Evict+Spec+Time:Exploiting Out-of-Order Execution to Improve Cache-Timing Attacks

Chitchanok Chuengsatiansup

Joint work with Shing Hing William Cheng, Daniel Genkin, Dallas McNeil, Toby Murray, Yuval Yarom, and Zhiyuan Zhang

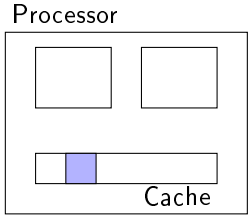
Cache-Timing Attacks

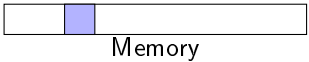
Cache-Timing Attacks

- Timing depends on where data resides

Cache-Timing Attacks

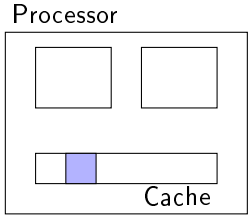
- Timing depends on where data resides

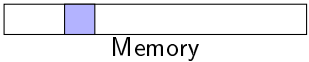




Cache-Timing Attacks

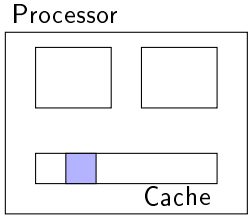
- Timing depends on where data resides
- Check if it is in the cache

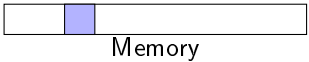




Cache-Timing Attacks

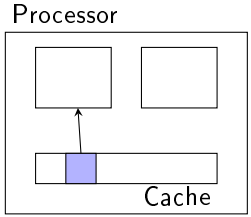
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache

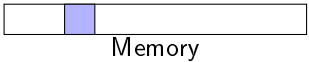




Cache-Timing Attacks

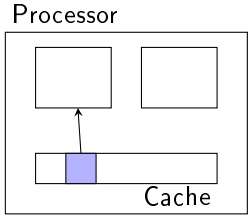
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache

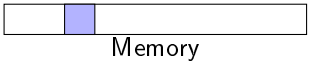




Cache-Timing Attacks

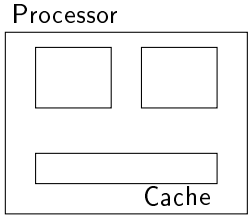
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache → fast

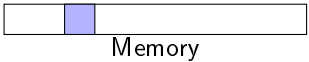




Cache-Timing Attacks

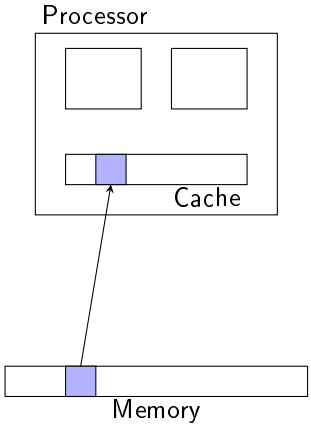
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache → fast
  - cache miss: get data from memory





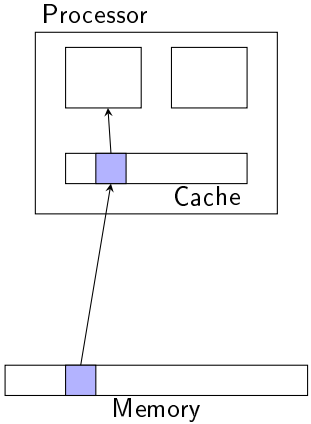
Cache-Timing Attacks

- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache → fast
  - cache miss: get data from memory



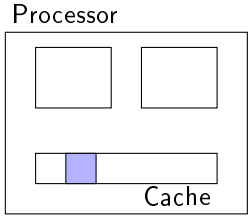
Cache-Timing Attacks

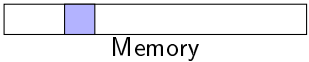
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache → fast
  - cache miss: get data from memory → slow



Cache-Timing Attacks

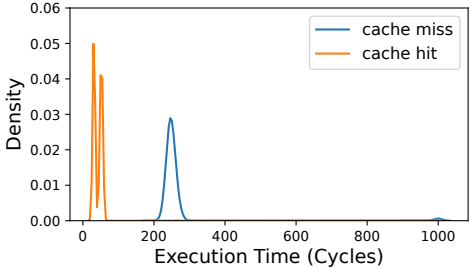
- Timing depends on where data resides
- Check if it is in the cache
  - cache hit: serve data from cache → fast
  - cache miss: get data from memory → slow
- Latency of hit vs miss is distinguishable



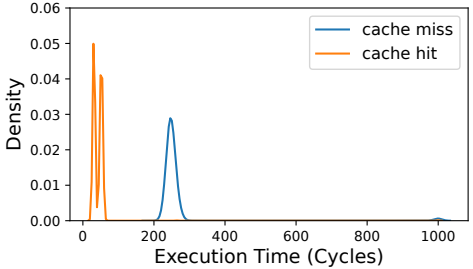


Cache Hit vs Miss Latency

Cache Hit vs Miss Latency

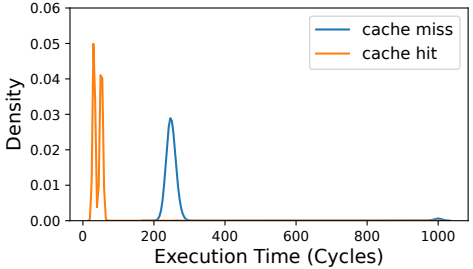


Cache Hit vs Miss Latency



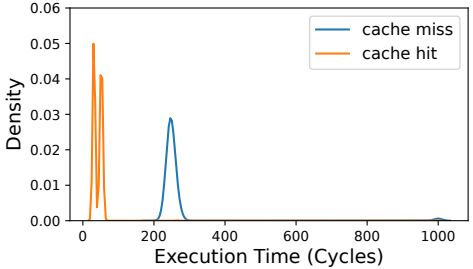
- Measure single memory access

Cache Hit vs Miss Latency



- Measure single memory access
- What about attacks with long code?

Cache Hit vs Miss Latency



- Measure single memory access
- What about attacks with long code?
  - Ex. Evict+Time [OST06]

Evict+Time [OST06]

Evict+Time [OST06]

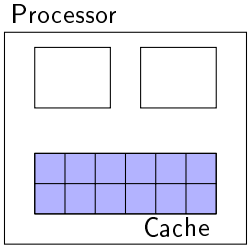
- Steps:

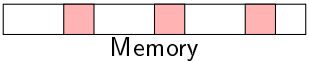
Evict+Time [OST06]

- Steps:
  1. **Evict:** victim's data from cache

Evict+Time [OST06]

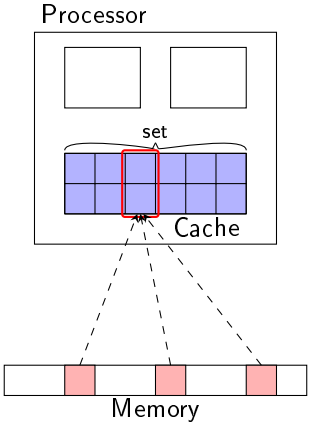
- Steps:
  1. **Evict:** victim's data from cache





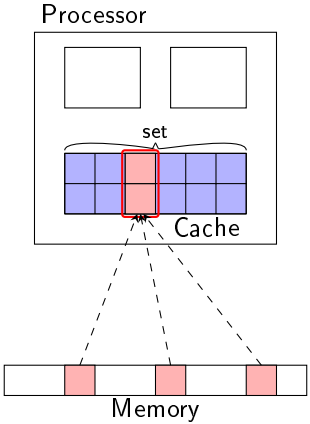
Evict+Time [OST06]

- Steps:
  1. **Evict:** victim's data from cache



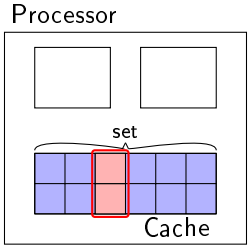
Evict+Time [OST06]

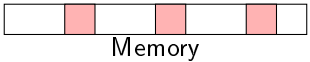
- Steps:
  1. **Evict:** victim's data from cache



Evict+Time [OST06]

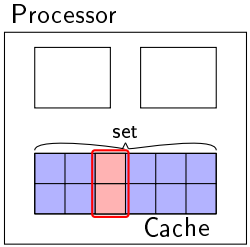
- Steps:
  1. **Evict:** victim's data from cache
  2. **Time:** victim's execution

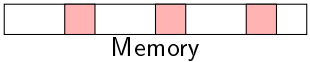




Evict+Time [OST06]

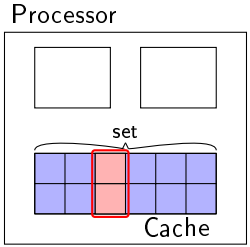
- Steps:
  1. **Evict:** victim's data from cache
  2. **Time:** victim's execution
- Measure latency of **multiple** instructions

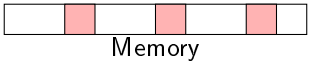




Evict+Time [OST06]

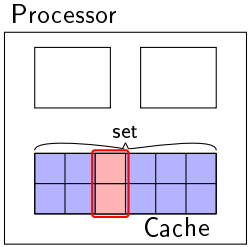
- Steps:
  1. **Evict:** victim's data from cache
  2. **Time:** victim's execution
- Measure latency of **multiple** instructions
- Why does it work?

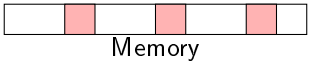




Evict+Time [OST06]

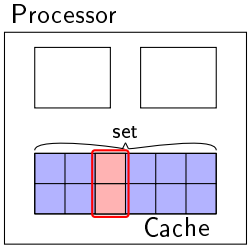
- Steps:
  1. **Evict:** victim's data from cache
  2. **Time:** victim's execution
- Measure latency of **multiple** instructions
- Why does it work?
  - Any miss in victim's execution → slow

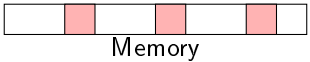




Evict+Time [OST06]

- Steps:
  1. **Evict:** victim's data from cache
  2. **Time:** victim's execution
- Measure latency of **multiple** instructions
- Why does it work?
  - Any miss in victim's execution → slow
  - Latency is additive





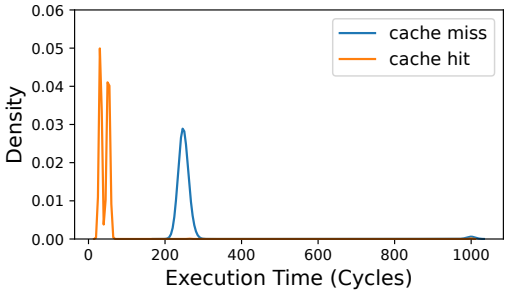
Cache Latency and Parallel Execution

Cache Latency and Parallel Execution

```
a = listhead->nextnode->nextnode;
b = *ptr
```

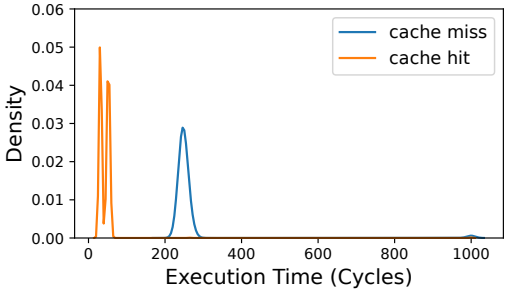
Cache Latency and Parallel Execution

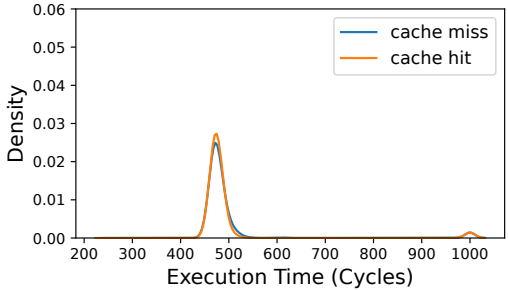
```
a = listhead->nextnode->nextnode;
b = *ptr
```



Cache Latency and Parallel Execution

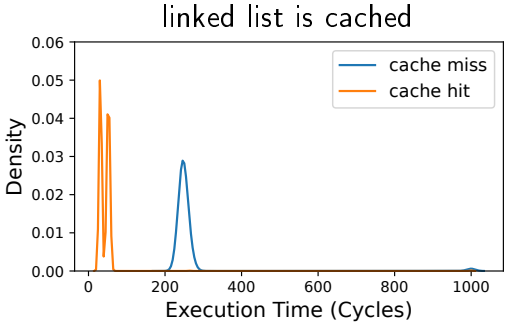
```
a = listhead->nextnode->nextnode;
b = *ptr
```

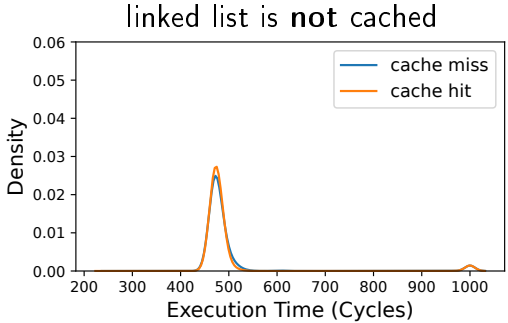




Cache Latency and Parallel Execution

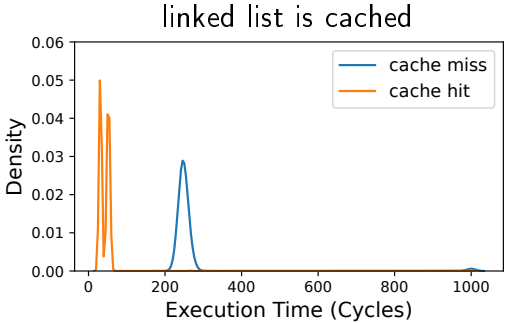
```
a = listhead->nextnode->nextnode;
b = *ptr
```

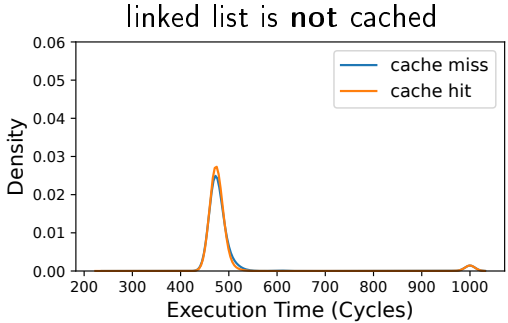




Cache Latency and Parallel Execution

```
a = listhead->nextnode->nextnode;
b = *ptr
```





Latency is **not** additive because processors are superscalar

Observations

Observations

- Parallel execution causes latency not additive

Observations

- Parallel execution causes latency not additive
- Out-of-order execution allows reordering instructions
  - instructions are not necessarily executed in the program order

Observations

- Parallel execution causes latency not additive
- Out-of-order execution allows reordering instructions
  - instructions are not necessarily executed in the program order
- Cache miss is **not** always slower than cache hit, if:
  - there are enough instructions to bring forward
  - total latency of those instructions is at least that of cache miss

Observations

- Parallel execution causes latency not additive
- Out-of-order execution allows reordering instructions
  - instructions are not necessarily executed in the program order
- Cache miss is **not** always slower than cache hit, if:
  - there are enough instructions to bring forward
  - total latency of those instructions is at least that of cache miss

How latency interacts with out-of-order execution?

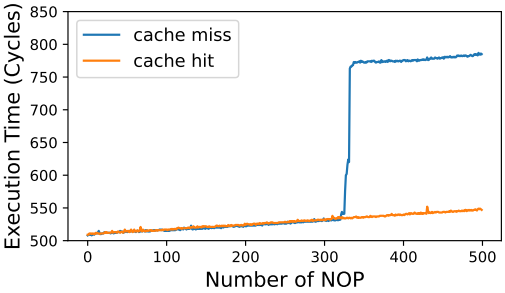
Latency-Hiding Capability

Latency-Hiding Capability

```
a = listhead->nextnode->nextnode;
NOP
...
NOP
b = *ptr
```

Latency-Hiding Capability

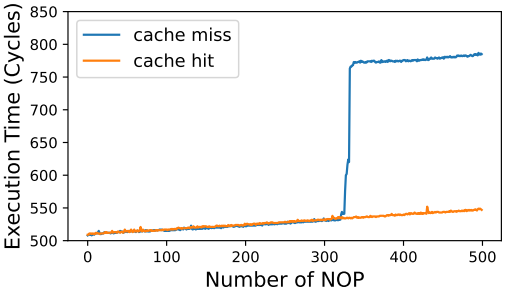
```
a = listhead->nextnode->nextnode;
NOP
...
NOP
b = *ptr
```



Latency-Hiding Capability

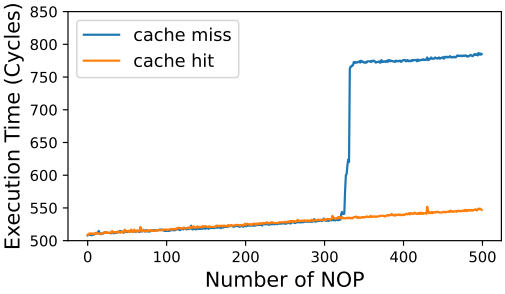
```
a = listhead->nextnode->nextnode;
NOP
...
NOP
b = *ptr
```

```
a = listhead->nextnode->nextnode;
mov r11, [a]
cmp r11, rdx
...
cmp r11, rdx
b = *ptr
```

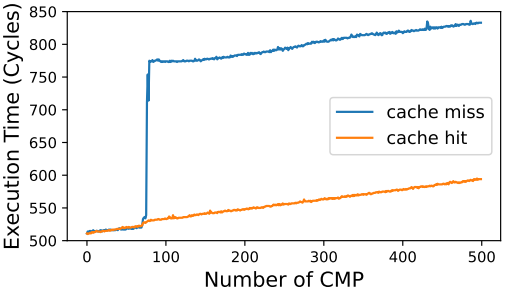


Latency-Hiding Capability

```
a = listhead->nextnode->nextnode;
NOP
...
NOP
b = *ptr
```




```
a = listhead->nextnode->nextnode;
mov r11, [a]
cmp r11, rdx
...
cmp r11, rdx
b = *ptr
```



Our Findings

Our Findings

- Out-of-order execution capability is limited
  - depend on available resources

Our Findings

- Out-of-order execution capability is limited
  - depend on available resources
- Cache miss is **not** always slower than cache hit, if:
  - there are enough instructions to bring forward
  - total latency of those instructions is at least that of cache miss

Our Findings

- Out-of-order execution capability is limited
  - depend on available resources
- Cache miss is **not** always slower than cache hit, if:
  - there are enough instructions to bring forward
  - total latency of those instructions is at least that of cache miss
- Latency of cache miss is **not** constant
  - depend on execution-time overlap

Our Findings

- Out-of-order execution capability is limited
  - depend on available resources
- Cache miss is **not** always slower than cache hit, if:
  - there are enough instructions to bring forward
  - total latency of those instructions is at least that of cache miss
- Latency of cache miss is **not** constant
  - depend on execution-time overlap
- Control resources can manipulate cache-miss latency hiding
  - allow us to identify when cache miss occurs
  - Evict+Spec+Time: know *when* beyond whether access occurs

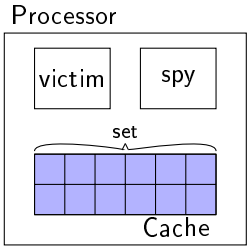
Evict+Spec+Time

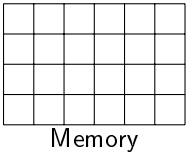
Evict+Spec+Time

- **Evict**: access a block of memory mapped to target cache sets
  $\rightarrow$ remove victim's data from cache

Evict+Spec+Time

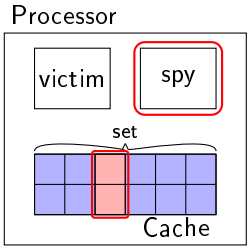
- **Evict**: access a block of memory mapped to target cache sets
  → remove victim's data from cache

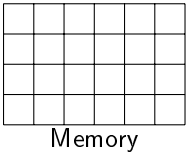




Evict+Spec+Time

- **Evict**: access a block of memory mapped to target cache sets
  → remove victim's data from cache

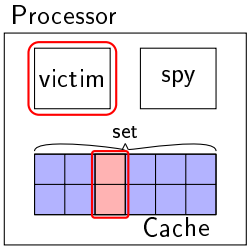


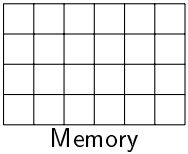


Evict+Spec+Time

- **Evict**: access a block of memory mapped to target cache sets
  → remove victim's data from cache

- **Spec+Time**: let victim execute then measure execution time
  - slow → cache miss
  - fast → no or early cache miss

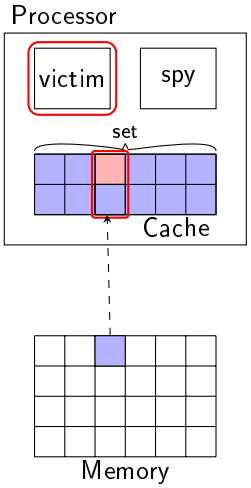




Evict+Spec+Time

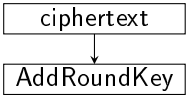
- **Evict**: access a block of memory mapped to target cache sets
  → remove victim's data from cache

- **Spec+Time**: let victim execute then measure execution time
  - slow → cache miss
  - fast → no or early cache miss



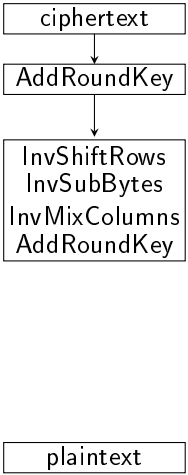
AES Decryption

AES Decryption

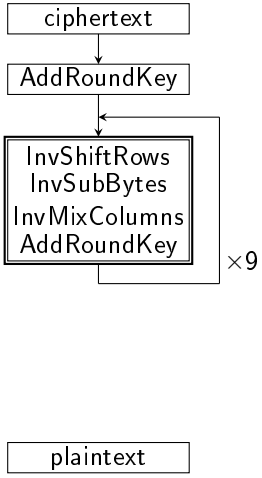


plaintext

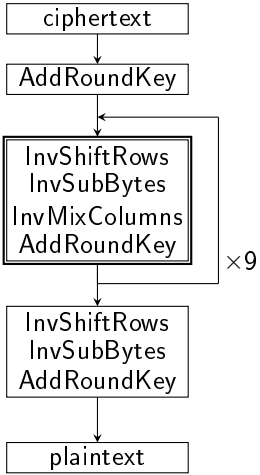
AES Decryption



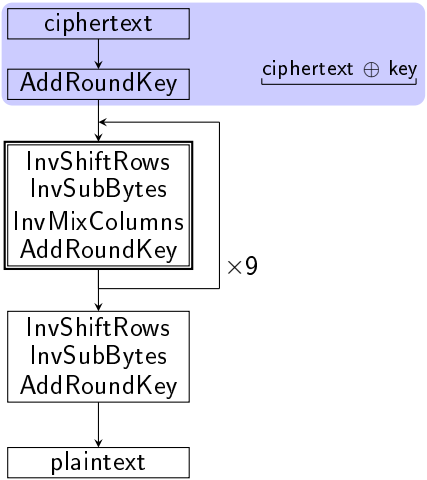
AES Decryption



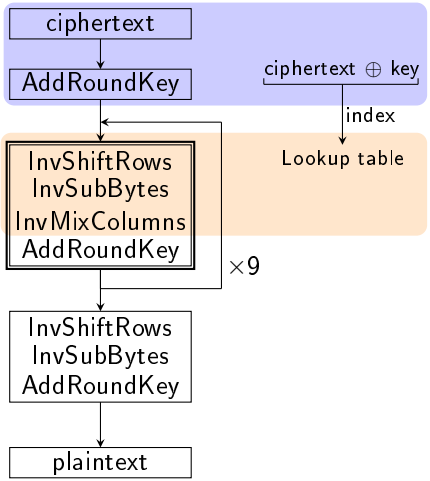
AES Decryption



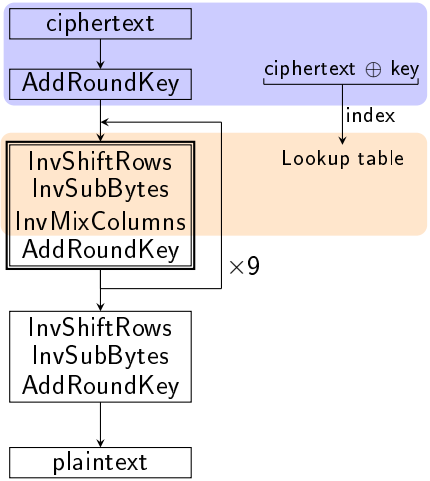
AES Decryption



AES Decryption

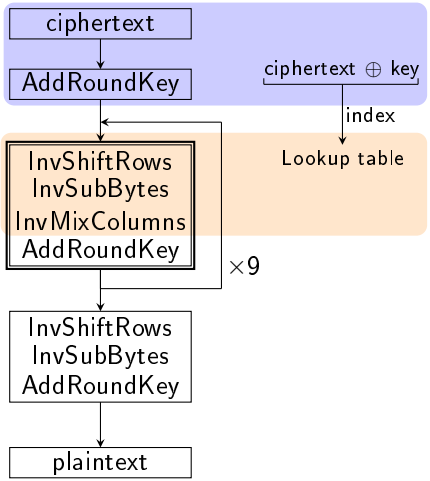


AES Decryption



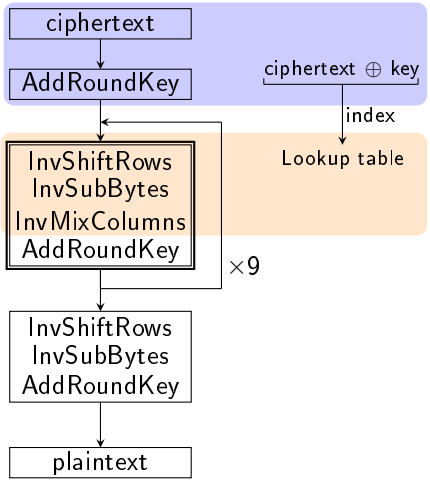
- (Assume a cache with 64 sets)

AES Decryption



- (Assume a cache with 64 sets)
- **T-table:** 256 entries of 4 bytes each
  $\frac{256 \times 4}{64} = 16$ cache lines $\Rightarrow$ 4 bits

AES Decryption



- (Assume a cache with 64 sets)
- **T-table:** 256 entries of 4 bytes each
  $\frac{256 \times 4}{64} = 16$ cache lines $\Rightarrow$ 4 bits
- **S-box:** 256 entries of 1 byte each
  $\frac{256 \times 1}{64} = 4$ cache lines $\Rightarrow$ 2 bits

Attack AES T-Table: concept

Attack AES T-Table: concept

- Target first-round
  - index $\oplus$ ciphertext $\rightarrow$ key
  - recover half key (4 out of 8 bits per byte)

Attack AES T-Table: concept

- Target first-round
  - index $\oplus$ ciphertext $\rightarrow$ key
  - recover half key (4 out of 8 bits per byte)
- Evict targeted cache line
  - manipulate execution-time overlap
  - control by exhausting, e.g., reservation station (RS)

Attack AES T-Table: concept

- Target first-round
  - index $\oplus$ ciphertext $\rightarrow$ key
  - recover half key (4 out of 8 bits per byte)
- Evict targeted cache line
  - manipulate execution-time overlap
  - control by exhausting, e.g., reservation station (RS)
- Allow only first-round instructions in, e.g., RS
  - can detect if cache miss is in first or later round
  - fast: first round or no miss / slow: later round

Attack AES T-Table: concept

- Target first-round
  - index $\oplus$ ciphertext $\rightarrow$ key
  - recover half key (4 out of 8 bits per byte)
- Evict targeted cache line
  - manipulate execution-time overlap
  - control by exhausting, e.g., reservation station (RS)
- Allow only first-round instructions in, e.g., RS
  - can detect if cache miss is in first or later round
  - fast: first round or no miss / slow: later round
- Guess and use Pearson correlation

Attack AES T-Table: comparison

Attack AES T-Table: comparison

Evict+Time [OST06] Evict+Spec+Time (ours)

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access		

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.		

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	

Attack AES T-Table: comparison

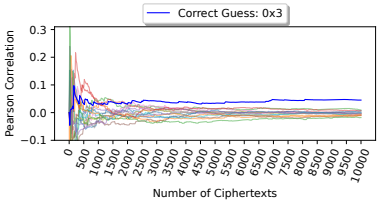
	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	$(1-(\frac{15}{16})^{36}) \times (\frac{15}{16})^{3} \approx 74\%$

Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	$(1-(\frac{15}{16})^{36}) \times (\frac{15}{16})^{3} \approx 74\%$
# samples		

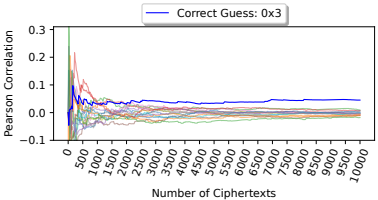
Attack AES T-Table: comparison

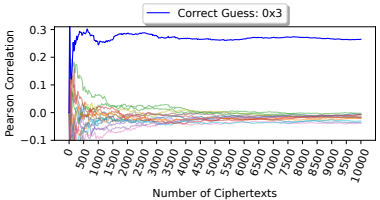
	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	$(1-(\frac{15}{16})^{36})\times(\frac{15}{16})^{3} \approx 74\%$
# samples	2500	



Attack AES T-Table: comparison

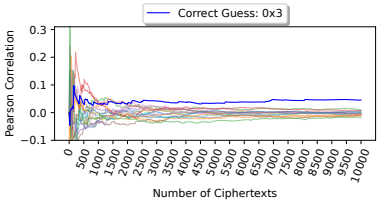
	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	$(1-(\frac{15}{16})^{36})\times(\frac{15}{16})^{3} \approx 74\%$
# samples	2500	250

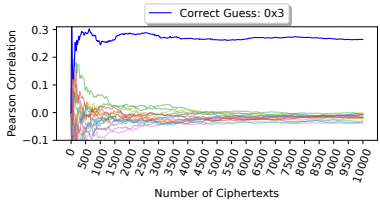




Attack AES T-Table: comparison

	Evict+Time [OST06]	Evict+Spec+Time (ours)
when access	any rounds	first round
not access prob.	$(\frac{15}{16})^{39} \approx 8\%$	$(1-(\frac{15}{16})^{36})\times(\frac{15}{16})^{3} \approx 74\%$
# samples	2500	250





$\sim$ ten time higher prob. $\rightarrow$ $\sim$ ten time fewer samples

Attack AES S-Box: outline

Attack AES S-Box: outline

- **Step 1:**
- **Step 2:**

Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
- **Step 2:**

Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:**

Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs

Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

Attack AES S-Box: outline

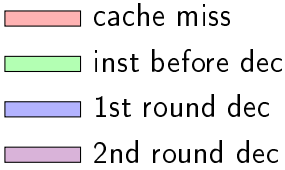
- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

one memory access	`a = *ptr1`
some instructions	`NOP` $\times$ `100`
another memory access	`a = *ptr2`
some instructions	`NOP` $\times$ `100`
AES	`AES_dec(...)`

Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

one memory access	`a = *ptr1`
some instructions	`NOP` $\times$ 100
another memory access	`a = *ptr2`
some instructions	`NOP` $\times$ 100
AES	`AES_dec(...)`

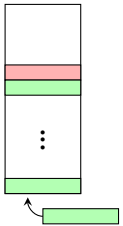


Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

cache miss
inst before dec
1st round dec
2nd round dec

one memory access	`a = *ptr1`
some instructions	`NOP` $\times$ 100
another memory access	`a = *ptr2`
some instructions	`NOP` $\times$ 100
AES	`AES_dec(...)`

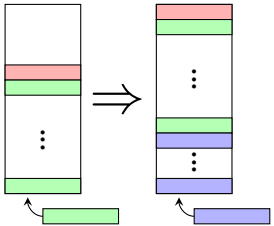


Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

cache miss
inst before dec
1st round dec
2nd round dec

one memory access	`a = *ptr1`
some instructions	`NOP` $\times$ 100
another memory access	`a = *ptr2`
some instructions	`NOP` $\times$ 100
AES	`AES_dec(...)`

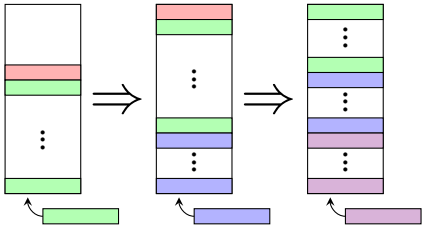


Attack AES S-Box: outline

- **Step 1:** recover 2 MSBs of each byte
  - index $\oplus$ ciphertext $\rightarrow$ key
- **Step 2:** recover remaining 6 LSBs
  - manipulate first two rounds execution

cache miss
inst before dec
1st round dec
2nd round dec

one memory access	`a = *ptr1`
some instructions	`NOP` $\times$ 100
another memory access	`a = *ptr2`
some instructions	`NOP` $\times$ 100
AES	`AES_dec(...)`



Summary

Summary

- Cache hit and miss can have the same latency

Summary

- Cache hit and miss can have the same latency
- Cache miss latency is not constant

Summary

- Cache hit and miss can have the same latency
- Cache miss latency is not constant
- Evict+Spec+Time: identify whether *and when* cache miss occurs

Summary

- Cache hit and miss can have the same latency
- Cache miss latency is not constant
- Evict+Spec+Time: identify whether *and when* cache miss occurs
- See paper for more details
  - https://eprint.iacr.org/2024/149.pdf