$ which (for our attack) returns the estimated EM activity for the known associated data $D$ and some guess $\mathcal{K}$ for the unknown secret $K$. Then, after obtaining $m$ side channel traces and calculating the associated leakage models, we can calculate Pearson's correlation coefficient at each sample index $i$:

$$\rho_i = \frac{\sum_{j=0}^{m-1} (\mathcal{L}(D_j, \mathcal{K}) - \mu^P)(s_{j,i} - \mu^i)}{\sigma^P \sigma^i} \tag{1}$$

where $\mu^{i/p}$, $\sigma^{i/p}$ represent the mean and standard deviation across all of the samples at a specific index $i$ and of the leakage models, respectively. Higher values of $\rho_i$ then indicate that we observe the hypothesized activity from the leakage model at that specific sample index $i$, and can also provide evidence that the guessed secret value $\mathcal{K}$ is correct, compared to other guesses.

## 2.4 Previous Attacks on ARM CPUs

Previous works have dealt with EM attacks on ARM processors generally, as well as iPhones specifically. Most notable is [10], where the authors are able to extract secret ECDSA keys from iPhones using low-cost EM acquisition equipment and unmodified iOS software. However, this is a rather coarse-grained attack on an *asymmetric* cryptosystem, and the side channel information is mainly used to infer control flow. Such an attack is not applicable to symmetric cryptosystems such as AES.

Other work [22] has focused on analyzing the various AES implementations of the BeagleBone Black, an ARM-based development board. In this work, the authors use an EM-based side channel attack to break AES as implemented in software, a proprietary crypto coprocessor, and a NEON-based implementation. We note that while the authors also attack the NEON FPU, they target a bitsliced [4, 17] implementation which does not utilize ARM-CE. Similarly, further work [2] has shown that such EM attacks can succeed even against *masked*, bitsliced, software implementations of AES.

More recent work [21] has shown that Apple's proprietary AES coprocessor hardware is also vulnerable to EM side channel attacks. The attack is shown on significantly older model of iPhone (*e.g.*, iPhone 4) and argued to be infeasible for iPhone 6 and future generations. We also note that this coprocessor is primarily used for user authentication and firmware decryption rather than general-purpose cryptography. For many use cases outside of these, Apple's cryptographic library will prefer an ARM CE implementation.

## 3 THREAT MODEL

Our threat model follows the standard physical side-channel attack threat model of prior work [10, 20–22]. We consider adversaries (not necessarily privileged) who have the ability to observe electromagnetic radiation from a victim device, while the device is executing a cryptographic operation utilizing an ARM CE-based AES implementation. The adversary knows both the input plaintext as well as the output ciphertext and aims to extract the secret key. Our attack is significantly improved if the adversary has access to accurate timestamps collected at the beginning of each 128-bit encryption block (Section 4.2). However, this is not a hard requirement—lack of such knowledge only leads to increased trace acquisition times.

Unlike template [6] or machine-learning [14] based attacks, our attack does not require building precise models by configuring all (or a representative set of) possible sub-keys on each device. Instead, we conduct an *unprofiled* attack with CPA. Our attack is therefore more generic and assumes a weaker adversary. We still do initially configure the device with a known key to analyze which operations leak more information in order to design the run-time attack (Section 4.3). However, this must only be done once for a specific device family or software, and the final attack at runtime does not require profiles/models.

## 3.1 Victim Behavior

As described in our results (Section 5), our attack requires on the order of 5-30 million traces to succeed. This corresponds to a cryptographic operation on approximately 80-480 megabytes of data. Thanks to our preprocessing approach, this data could be collected in a single trace (assuming a CBC or sequential ECB, etc mode of encryption) and then split apart into its component encryption operations. We measured an average encryption time of 62.5 nanoseconds per 128-bit block, so all necessary data could be acquired within 0.313 - 1.875 seconds. Example mobile applications which might fit this execution model could include encrypted cloud backup solutions (such as iCloud [16]) or decryption of system software updates. We argue that such applications are realistic targets for the attack we present in this work. Prior work [20, 24] requires a similarly high number of traces, but also requires much longer data acquisition times (approximately 2 weeks).

## 4 PROPOSED ATTACK

iPhones are a nontrivial platform to perform hardware security research on. Therefore, we use the iTimed toolkit [12] to bootstrap our research. We first compile a custom Linux kernel which runs on our target iPhone 7. Then, within the kernel, we implement a custom system call which allows for our victim code to execute atomically, without interrupts. We integrate Apple's ARM-CE AES implementation [15] into the Linux kernel's cryptography infrastructure, which allows us to evaluate its side channel security. The Linux kernel also includes other highly useful functionality, such as GPIO control and CPU frequency scaling. We note that, due to the scalability of the iTimed toolkit [12], our attack can work similarly for all iPhones up to the iPhone X.

[1] $S = [\texttt{0x63}, \texttt{0x7c}, \ldots, \texttt{0x16}]$, $X = [2, 3, 1, 1]$

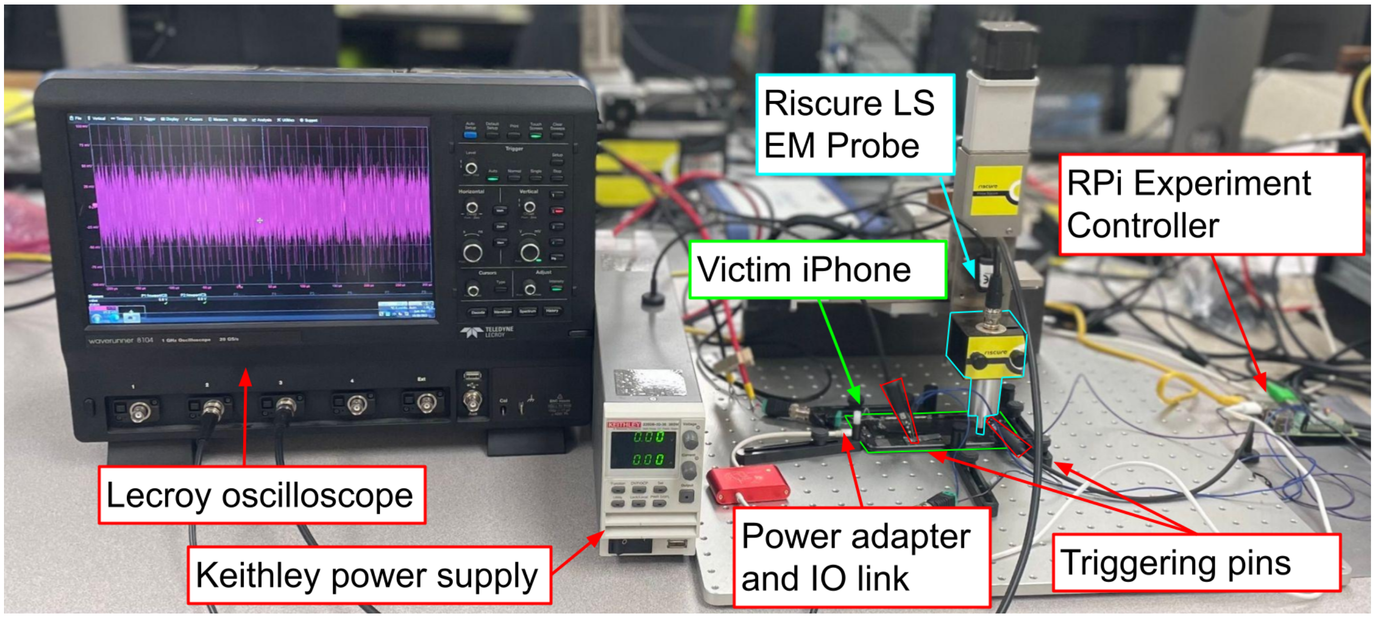

**Figure 1: Overall view of our full experimental infrastructure. Components near the victim iPhone are highlighted for clarity.**

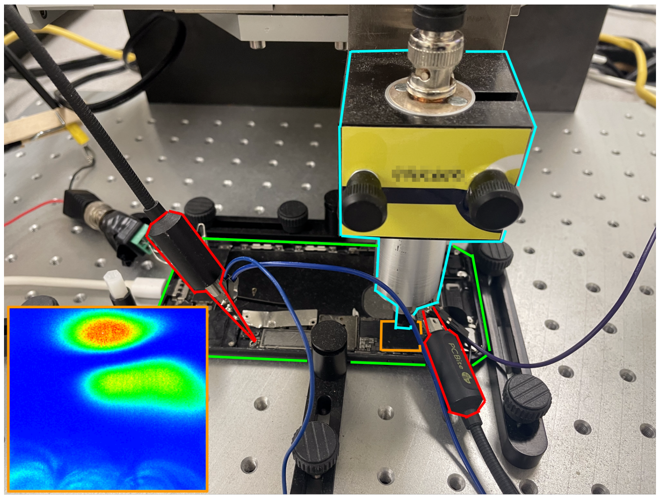

**Figure 2: Closer view of the victim iPhone in Figure 1, following the same highlighting scheme. Additionally highlighted in orange is the Apple A10 SoC, whose floating-point pipeline is the target of this work. *(Inset, bottom left)* An EM heatmap obtained by scanning the surface of the SoC, and filtering for the 2.34GHz operating frequency. This heatmap helped inform our probe placement near the chip's top edge.**

## 4.1 iPhone 7 PCB Setup

Figures 1 and 2 are pictures of our acquisition infrastructure. For collecting measurements, we use a Teledyne Lecroy Waverunner 8104 running at 10GHz and collecting traces 32 million samples long (discussed more in the following section). A Riscure low-sensitivity (LS) EM probe sits above our region-of-interest on the target iPhone. The exact position of the probe is set by scanning the chip surface, generating an EM heatmap by filtering for the clock frequency, and positioning the probe to the high activity region. Power is supplied by a Keithley 2260b-30-36 DC power supply, through a custom battery connector into the iPhone's logic board. We connect the oscilloscope trigger to an exposed, software-controllable GPIO pin on the logic board—this allows us to reliably measure AES encryptions if we set the GPIO pin high before.

Prior research toolkits [12] allow us to set the encrypting CPU's frequency to 2.34 GHz for each of our experiments—this is the maximum value. This makes the attack as hard and as realistic as possible. In addition, we developed several hardware modifications to the iPhone 7 PCB which further increase the efficiency of our acquisition infrastructure. For example, we probe the aforementioned GPIO pins using a set of needle probes—finding the physical locations of these software-controllable pins required an extensive reverse-engineering process. Furthermore, the custom battery connector was precisely extracted from a real iPhone 7 battery so that we could externally and stably power the iPhone for long experiment runs.

## 4.2 EM Acquisition Methodology

Our experimental setup sends plaintexts $M$ to the target device, coordinates the aforementioned acquisition equipment, and then receives the final ciphertext $C$ after encryption completes. At first, we simply captured EM traces for one plaintext at a time—however, this approach led to high acquisition times for the number of traces our attack requires (Section 5). We thus follow approaches suggested in previous works [25], and instead acquire EM traces which contain side channel data for multiple plaintexts.

Our experimental setup first sends one plaintext $M_0$ to the device, and then triggers the acquisition setup. The device then proceeds to *iteratively* encrypt this plaintext, such that it first encrypts $M_0$, then $M_1 = E_K(M)$, then $M_2 = E_K(M_1)$, up to a set number of iterations $i$. Before each encryption, the device collects a timestamp $T_i$ which is used when preprocessing the traces. For this specific set of experiments, our acquisition equipment allowed us to set $i = 3000$ resulting in traces approximately 32 million samples long. By doing so, we increased our side channel acquisition speed by two orders of magnitude, up to approximately 16000 plaintexts per second.

*4.2.1 Trace Unpacking.* Before they can be used for side channel processing, these long traces first need to be "unpacked" into subtraces. Specifically, for each message $M_i$, we need to find the set of associated EM samples in the time domain. We do so with a dual approach. First, we define some reference sample pattern (of length $n$ samples) which always occurs right before the target encryption. Then, we calculate Pearson's correlation coefficient between each subpattern of length $n$ in the side channel trace and the reference pattern. We developed a highly optimized, GPU-accelerated Pearson correlation implementation, so we find that this calculation is not

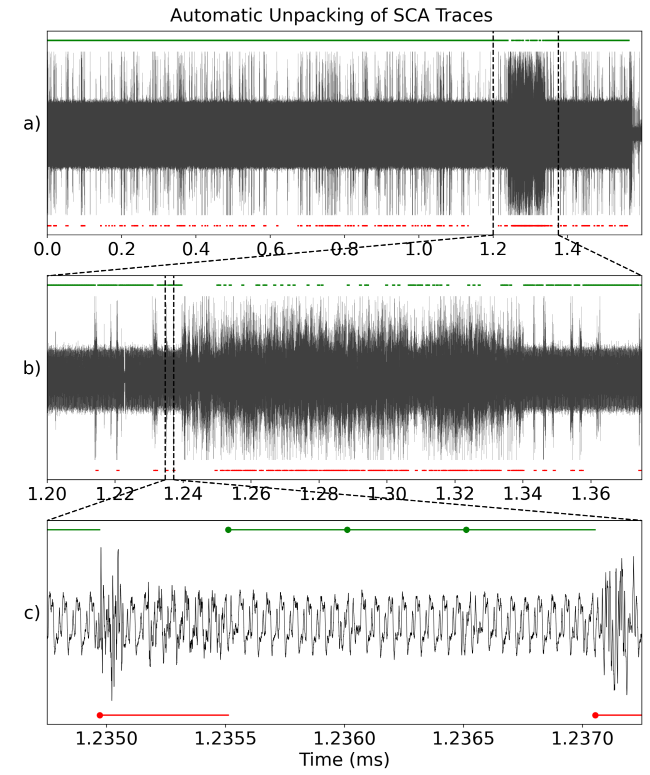

**Figure 3: EM side channel trace at various zoom levels. Red lines below the trace indicate regions for which we have only the timing measurement. Green lines above the trace indicate regions for which we also have a reliable pattern match. *(a)* A full EM trace, containing 32000000 samples and 3000 encryptions. *(b)* Zooming in to an interrupt (with no timing-confirmed encryptions) between 1.24 ms and 1.25 ms, as well as a following region of high-noise missed subtraces. *(c)* Individual subtraces containing 8 repeated encryptions of the same plaintext, with some noise.**

a bottleneck in practice. Then, with knowledge of likely pattern matches and the average encryption time, we can begin extracting subtraces with the correct associated data $M_i$.

However, we find that our EM traces sometimes exhibit sporadic noise or idle regions as if the encryption process were interrupted—see, for example, Figure 3 between 1.24 ms and 1.25 ms. In these cases, we find that there are often "gaps" in between confirmed encryption pattern matches where we expect to see one or more other subtraces, based on average encryption time. This situation is problematic—if we do not know how many "missed" subtraces are contained within a gap, we can no longer derive the correct associated plaintext $M_i$ for any matched subtraces after the gap. Therefore, we cross-reference with the collected timing measurements $T_i$ to determine how many subtraces are missed due to noise or interrupts. This approach allows us to reliably associate $M_i$ with the corresponding EM samples, while maintaining data integrity, greatly decreasing acquisition time, and only marginally increasing computation time.

Figure 3 shows an average long EM trace, visualized at three different zoom levels. Our subtrace extraction algorithm categorizes subtraces according to both reliable pattern matches and corresponding timing measurements. In our subsequent analyses, we discard subtraces for which we do not have a good pattern match due to noise. However, we must still know *how many* such subtraces are missed in noisy regions so we can derive the correct $M_i$. For all following experiments, we use the automatically extracted subtraces associated with their correct $M_i$.

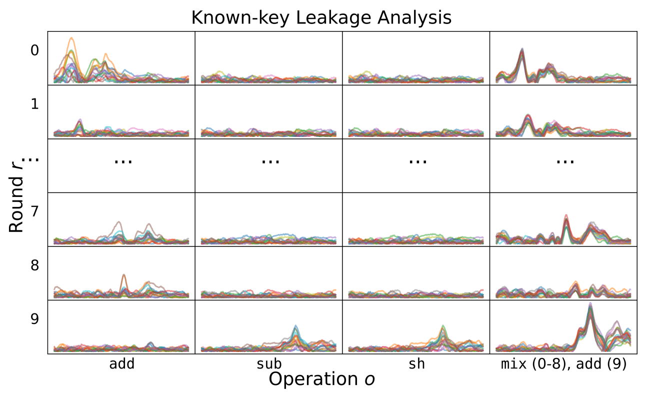

**Figure 4: Known-key correlation plots for each AES operation in each round. The correlation plots for all 16 byte positions are superimposed in each subplot. Clearly, the `Mixcolumns` operation leaks most regularly aside from early, input correlation-derived peaks.**

## 4.3 Leakage Models

Next, we need to identify a leakage model $\mathcal{L}(D, \mathcal{K})$ as discussed in Section 2.3. To do so, we first investigate which intermediate values *might* leak by running a known-key analysis. In this analysis, we assume that the adversary knows the secret key $K$ and thus the value of each byte of intermediate AES state. We emphasize that this approach is merely used as an initial security evaluation—in our final attack, we do not assume that the adversary has such knowledge. Then, having identified the `MixColumns` operation as potentially vulnerable, we define a CPA attack which will later allow us to extract the key (Section 5).

*4.3.1 Known-key Evaluation.* For the known-key evaluation, we define 656 leakage models[2] $\mathcal{L}^{o,r}_{y,x}(D, K)$. Each leakage model returns the Hamming weight of byte $y, x$ in the intermediate state produced *after* executing operation $o \in$ [`pt, add, sub, sh, mix`] in round $r \in [0-9]$. Each operation in $o$ corresponds to an AES operation as defined in Section 2.1, respectively—the operation `pt` is a special case, denoting the input plaintext to the cryptosystem. As an example, given that the associated data $D$ contains the plaintext $P$ and ciphertext $C$, we would have the following leakage models:

$$\begin{aligned}
\mathcal{L}^{\mathsf{pt},-}_{y,x}(D,K) &= HW\left(P_{y,x}\right)\\
\mathcal{L}^{\mathsf{add},0}_{y,x}(D,K) &= HW\left(P_{y,x}\oplus K^0_{y,x}\right)\\
\mathcal{L}^{\mathsf{sub},0}_{y,x}(D,K) &= HW\left(S[P_{y,x}\oplus K^0_{y,x}]\right)\\
&\cdots\\
\mathcal{L}^{\mathsf{sh},9}_{y,x}(D,K) &= HW\left(C_{y,x}\oplus K^{last}_{y,x}\right)\\
\mathcal{L}^{\mathsf{add},9}_{y,x}(D,K) &= HW\left(C_{y,x}\right)
\end{aligned} \tag{2}$$

For each of these leakage models and sample indices $i$, we calculate Pearson's correlation coefficient $\rho^{o,r,i}_{y,x}$ and sort the corresponding coefficients by magnitude. We then focus on the highest-magnitude correlations for attacks on an unknown key.

*4.3.2 MixColumns.* Figure 4 shows the results of our known-key analysis for each AES operation in each round. We observe high-magnitude intermediate correlations both near the beginning and the end of the encryption process—these correlations are tightly coupled to input/output (IO) correlations, and thus not entirely useful for unknown-key intermediate correlations. More notably, we also see correlation spikes for outputs of the `AddRoundKey` and `MixColumns`

[2] 16 byte positions * (4 operations/round * 10 rounds + 1 plaintext state)

operations within the temporal region of interest. Comparatively, the `MixColumns` correlation spikes have consistently high magnitudes suggesting that Apple's implementation of **`aesmc`** leaks sensitive internal cryptographic data. It is this operation which we target in our subsequent attacks.

### 4.3.3 *Unknown-Key Correlation.*

Knowing now that we wish to target the output of `MixColumns` operations, we must decide on a suitable leakage model which can capture the correlation between $D$-derived intermediates and the trace samples $T = \{s_0, s_1, \ldots, s_{n-1}\}$. Specifically, we need a model which returns the Hamming weight of some byte from the output $M$ of some internal `MixColumns`, specified in terms of $D$ and some key guess(es) $\mathcal{K}$. Two such options are available, both of which we derive.

The first option would calculate some selected byte in the output of `MixColumns` in AES round 0, based entirely on the values of the plaintext $P$. However, due to the diffusion properties of `MixColumns` [7], each byte of this intermediate actually depends on the values of 4 bytes in $P$ and thus 4 bytes of key guesses. This can be seen by deriving the output of the target `MixColumns` (notated $M^0$) in terms of plaintext $P$ and key guesses $\mathcal{K}$, utilizing intermediate states $R, B, A$ corresponding to the outputs of the first round's `ShiftRows`, `SubBytes`, and `AddRoundKey` operations respectively:

$$\begin{aligned} M^0_{y,x} &= \sum_{i=0}^{3} X_{i-y} R^0_{i,x} = \sum_{i=0}^{3} X_{i-y} B^0_{i,x-i} \\ &= \sum_{i=0}^{3} X_{i-y} S\left[A^0_{i,x-i}\right] = \sum_{i=0}^{3} X_{i-y} S\left[\boldsymbol{P_{i,x-i}} \oplus \boldsymbol{K_{i,x-i}}\right] \end{aligned} \tag{3}$$

If we were to proceed with this hypothetical attack, we would have to define one leakage model for each of the $2^{32}$ possible combinations of $\mathcal{K}$ for each target byte $M^0_{y,x}$. This quickly becomes computationally infeasible.

The second option instead calculates some selected byte in the output of the *last* `MixColumns` operation in round 8, based on the values of the ciphertext $C$. To do so, all we need to do is invert round 9 from the known $C$ back to the output of `MixColumns` in round 8:

$$\begin{aligned} C_{y,x} &= A^{last}_{y,x} = R^9_{y,x} \oplus K^{last}_{y,x} = B^9_{y,x-y} \oplus K^{last}_{y,x} \\ &= S[A^9_{y,x-y}] \oplus K^{last}_{y,x} = S[M^8_{y,x-y} \oplus K^9_{y,x-y}] \oplus K^{last}_{y,x} \\ &\to M^8_{y,x-y} = S^{-1}[\boldsymbol{C_{y,x}} \oplus \boldsymbol{K^{last}_{y,x}}] \oplus \boldsymbol{K^9_{y,x-y}} \\ &\to M^8_{y,x} = S^{-1}[\boldsymbol{C_{y,x+y}} \oplus \boldsymbol{K^{last}_{y,x+y}}] \oplus \boldsymbol{K^9_{y,x}} \end{aligned} \tag{4}$$

This derivation, in contrast to the previous one, only depends on one ciphertext byte and two key guess bytes. We thus only need to define leakage models for $2^{16}$ possible combinations of $\mathcal{K}$ for each target byte $M^8_{y,x}$, which represents a significant speedup in computation time. For our unknown-key attacks in Section 5, we thus use the following leakage model:

$$\mathcal{L}\left(D, \mathcal{K}^{last}_{y,x+y}, \mathcal{K}^9_{y,x}\right) = HW\left(S^{-1}\left[C_{y,x+y} \oplus \mathcal{K}^{last}_{y,x+y}\right] \oplus \mathcal{K}^9_{y,x}\right) \tag{5}$$

## 5 RESULTS

In order to extract an unknown-key from EM side channel traces, we define a leakage model $\mathcal{L}(D, \mathcal{K}^{last}_{y,x+y}, \mathcal{K}^9_{y,x})$ for each byte position $(y, x)$ and key guess combination $(\mathcal{K}^{last}_{y,x+y}, \mathcal{K}^9_{y,x}) \in \{0-255\}^2$. We then collect long EM traces using our accelerated acquisition approach (Section 4.2) and preprocess them into subtraces (Section 4.2.1). Finally, using each subtrace, we calculate the Pearson correlation between each defined leakage model and subtrace data index and sort the leakage models based on peak absolute-value correlation. This sorting order then becomes the basis for our attack, under the assumption that higher correlation values indicate higher probability that the key hypothesis is correct.

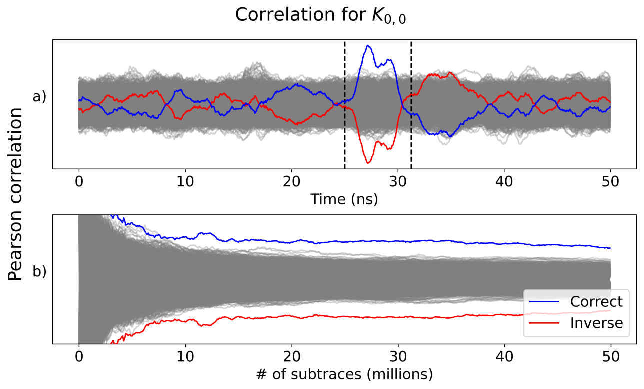

**Figure 5: Raw correlation plots from the unknown-key attack on $K_{0,0}$. The correlation for the correct guess is shown in blue, the inverse guess in red, and incorrect guesses in gray. *(a)* Correlation plotted against time. The leaky temporal region is highlighted. *(b)* Correlation plotted against number of traces. This key position begins leaking at 10 million subtraces.**

## 5.1 Correlation Plots

Figure 5 shows an example correlation trace for $K_{0,0}$, as well as the correlation coefficient's evolution across number of traces. In the top subfigure, we highlight the temporal region of interest for which we see the maximum correlation for the correct key guess—note that this region closely lines up with the peak in round 8 of Figure 4. The bottom subfigure shows the correlation coefficient's evolution as the number of subtraces increases. We can see that, for both the correct and the inverse (Section 5.1.1) key guesses, the absolute correlation magnitude consistently stays above the magnitudes for incorrect key guesses. This indicates that correct key guesses are distinguishable, and thus lead to key extraction for this byte position.

### 5.1.1 *Inverse guesses.*

For some byte positions ($K_{0,2}, K_{0,3}, K_{2,1}, K_{3,3}$ in Figure 6), we see a consistently strong false positive which has a similar, high absolute correlation as the correct key guess. We find that this false positive is always due to the leakage model where $\mathcal{K}^{last}_{y,x+y} = K^{last}_{y,x+y}$ and $\mathcal{K}^9_{y,x} = \overline{K^9_{y,x}}$, *i.e.*, the last round key guess is correct, but the second-to-last round key guess is the bitwise `NOT` of the correct guess. We thus name this false positive the *inverse* guess.

When comparing the correlation plots for the inverse guess to the correct guess, we see inverse correlations (Figure 5). That is, a strong *positive* correlation for the correct key guess will be reflected as a strong *negative* correlation for the inverse guess. We note that within our temporal region of interest, the correct key guess always has the positive-magnitude correlation. Therefore, the presence of the single inverse guess does not harm our attack since it is always possible to distinguish it from the correct guess.

## 5.2 Per-Position Results

Figure 6 shows the results of an unknown-key attack as described above. For each key index, and for various amounts of accumulated subtraces, we plot the *guessing entropy* [26] at each byte position. Recall that each byte index will have $256 * 256 = 2^{16}$ leakage models associated with it. After sorting by Pearson correlation, we find the index $\mathcal{I}$ of the correct leakage model (where $\mathcal{K}^{last}_{y,x+y} = K^{last}_{y,x+y}$ and $\mathcal{K}^9_{y,x} = K^9_{y,x}$) and take $\log_2 \mathcal{I}$ as an approximation for the key byte's remaining security level. This models a brute-force adversary who simply searches the sorted leakage model list in ascending order.

Our results show that, although our specific attack model initially doubles the bits of security at each byte position, the correlation-sorting approach quickly leads to successful key recovery for many byte positions. This effect is perhaps most significant for $K_{3,1}$, where we successfully recover the key after correlating only 5 million

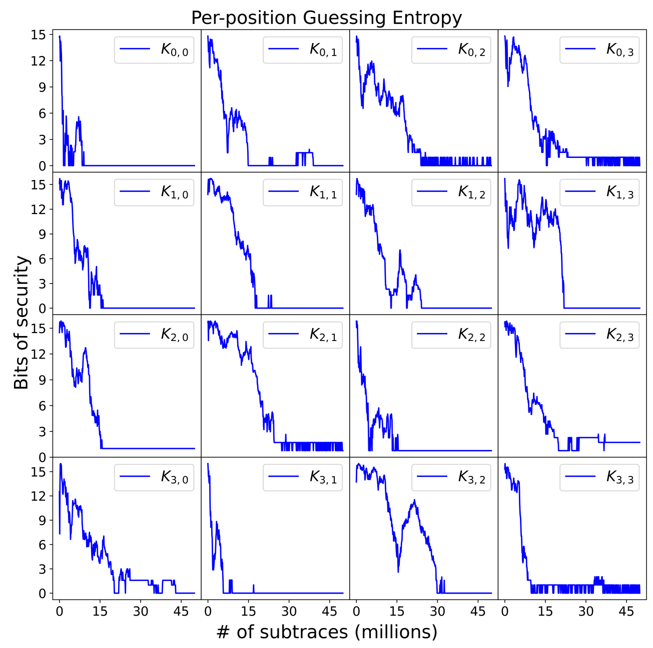

**Figure 6: Guessing entropy evolution for each byte position, as a function of number of side channel traces. Note that our maximum guessing entropy is 16 bits of security for each position, rather than a brute force attack's 8 bits. This is because we must guess 16 bits of key to correlate each 8-bit position. For most key positions, this effect is quickly overcome.**

subtraces. Even for more resilient byte positions such as $K_{3,0}$ and $K_{3,2}$, correlating 30 million subtraces leads to successful key extraction.

## 5.3 Overall Results

For our final guessing entropy calculation, we model an adversary similar to the adversary in Figure 6. However, this adversary additionally utilizes the inverse guess detection strategy described in Section 5.1.1. This allows us to fully extract the entire 128-bit secret key after correlating approximately 43 million subtraces. We note that even if the adversary does not try to detect the inverse guesses, the worst-case guessing entropy of the attack is only 16 bits (or 65536 key combinations), which is trivially brute-forceable by even slow hardware.

## 6 CONCLUSION

In this work, we show the first side channel attack on a RISC-style AES instruction set extension. We find that an EM-based side channel attack targeting the output of the ARM `aesmc` instruction can successfully extract secret keys (Section 4). This attack requires between 5-30 million traces to succeed, which corresponds to an operation on approximately 80-480 megabytes of data. We justify these arguably high data requirements by explicitly defining a victim model (Section 3) which naturally allows for such data collections—when combined with our advanced preprocessing approach (Section 4.2), such attacks move from academic significance to practical significance. This highlights the need for side channel leakage evaluations/defenses for real-world devices and production software.

## 7 RESPONSIBLE DISCLOSURE

We took the necessary steps for ethical disclosure. The findings have been reported to Apple Inc. on November 8th, 2021, which is prior to publishing our paper in a public medium.

## 8 ACKNOWLEDGEMENTS

We thank anonymous reviewers for their feedback on the paper. This research is funded in part by NSF Grants No. 1850373 and 1943245.

## REFERENCES

[1] Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, and Ronen Zohar. 2010. Breakthrough AES performance with intel AES new instructions. *White paper, June* (2010), 11.

[2] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. 2015. DPA, bitslicing and masking at 1 GHz. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 599–619.

[3] Daniel J Bernstein. 2005. Cache-Timing Attacks on AES. (2005). http://palms.ee.princeton.edu/system/files/Cache-timing+attacks+on+AES.pdf

[4] Eli Biham. 1997. A fast new DES implementation in software. In *International Workshop on Fast Software Encryption*. Springer, 260–272.

[5] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation Power Analysis with a Leakage Model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.

[6] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. 2002. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 13–28.

[7] Joan Daemen and Vincent Rijmen. 1999. AES proposal: Rijndael. (1999).

[8] Joan Daemen and Vincent Rijmen. 2002. *The design of Rijndael*. Vol. 2. Springer.

[9] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *Cryptographic Hardware and Embedded Systems — CHES 2001*, Çetin K. Koç, David Naccache, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 251–261.

[10] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. 2016. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1626–1638.

[11] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.

[12] Gregor Haas, Seetal Potluri, and Aydin Aysu. 2021. iTimed: Cache Attacks on the Apple A10 Fusion SoC. *IACR Cryptol. ePrint Arch.* 2021 (2021), 464.

[13] ARM Holdings. 2021. Arm® Architecture Reference ManualArmv8, for Armv8-A architecture profile. (2021).

[14] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. 2011. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering* 1, 4 (2011), 293.

[15] Apple Inc. 2020. Apple corecrypto Library. (2020). https://developer.apple.com/security/

[16] Apple Inc. 2020. Apple Platform Security Guide. (Spring 2020). https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

[17] Emilia Käsper and Peter Schwabe. 2009. Faster and timing-attack resistant AES-GCM. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 1–17.

[18] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology — CRYPTO' 99*, Michael Wiener (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 388–397.

[19] David P Leech, Stacey Ferris, John T Scott, et al. 2019. The economic impacts of the advanced encryption standard, 1996–2017. *Annals of Science and Technology Policy* 3, 2 (2019), 142–257.

[20] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[21] Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. 2021. Let's take it offline: Boosting brute-force attacks on iPhone's user authentication through SCA. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 496–519.

[22] Jake Longo, Elke De Mulder, Dan Page, and Michael Tunstall. 2015. SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 620–640.

[23] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Cryptographers' track at the RSA conference*. Springer, 1–20.

[24] Sami Saab, Pankaj Rohatgi, and Craig Hampel. 2016. Side-channel protections for cryptographic instruction set extensions. *Cryptology ePrint Archive* (2016).

[25] T Schnneider and A Moradi. 2015. Leakage assessment methodology—A clear roadmap for side-channel evaluations. *Proc. Cryptographic Hardware Embedded Syst* (2015), 495–513.

[26] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. 2009. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *Advances in Cryptology - EUROCRYPT 2009*, Antoine Joux (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 443–461.

[27] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. 719–732.