

Cache attacks on secret key cryptosystems and effectiveness of defenses

By: Brian Sanders

[What will be covered]

- What are cache attacks
- Types of cache attacks
- What is affected
- Proposed defenses



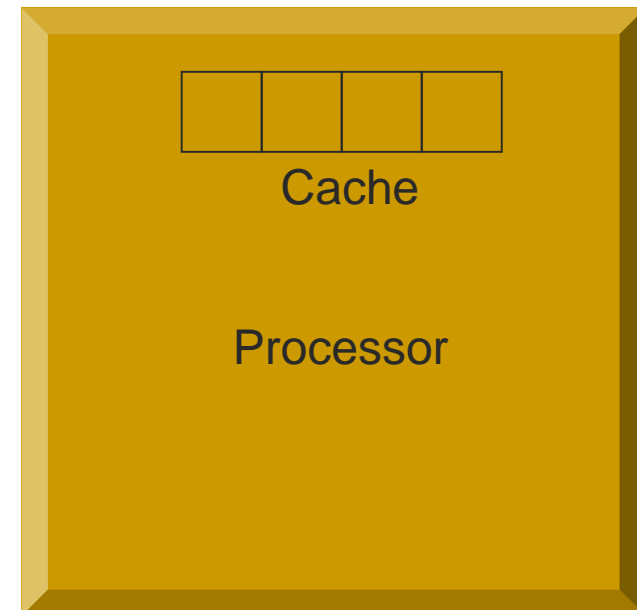
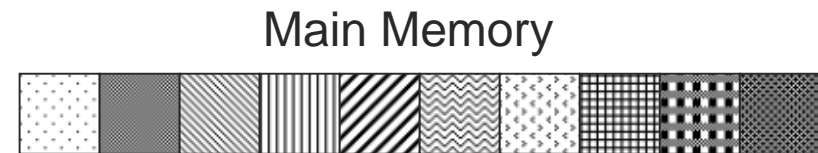
What are Cache Attacks?

[What are cache attacks]

- Method of side channel attack against cryptosystems
- Two basic categories of cache attacks
 - Timing attacks
 - Trace driven attacks
- Uses a combination of software implementation and cache behavior

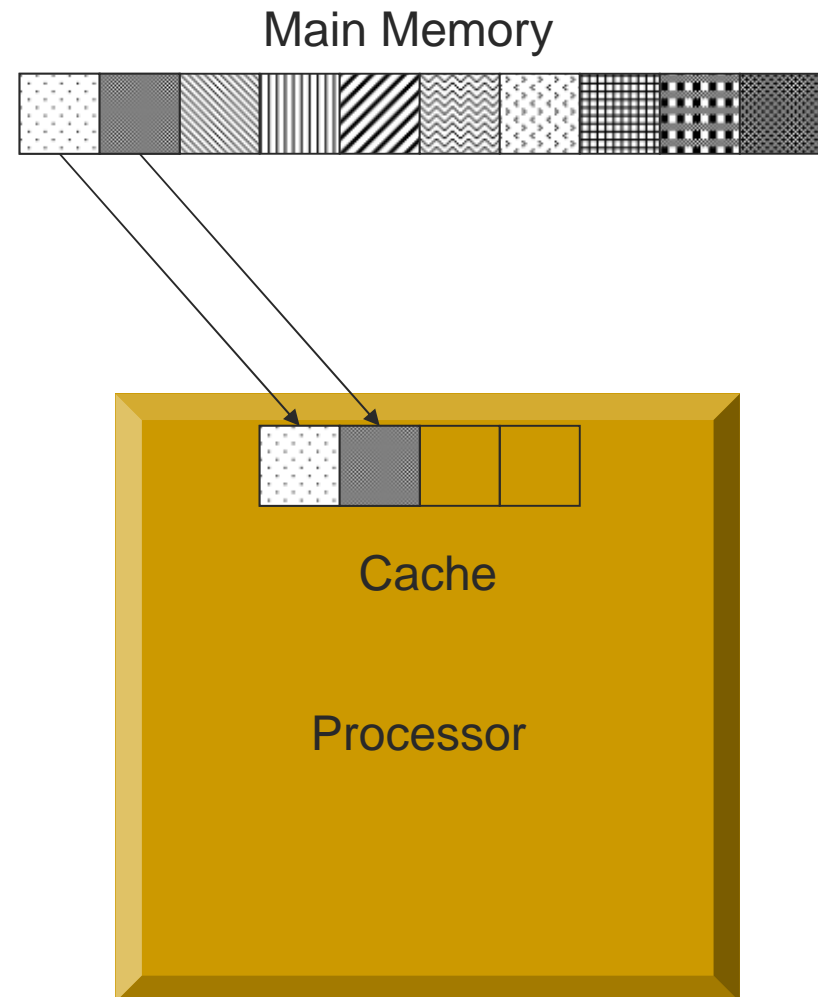
[Understanding caching]

- When data from main memory is needed by the processor, it is fetched and stored in the cache.
- Data in the cache can be accessed much more quickly than data in main memory.
- Cache size is limited, so many memory locations map to the same cache location.
- As conflicting mappings are accessed, cache lines are replaced.



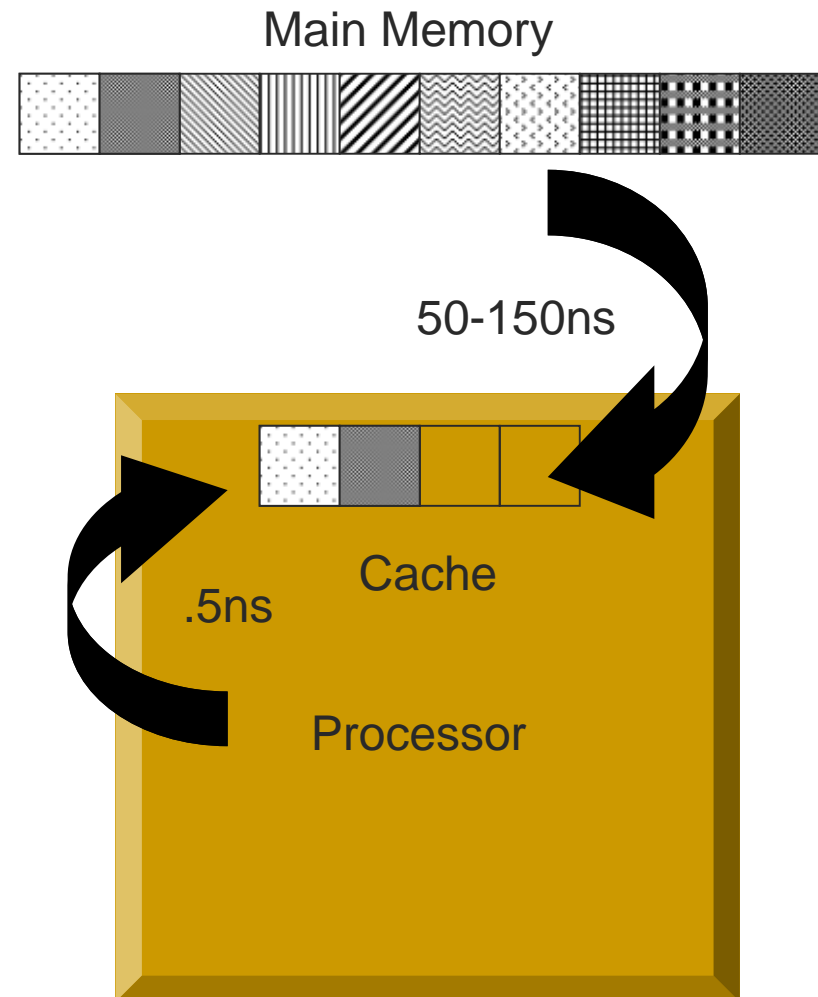
[Understanding caching]

- **When data from main memory is needed by the processor, it is fetched and stored in the cache.**
- Data in the cache can be accessed much more quickly than data in main memory.
- Cache size is limited, so many memory locations map to the same cache location.
- As conflicting mappings are accessed, cache lines are replaced.



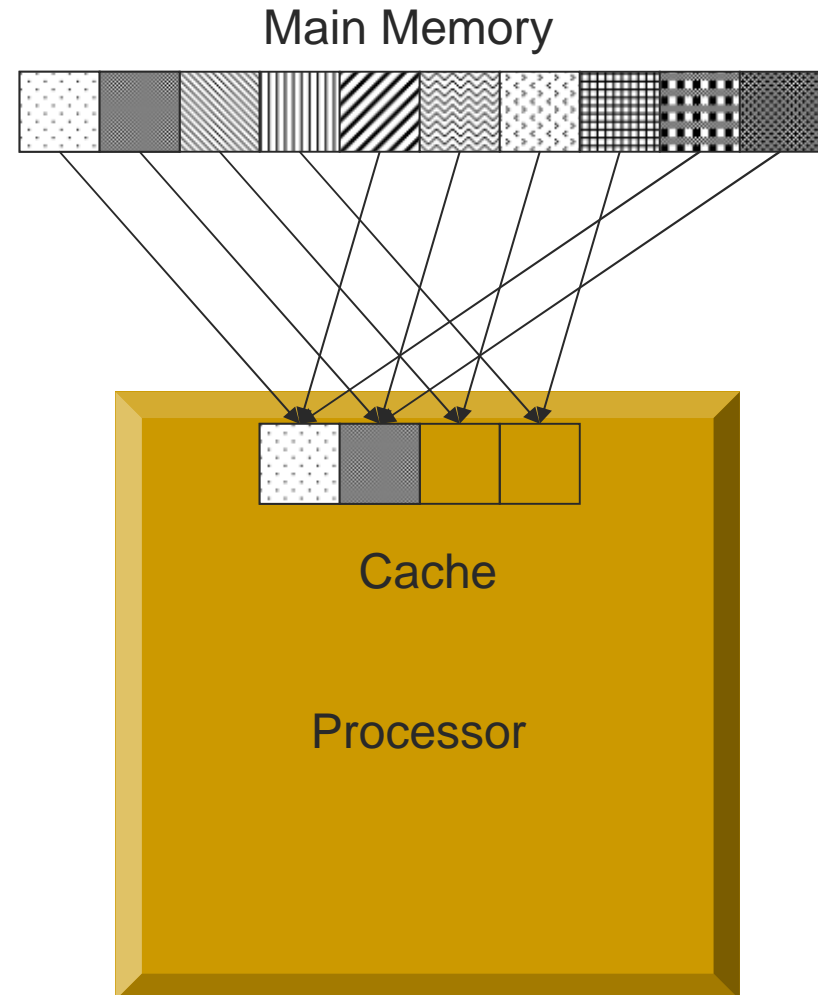
[Understanding caching]

- When data from main memory is needed by the processor, it is fetched and stored in the cache.
- **Data in the cache can be accessed much more quickly than data in main memory.**
- Cache size is limited, so many memory locations map to the same cache location.
- As conflicting mappings are accessed, cache lines are replaced.



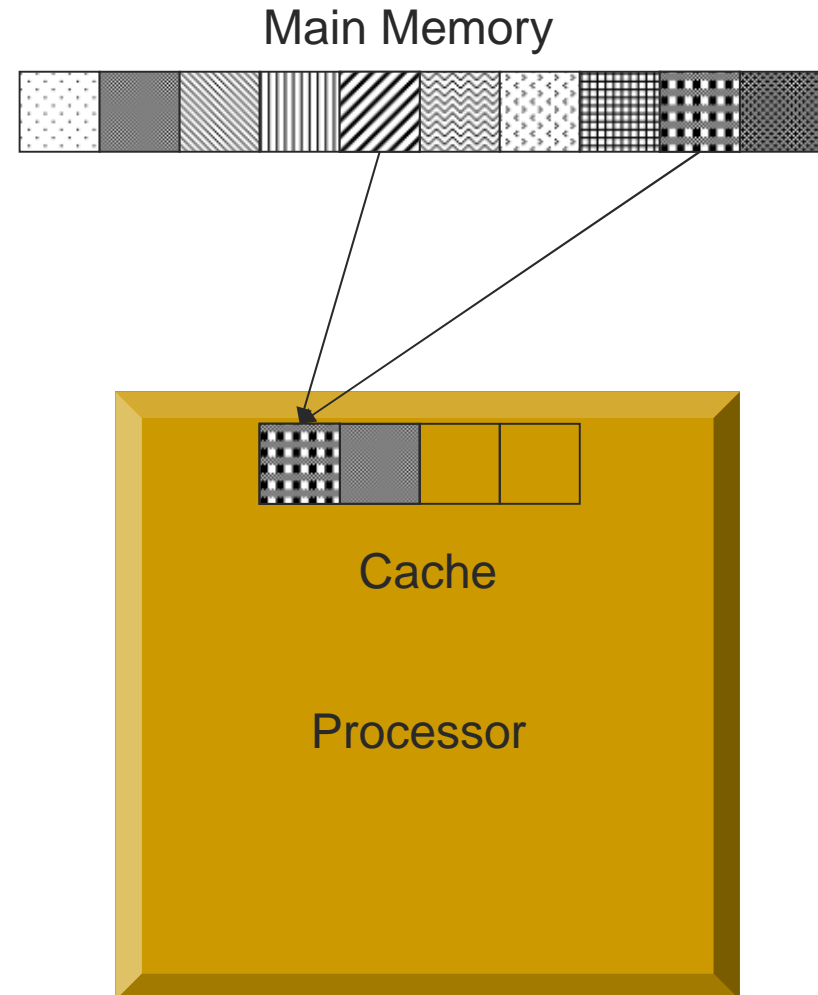
[Understanding caching]

- When data from main memory is needed by the processor, it is fetched and stored in the cache.
- Data in the cache can be accessed much more quickly than data in main memory.
- **Cache size is limited, so many memory locations map to the same cache location.**
- As conflicting mappings are accessed, cache lines are replaced.



[Understanding caching]

- When data from main memory is needed by the processor, it is fetched and stored in the cache.
- Data in the cache can be accessed much more quickly than data in main memory.
- Cache size is limited, so many memory locations map to the same cache location.
- **As conflicting mappings are accessed, cache lines are replaced.**





Types of Cache Attacks

[Types of cache attacks]

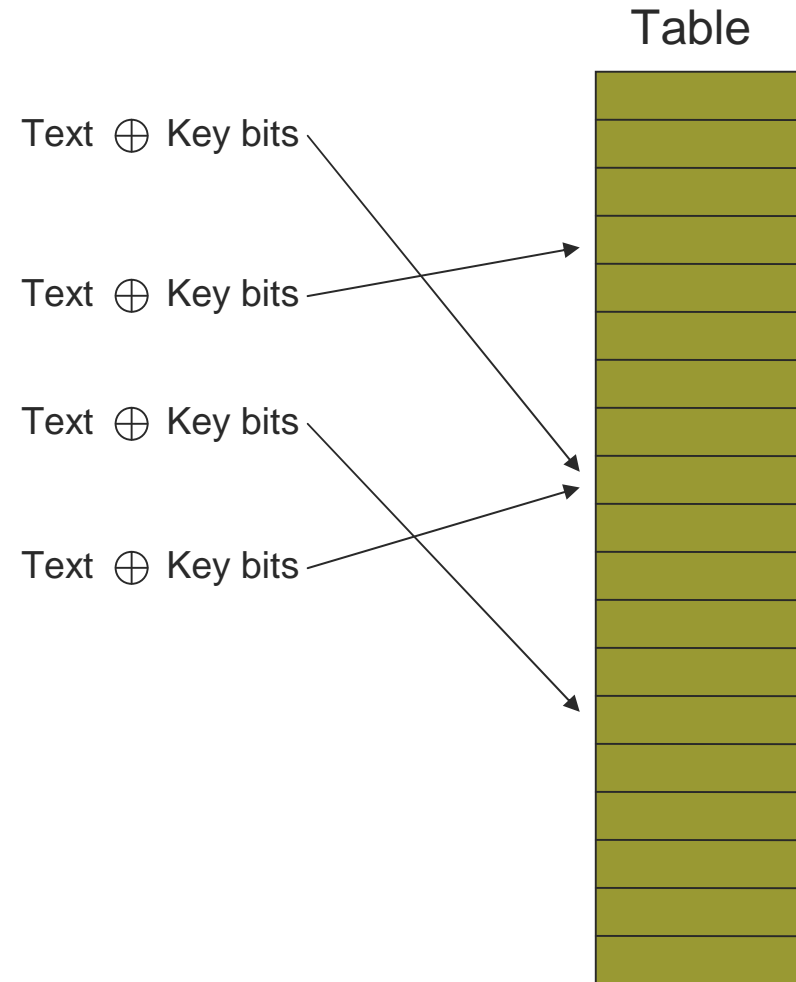
- Timing based attacks
 - Uses overall time of a process to determine cache locations accessed.
 - Attack is not very complex to initiate, but the mathematical analysis is much more complex.
- Trace driven attacks
 - Uses timing differences of specific locations in cache.
 - Very complex attacks, but simpler mathematical analysis.
- Both rely on large time difference between data which is in the cache, and data which is not.

[Timing attacks]

- Generic timing attack
 - Profiling
 - known key
 - determine a timing baseline for the machine with large numbers of test packets.
 - Attack
 - unknown key
 - Repeat large numbers of test packets, and compare to baseline generated during profiling to determine key bits.
- Specialized timing attacks
 - Focuses on qualities of a specific algorithm
 - Final round in AES
 - Algorithm specific, but produces results with less data and possibly with out a profiling phase.

Cache hits and AES encryption time

- AES implements large tables instead of multiple calculations
- In the first round, the indexes used to perform lookups into this table, are products of the key and the input text.
- An attacker can select the plain text, and therefore affect which positions in the table are accessed.
- Using timing characteristics to determine which location in the table was accessed, an attacker can find key bits used.



[Trace attacks against AES]

- Uses the fact that the cache is shared to gain information about cache usage of another process.
- Determining cache access directly requires much less time and data than timing attacks.
- Two proposed methods for determining cache usage of another process
 - Evict + Time
 - Prime + Probe

[Determining cache access]

- Evict + time:
 - Encrypt a known message
 - Evict specific cache location by accessing your own memory.
 - Re-encrypt and compare time difference.
- Prime + Probe:
 - Fill cache with attackers data
 - Initiate encryption of a know message.
 - Check cache for attackers original data.
- A small number of tests can quickly determine if the access was due to rounds later in the algorithm.



What is Affected?


Cipher analysis

Cipher	Timing attacks	Trace attacks
DES/3DES	Yes	Yes
Blowfish	Yes	Yes
RC4	No	No
Mars	Yes	Yes
RC6	No	No
Twofish	Yes	Yes
Serpent (bit slice)	No	No
AES (Rijndael)	Yes	Yes

Library analysis

Library	Timing defenses	Trace defenses
OpenSSL	Yes	No
Crypto++	Yes	No

- **OpenSSL**
 - Defenses contained in the assembly language code for AES.
 - Implements “compressed” lookup tables, and pre-fetches them.
 - Only in CBC mode
 - Pentiums saw a 30% decrease, Pentium III 10% decrease
 - Avoids aliasing between stack frame, s-boxes, and key schedule
- **Crypto++**
 - Compresses the tables for the first and last round of AES
 - Pre-loads the smaller before encryption
 - Attempts to provide additional instructions to prevent execution before the tables have been loaded.



Proposed Defenses

[Proposed Defenses]

■ Software Defenses

- Can be implemented quickly
- No large cost involved
- All currently suggested methods will cause some form of performance degradation
- Can be complicated and difficult to write correctly.

■ Hardware Defenses

- Can provide security with out decreased performance
- Avoids additional restrictions on cryptographic algorithms.
- Can not be quickly applied to currently running systems
- Very large cost to produce new hardware



Constant Time Code

Constant time code considerations

- Align relevant pieces of code in memory, avoiding evictions from cache during execution.
 - Stack, Tables, and key scheduling need to be aligned in such a way as to prevent evictions.
 - Complicated architecture specific code
- Pre-load code after any non-relevant piece of code is run.
 - Decreases performance

[Constant time code considerations]

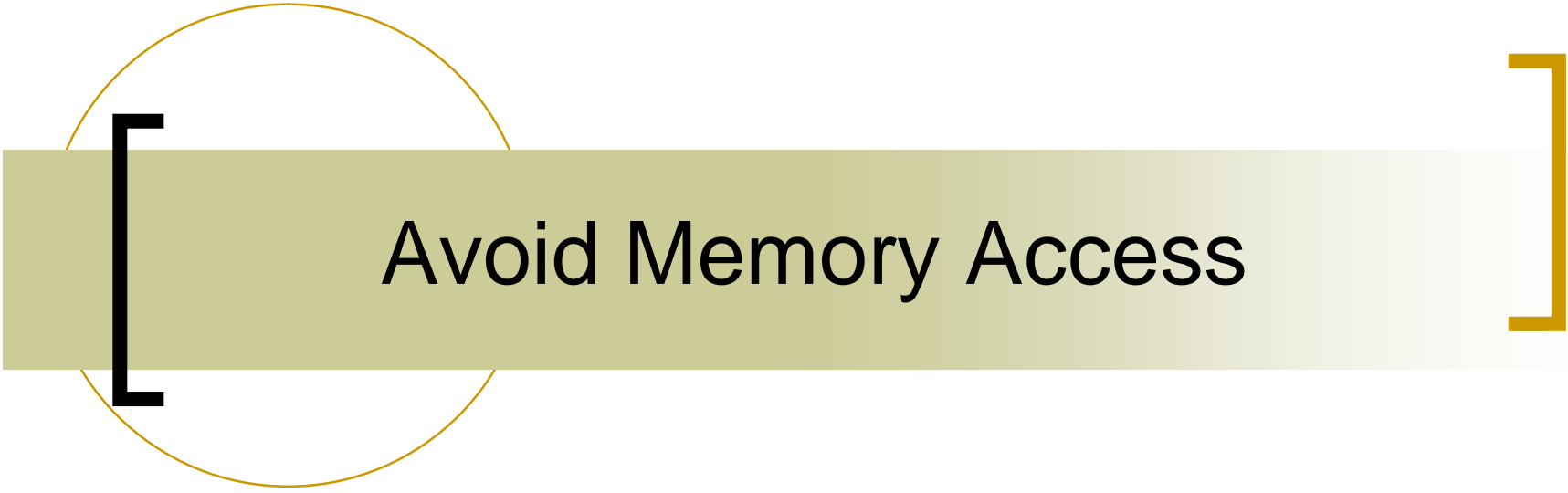
- Interrupts must be considered, but this requires operating systems assistance.
- Architecture specific variances



Operating System Assistance

Operating System Assistance

- Provide cryptographic primitives as operating system services.
 - Requires OS to support each primitive.
- Provide guarantees on execution of sensitive code.
 - Disable interrupts during execution.
 - Can be detrimental to the system
 - Disable cache sharing, forcing a cache flush on each context switch.
 - Performance degradation to both encryption processes as well as other processes on the same machine.



Avoid Memory Access

[Avoiding memory access]

- Algorithms which do not rely on s-box type implementations are not susceptible to cache based attacks.
 - Serpent and RC6 both performed poorly compared to Rijndael in the software comparisons.
- Replacing s-box lookups with logic can remove this attack all together
 - AES with out the optimized tables performs poorly
 - Using a bitslice implementation
 - Can be architecture dependent
 - Requires sufficient input data to reach full throughput.
 - Some implementations show as little as a 5% speed decrease from current optimized implementations of AES.



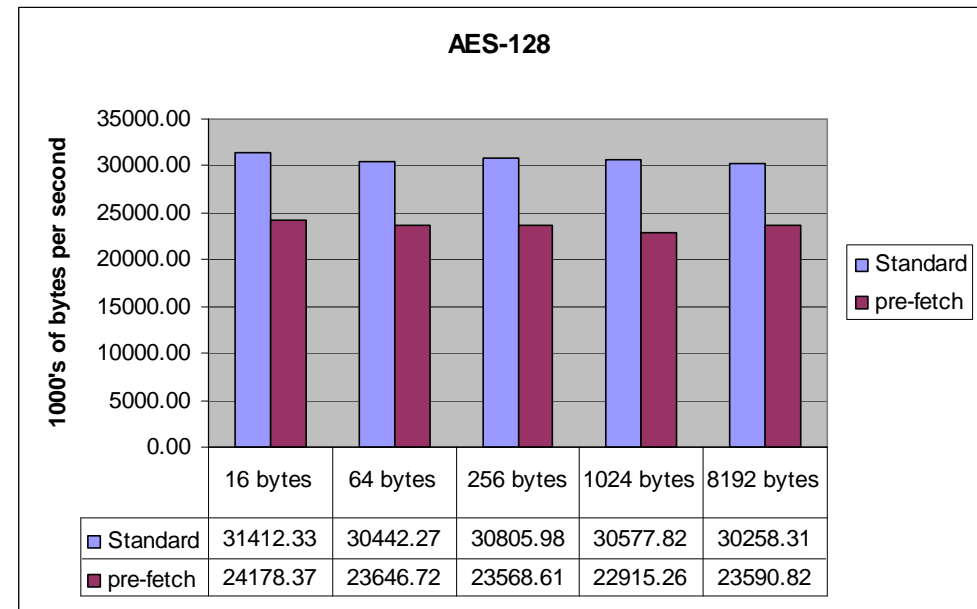
Data oblivious memory access
patterns


[Data-oblivious memory access patterns]

- Allow table usage, but remove their dependency on input data.
 - RC4 uses a large table, but it does not depend on the input data for lookups
- Run a dummy encryption at the same time, generating a memory access profile which is not constant
 - Noise can be filtered with larger sample sizes.
- Access one location from each cache line, pre-fetching the tables.
 - Decreases performance due to extra loads.
 - Interrupts can still allow code execution after pre-fetching.

Data-oblivious memory access patterns

- Comparison of performance of OpenSSL with and without pre-fetching
 - Pre-fetch was only executed before encryption began, not before each round of encryption.
 - Average of about 20% decrease in speed.





Dynamic Table Storage

[Dynamic Table Storage]

- Store the tables in multiple memory locations, or move the table multiple times during execution.
 - Additional cache misses will generate a performance decrease
- Permute the order of the elements in the table during encryption, breaking links to cache locations.
 - May require an external source of entropy for mixing
 - Additional cache misses will generate performance decrease



Partitioned Cache

[Partitioned cache]

- Separates the cache into partitions per process.
 - Can cause poor usage of cache
- Access to the cache would have a partition identifier added to them to indicate what can be accessed
 - Separation between processes provides protection against trace driven attacks.
 - Does not protect against timing attacks due to processes causing evictions of their own space.
- Additional instructions would need to be added to the ISA to manage the partitions



Partition Locked Cache

Partition locked cache

- Using a lock and an ID bit, cache lines are locked to specific processes as needed.
 - Does not require locking of space for processes that do not require the protection.
 - Avoids wasted cache space by only locking lines in use.
- Two methods of locking a cache line:
 - New instructions to provide the compiler or programmer the ability to lock sensitive sections
 - Requires program to have knowledge of these capabilities
 - Lock cache when accessing locked pages/segments in memory.
 - This method provides backwards compatibility with older code

[Partition locked cache]

- Only requires the addition of 3 fields
 - 1 bit lock field to the cache line
 - N bit ID field to the cache line
 - 1 bit field to the TLB entry
- Performance decreases drastically for direct mapped caches, or for smaller caches where locked sections cause many conflicts.
- In 2-way caches with large enough space to hold the locked code, the speed is equivalent to standard cache speeds.



Random Permutation Cache

[Random Permutation Cache]

- Memory-cache mappings are randomized by a permutation tables
 - A number of PT can be added for each protected application
- Protection fields are added to the cache.
 - When two non-protected cache lines conflict, normal replacement methods are used.
 - When protected memory conflicts, it randomly replaces a line in the cache and swaps the mappings in the PT.
- Repeatedly running the same encryption will generate conflicts with random lines of cache, preventing any statistical analysis of the time take to do the encryption.

[Random Permutation Cache]

- Similar to the Partition Locked Cache scheme
3 fields are required to be added
 - 1 Protection bit added to each cache line
 - N ID field added to each cache line
 - 1 bit added to TLB entries
- Simulated results have shown less than 2% performance decrease over all cache sizes.
- Power increase from additional hardware was around 10% above a traditional cache system.

Defense Summary

