

# Cache Attacks and Countermeasures – the Case of AES

Elena Frank, Frederik Safner & Lukas Kaibel

July 5, 2023

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Synchronous Known-Data Attacks</b>	<b>3</b>
<b>3</b>	<b>Asynchronous Attacks</b>	<b>4</b>
<b>4</b>	<b>Countermeasures</b>	<b>5</b>
<b>5</b>	<b>Conclusion</b>	<b>6</b>

# 1 Introduction

The paper discusses cache attacks as a type of side-channel attacks that leverage cross-process information leaking through CPU caches.

The attacks are possible due to how *set-associative memory caches* are implemented. Their structure limits the mapping of memory addresses to the cache in the sense a) always full memory blocks are cached, and b) that a memory block can only be cached in specific cache lines.

The attacked cipher is the AES cipher, where the attacks leverages how it is implemented in performance-oriented software. Concrete, the implementation pre-computes Lookup-tables. These tables are then used to realize the algebraic Shift-Rows, Mix-Columns and Sub-Bytes operations needed in each round of the AES computation.

The paper describes multiple attacks where a process can obtain information about another process' memory access patterns by leveraging said implementation details of the the memory caches and AES implementation. Finally, the paper also discusses countermeasures for such attacks.

## 2 Synchronous Known-Data Attacks

### 3 Asynchronous Attacks

Synchronous attacks require that the attacker can interact with the encryption code. For asynchronous attack this is not necessary: a process doesn't have to interact with encryption program and can still obtain information simply by running on the same CPU.

It is assumed that multithreading is supported and that the attacker process runs in parallel with the victim. The attacker can analyze the frequency score of the victims memory accesses, i.e. how often it accesses a memory location. The preliminary is that the attacker's and the victim's used memory blocks map to the same cache lines. The attacker measures the time it takes to retrieve memory blocks when it runs alone, vs when it runs in parallel with the victim. Depending on how often the victim access its memory blocks, the attacker's process gets cache misses when loading its blocks, which shows through an increased access time. By measuring over a larger number of access, the frequency score for the victim can be obtained.

This frequency score for each AES Lookup Table can be used together with the frequency information about the encrypted text (e.g. that if it's an english text, it is very likely that a bytes has there highest nibble set to 6, which represent a letter between 'a' and 'p') to obtain information about the highest nibble in some key bytes. The paper described that through this method, 45.66 bits of the secret key could be obtained.

It also proposes that if additionally the order of accesses could be analyzed by the attacker, even more bits can be obtained. Furthermore, it describes an way to remove the limitation that the host system must be multithreaded: by exploiting interrupt mechanism an attacker can pause the encryption process and analyze the state of the cache (again through loading memory blocks and measuring cache misses) again as discussed. For processors with multiple layers of caching i.e. where L1 caches are private to a core, the paper claims that the attacks may still be possible if the cryptographic code to L2 / L3 caches that are shared among cores.

## 4 Countermeasures

Several methods can be considered to mitigate information leakage in cryptographic systems, each with unique trade-offs:

**Avoiding Table Lookups:** Techniques include using logical operations instead of table lookups, using bitslice implementations to execute multiple encryptions simultaneously through vectorization, or placing lookup tables in registers instead of cache. These methods may reduce performance or be architecture-dependent.

**Alternative Lookup Tables:** By using smaller or combined lookup tables, the chance of a given memory block not being accessed during encryption is reduced. This method is more resistant to synchronous attacks but still vulnerable to asynchronous attacks, and is most effective when used with other countermeasures.

**Data-Oblivious Memory Access Pattern:** This involves altering memory access patterns to be independent of data. Techniques include reading all entries from relevant tables, memory shuffling or permutation, or adding spurious memory accesses. All these methods result in some performance slowdown.

**Algorithmic Masking Techniques:** These randomize data-dependent operations. Current AES masking methods are either insecure or significantly slower.

**Cache State Normalization and Process Blocking:** This method involves normalizing the cache state before and after encryption and is effective against synchronous attacks but not asynchronous ones.

**Disabling Cache Sharing or Caching Mechanism:** Disabling cache sharing can prevent cache state effects from crossing process boundaries but may require extensive resources. A more drastic measure is disabling the CPU's caching mechanism altogether, which significantly affects performance. A more feasible alternative is the "no-fill" mode, which requires privileged mode.

**Dynamic Table Storage:** This method involves randomizing table lookup locations or changing table order multiple times during encryption. The effectiveness of this method depends on the architecture.

**Hiding Timing Information:** This involves adding random delays or normalizing all timings to a fixed value, but it doesn't provide complete protection against certain types of attacks.

**Selective Round Protection:** This method focuses on protecting specific rounds of encryption while leaving the internal rounds unprotected.

**OS-Level Support:** Some countermeasures may require OS-level support or privileged operations. One solution could be to provide secure execution of cryptographic primitives as OS services. An alternative is providing a secure execution facility to user processes.

These methods represent various trade-offs between security and performance. Due to the complexity and architecture-dependency, secure implementation requires a delicate balance and no one-size-fits-all solution exists.

## 5 Conclusion