Malware Guard Extension: Using SGX to Conceal Cache Attacks

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard

July 6, 2017

Graz University of Technology

Outline

- We show that malicious SGX enclaves are possible

Outline

- We show that malicious SGX enclaves are possible
- We present methods to circumvent certain limitations of SGX

Outline

- We show that malicious SGX enclaves are possible
- We present methods to circumvent certain limitations of SGX
- We mount a cache attack on a RSA implementation within a different enclave

Outline

- We show that malicious SGX enclaves are possible
- We present methods to circumvent certain limitations of SGX
- We mount a cache attack on a RSA implementation within a different enclave
- We abuse SGX to prevent detection of the attack

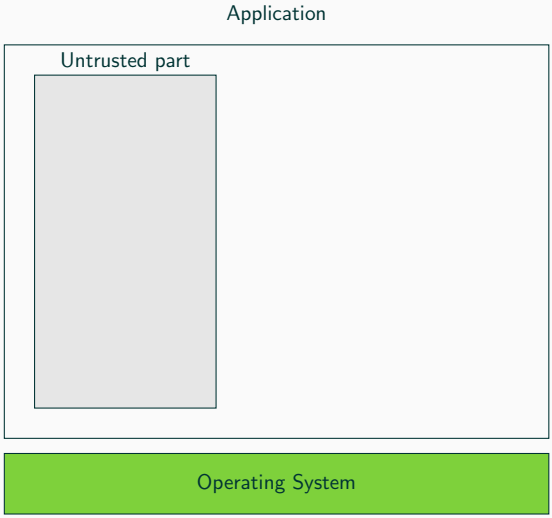
Outline

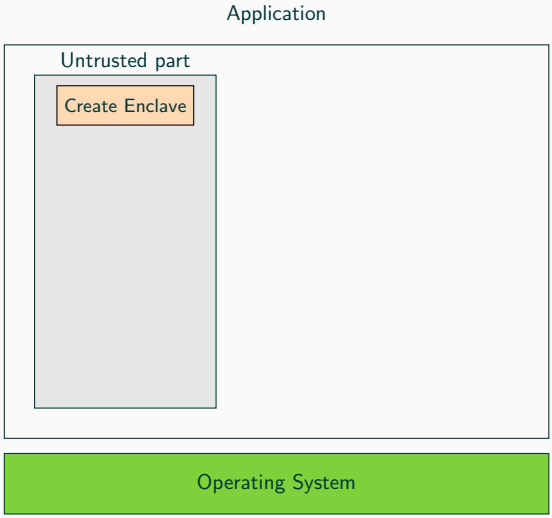
- We show that malicious SGX enclaves are possible
- We present methods to circumvent certain limitations of SGX
- We mount a cache attack on a RSA implementation within a different enclave
- We abuse SGX to prevent detection of the attack
- We show that the stealthy attack even works across Docker containers

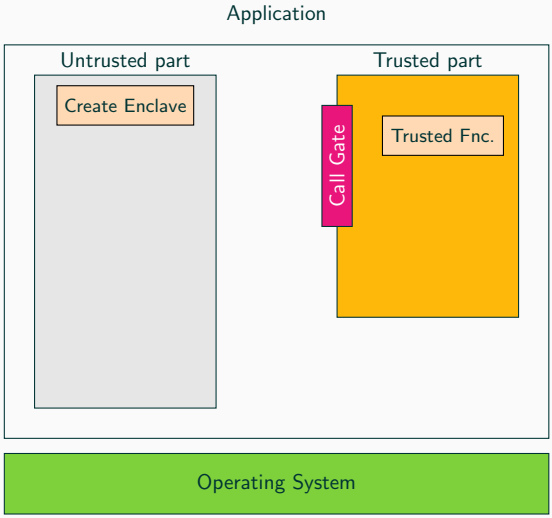
Outline

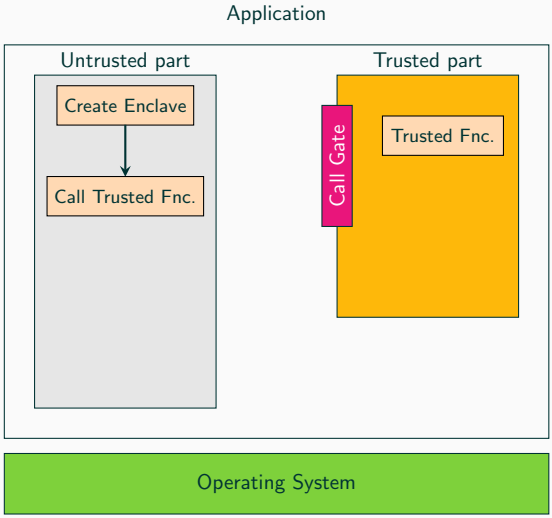
- We show that malicious SGX enclaves are possible
- We present methods to circumvent certain limitations of SGX
- We mount a cache attack on a RSA implementation within a different enclave
- We abuse SGX to prevent detection of the attack
- We show that the stealthy attack even works across Docker containers
- We discuss countermeasures to prevent such attacks

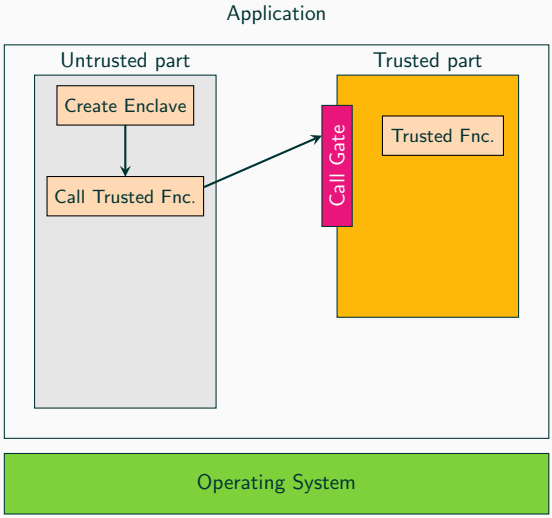
Background

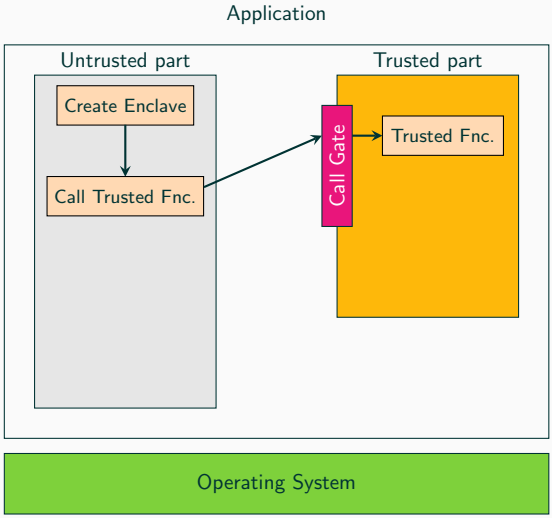

Application
Untrusted part
Operating System

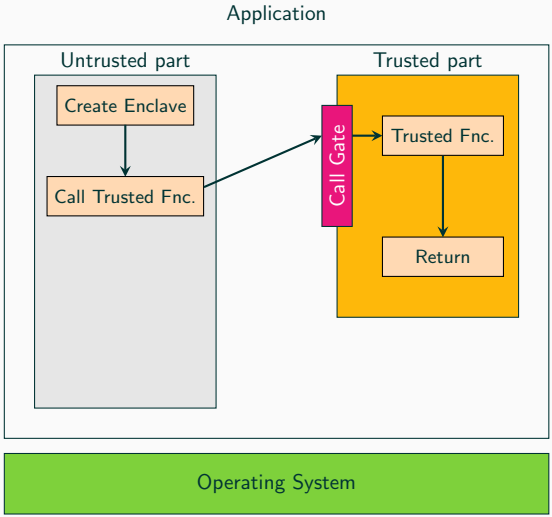

Application
Untrusted part
Create Enclave
Operating System

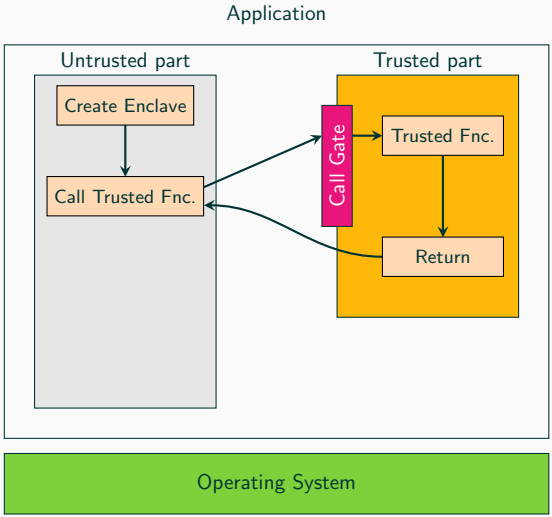

Application
Untrusted part
Create Enclave
Trusted part
Call Gate
Trusted Fnc.
Operating System

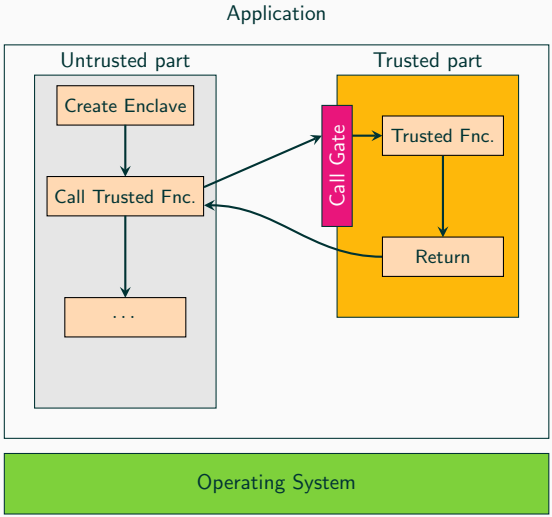

Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

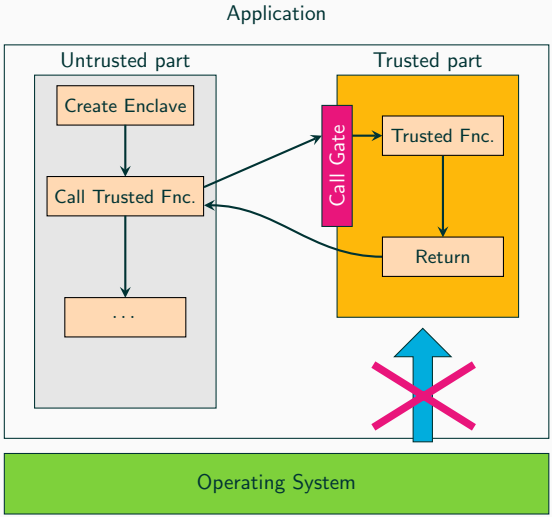

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System

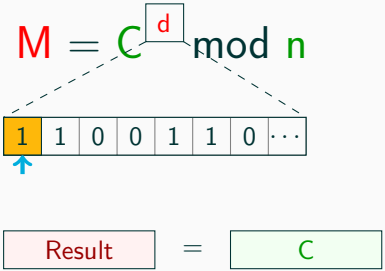

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System

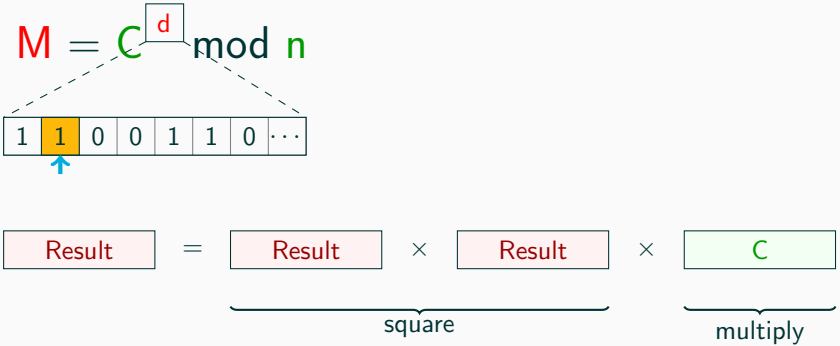

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System

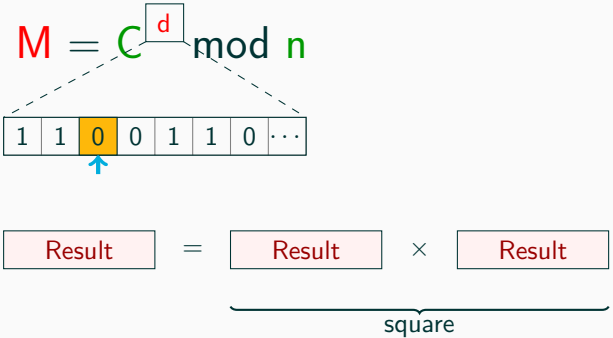

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System

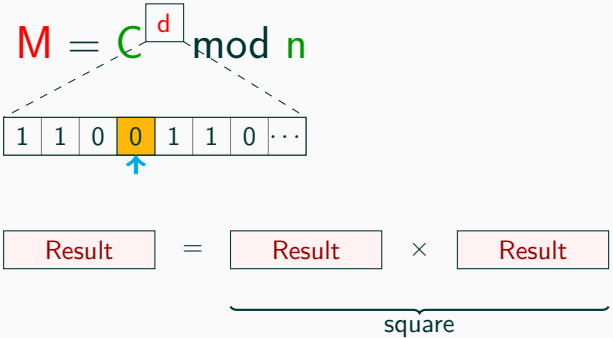

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System

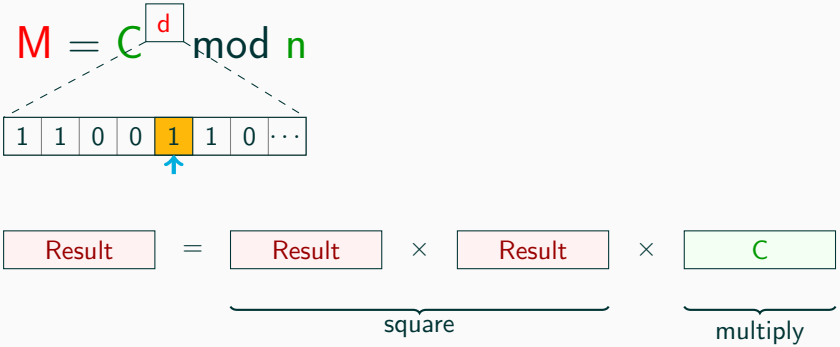
$$M = C^{d} \bmod n$$

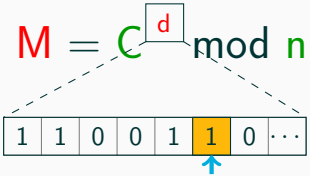

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = C

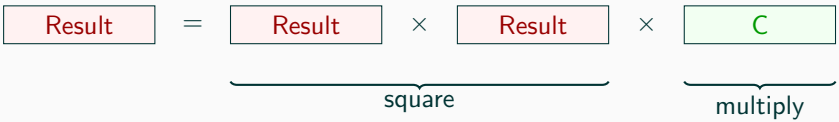

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result × C
square
multiply

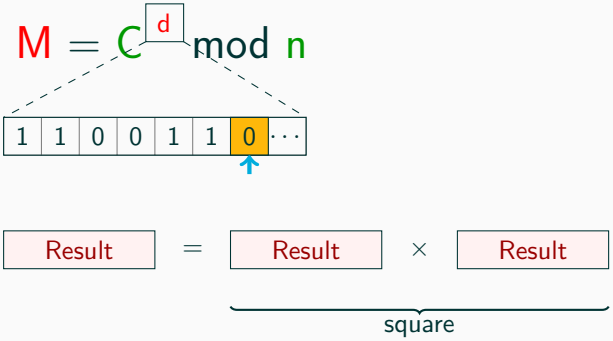

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···


Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square

Prime+Probe...

Prime+Probe...

- exploits the timing difference when accessing...

Prime+Probe...

- exploits the timing difference when accessing...
  - cached data (fast)

Prime+Probe...

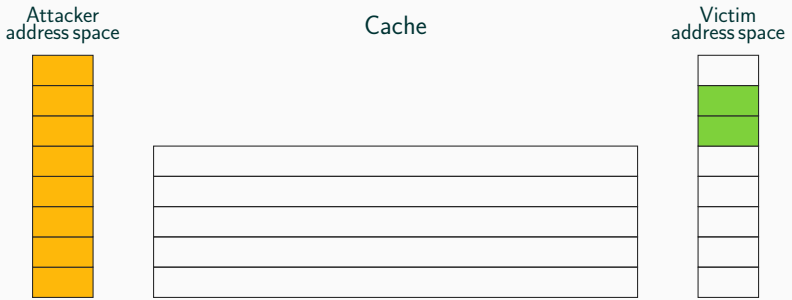
- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)

Prime+Probe...

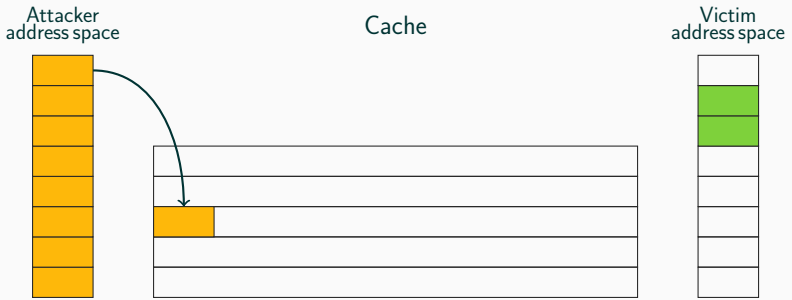
- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is applied to one cache set

Prime+Probe...

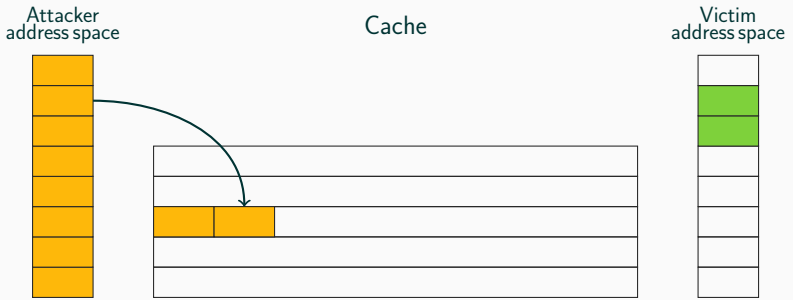
- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is applied to one cache set
- works across CPU cores as the last-level cache is shared



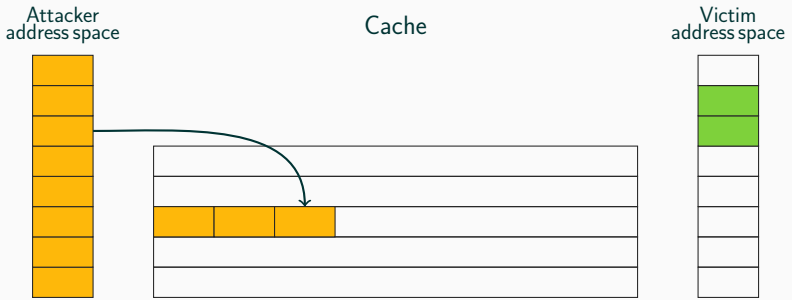
Step 0: Attacker fills the cache (prime)



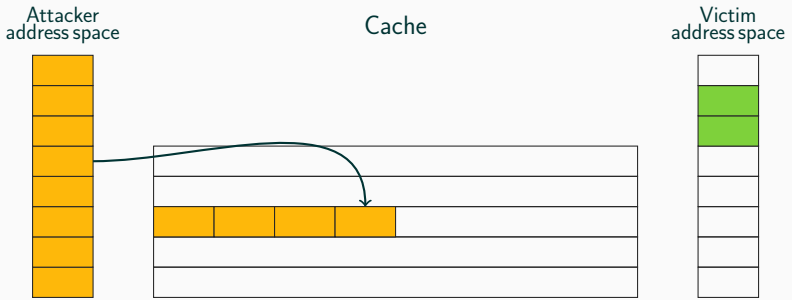
Step 0: Attacker fills the cache (prime)



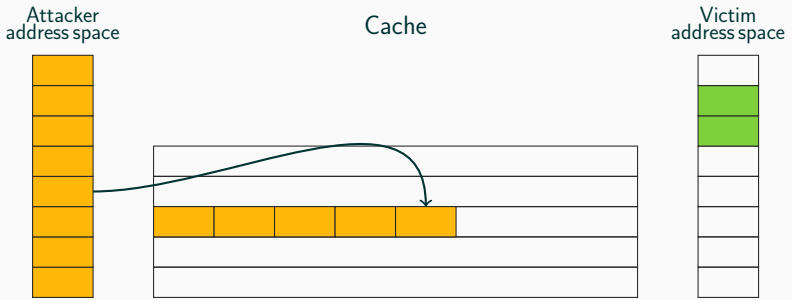
Step 0: Attacker fills the cache (prime)



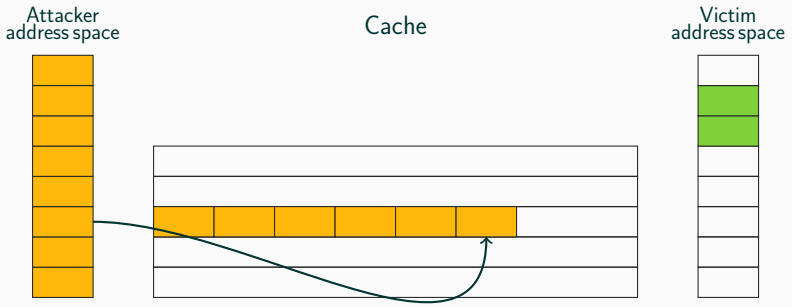
Step 0: Attacker fills the cache (prime)



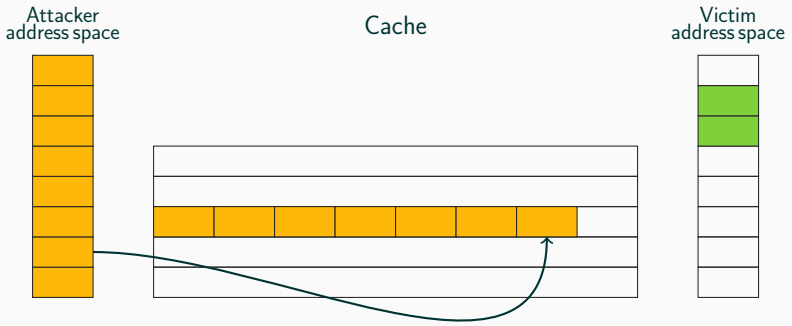
Step 0: Attacker fills the cache (prime)



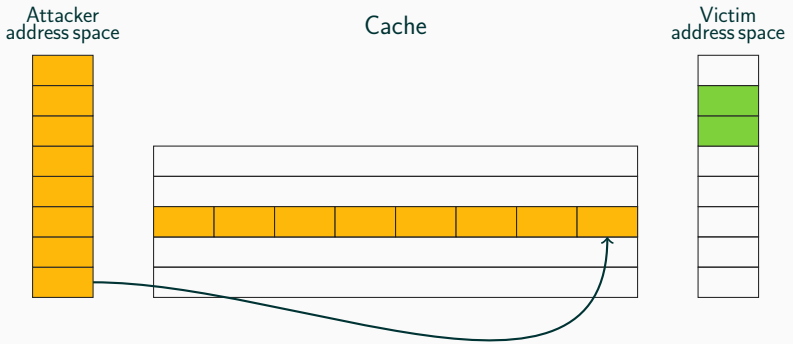
Step 0: Attacker fills the cache (prime)



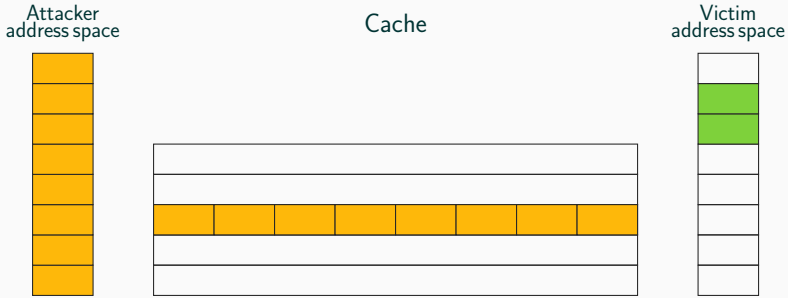
Step 0: Attacker fills the cache (prime)



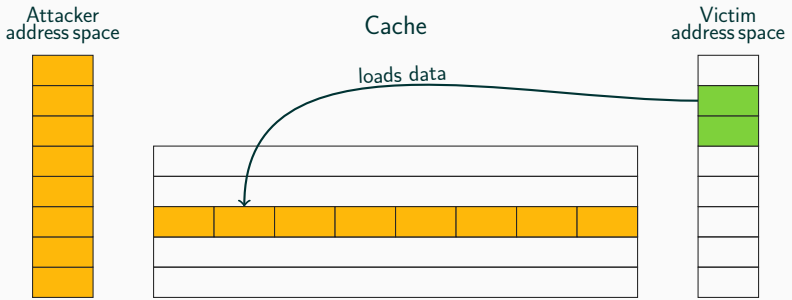
Step 0: Attacker fills the cache (prime)



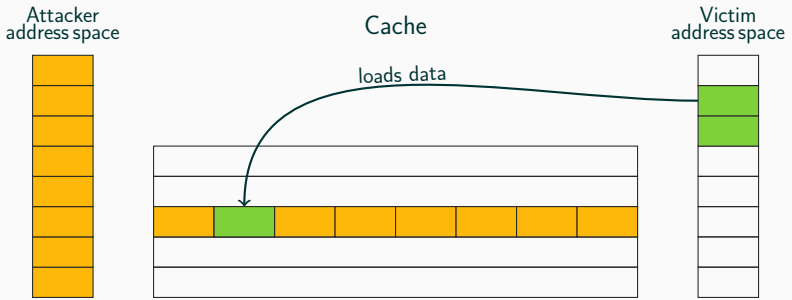
Step 0: Attacker fills the cache (prime)



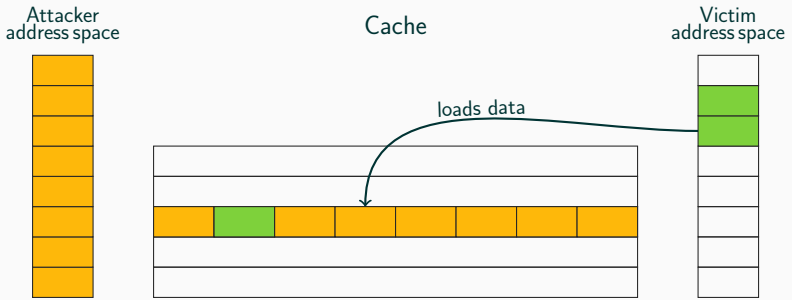
Step 0: Attacker fills the cache (prime)



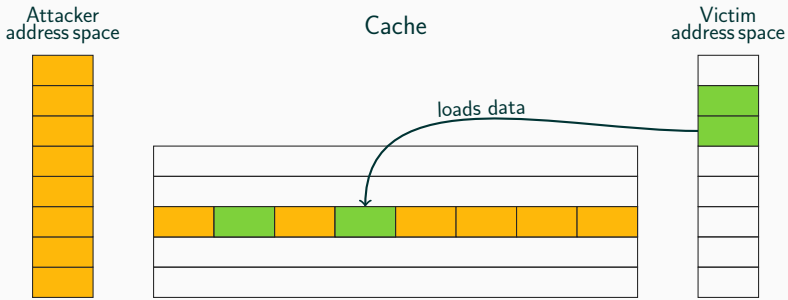
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data



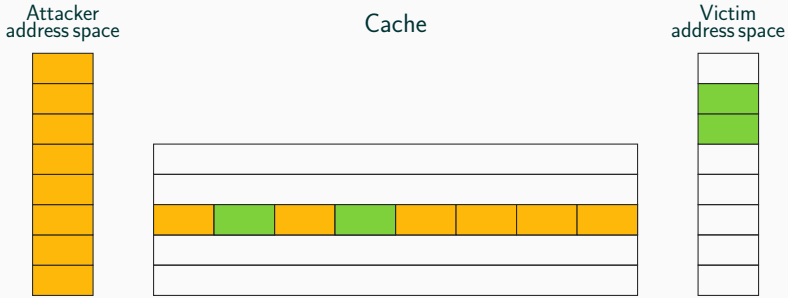
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data



Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data

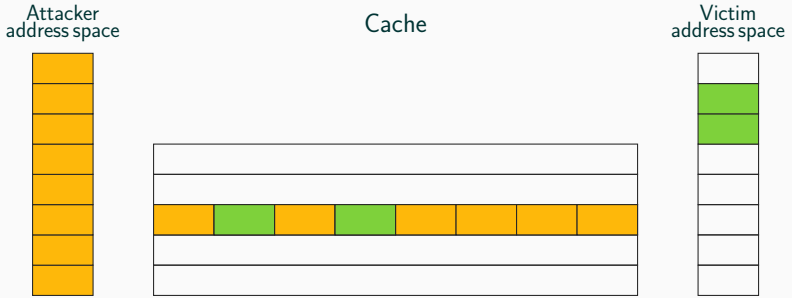


Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data

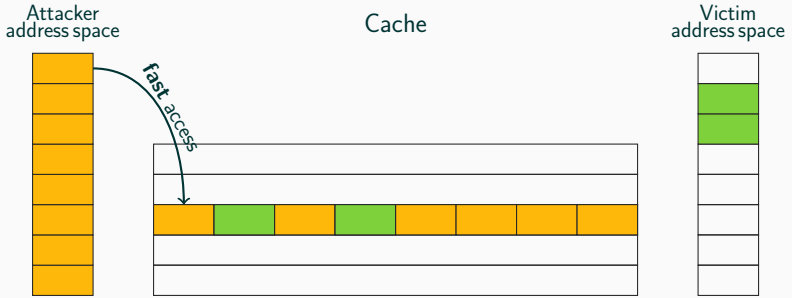


Step 0: Attacker fills the cache (prime)

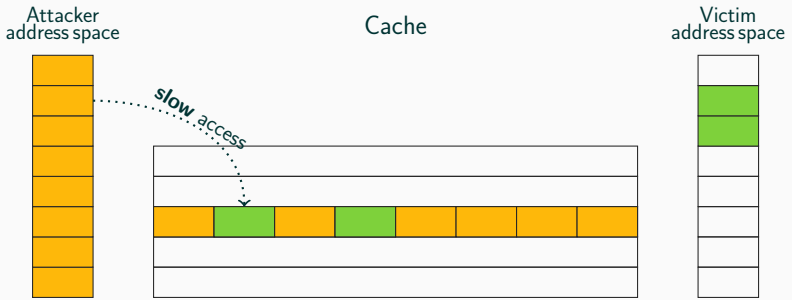
Step 1: Victim evicts cache lines by accessing own data



Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed



Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed

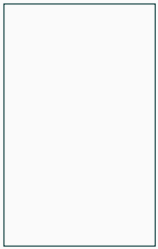


Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed

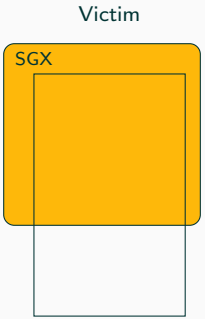
Attack

Attack Settings

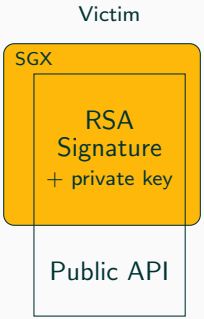
Victim



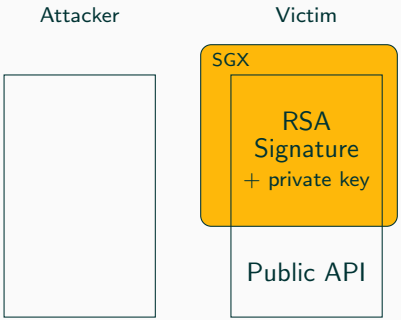
Attack Settings



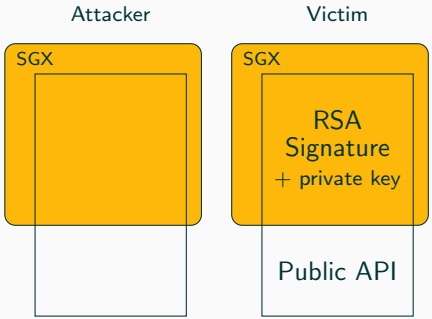
Attack Settings



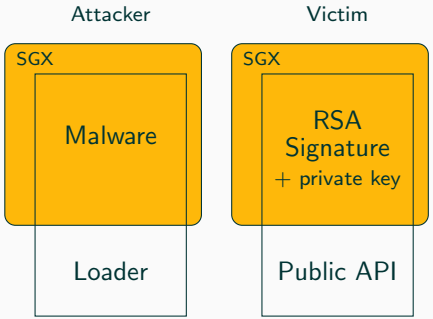
Attack Settings



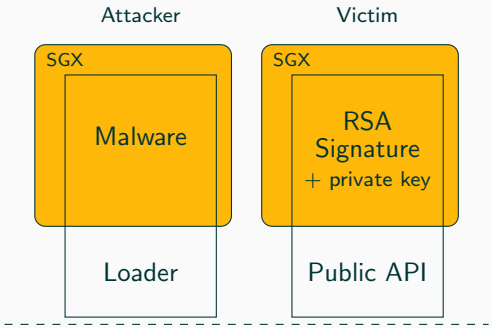
Attack Settings



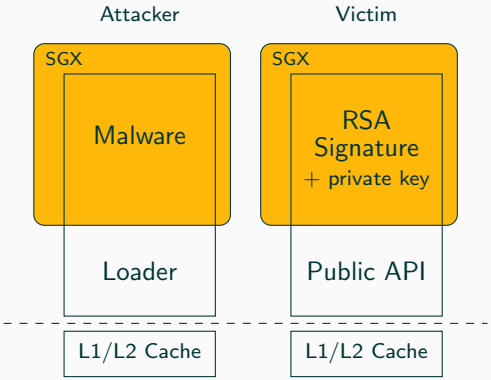
Attack Settings



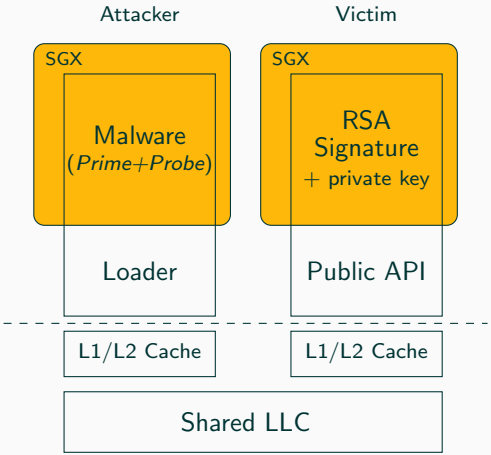
Attack Settings



Attack Settings



Attack Settings



SGX Limitations

- No access to high-precision timer (`rdtsc`)

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls

SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory

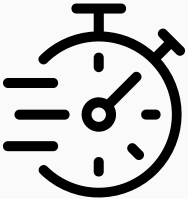
SGX Limitations

- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory
- No physical addresses

SGX Limitations

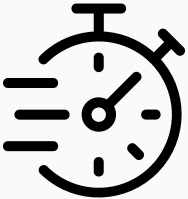
- No access to high-precision timer (`rdtsc`)
- No syscalls
- No shared memory
- No physical addresses
- No 2 MB large pages

Timer



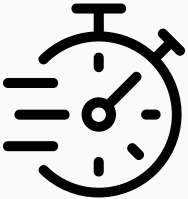
- We have to build our own timer

Timer



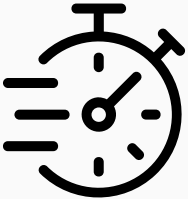
- We have to build our own timer
- Timer resolution must be in the order of cycles

Timer



- We have to build our own timer
- Timer resolution must be in the order of cycles
- Start a thread that continuously increments a global variable

Timer



- We have to build our own timer
- Timer resolution must be in the order of cycles
- Start a thread that continuously increments a global variable
- The global variable is our timestamp

Timer

CPU cycles one increment takes

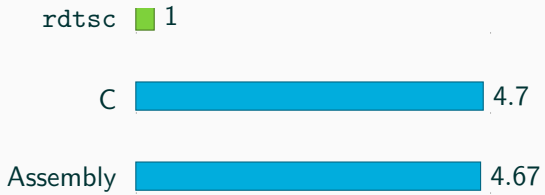
`rdtsc` 1

```
1 timestamp = rdtsc();
```

CPU cycles one increment takes

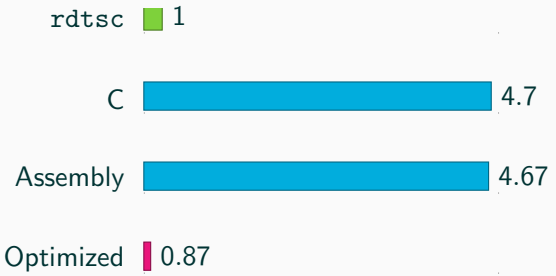

```
1 while(1) {
2   timestamp++;
3 }
```

CPU cycles one increment takes

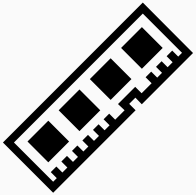



```
1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
```

CPU cycles one increment takes

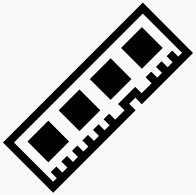



```
1 mov &timestamp, %rcx
2 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```



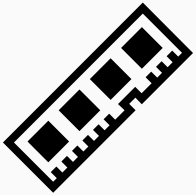
- Cache set is determined by part of physical address

Physical Addresses



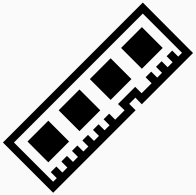
- Cache set is determined by part of physical address
- We have no knowledge of physical addresses

Physical Addresses



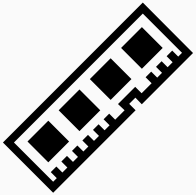
- Cache set is determined by part of physical address
- We have no knowledge of physical addresses
- Use the DRAM mapping reverse engineered by Pessl et al.

Physical Addresses



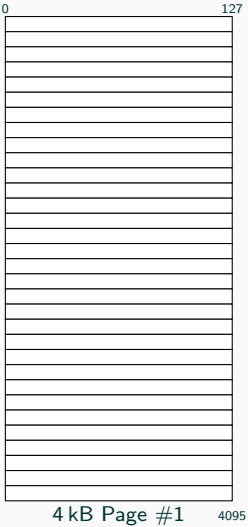
- Cache set is determined by part of physical address
- We have no knowledge of physical addresses
- Use the DRAM mapping reverse engineered by Pessl et al.
- Exploit timing differences to find DRAM row borders

Physical Addresses



- Cache set is determined by part of physical address
- We have no knowledge of physical addresses
- Use the DRAM mapping reverse engineered by Pessl et al.
- Exploit timing differences to find DRAM row borders
- The 18 LSBs are '0' at a row border

Physical Addresses



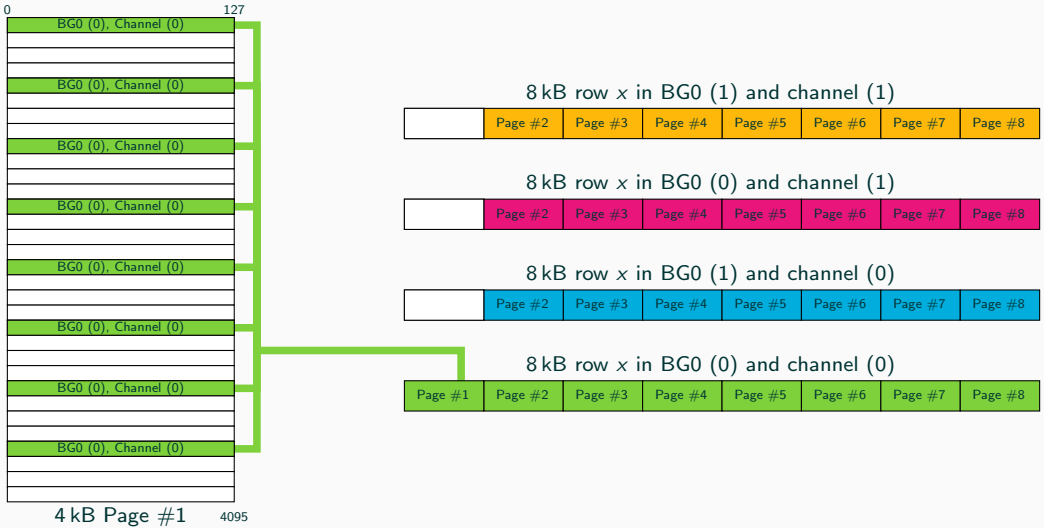
8 kB row x in BG0 (1) and channel (1)

8 kB row x in BG0 (0) and channel (1)

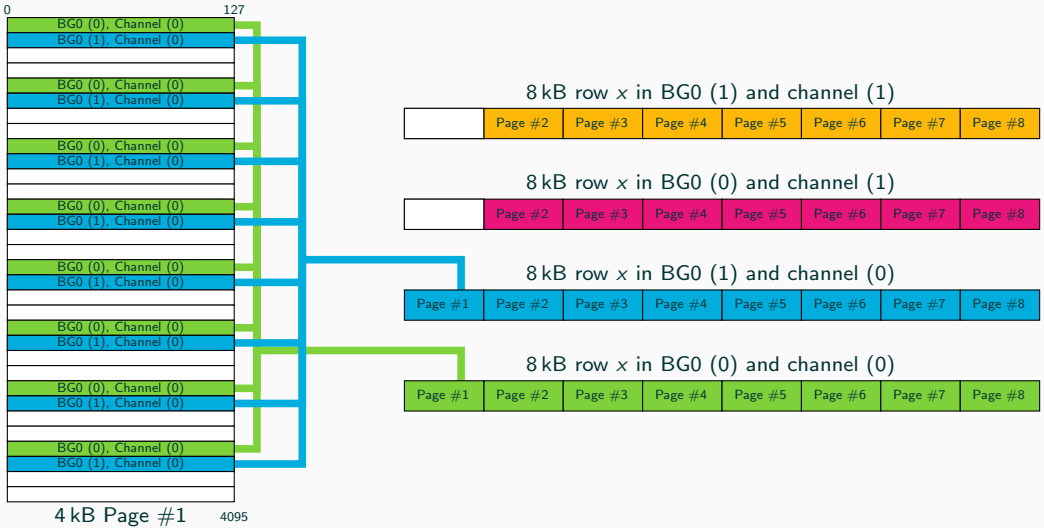
8 kB row x in BG0 (1) and channel (0)

8 kB row x in BG0 (0) and channel (0)

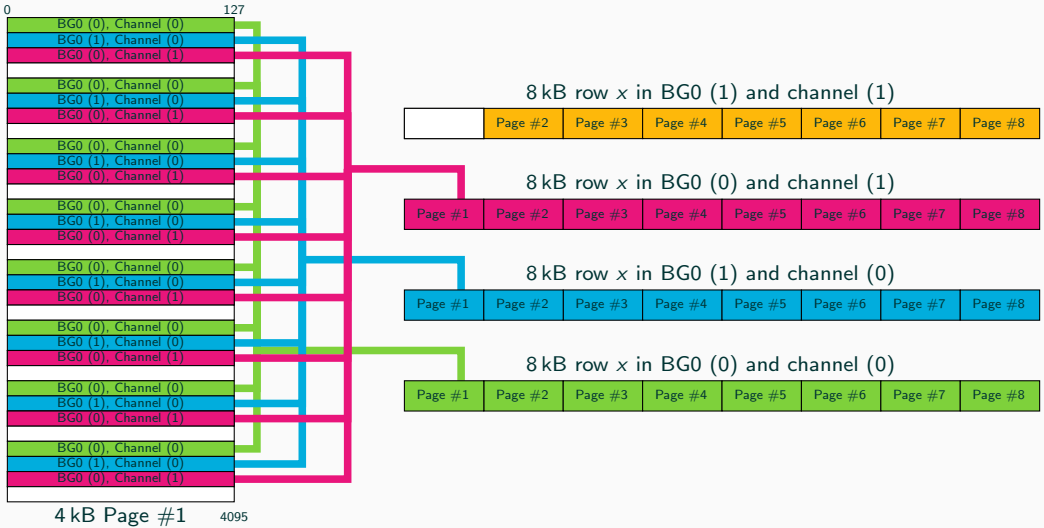
Physical Addresses

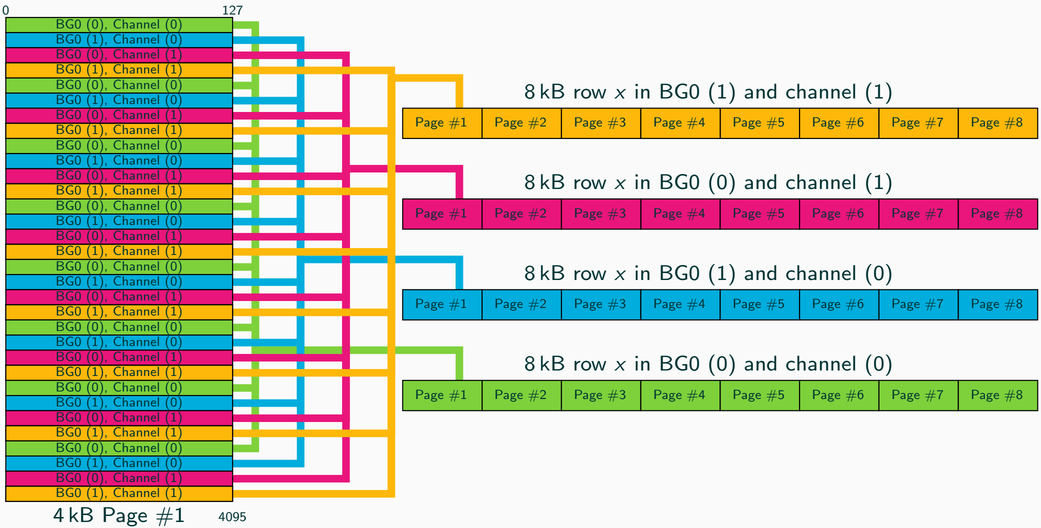


Physical Addresses

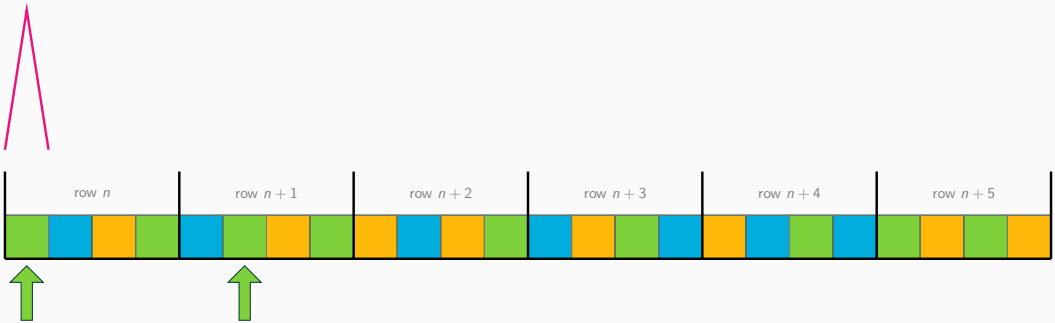


Physical Addresses

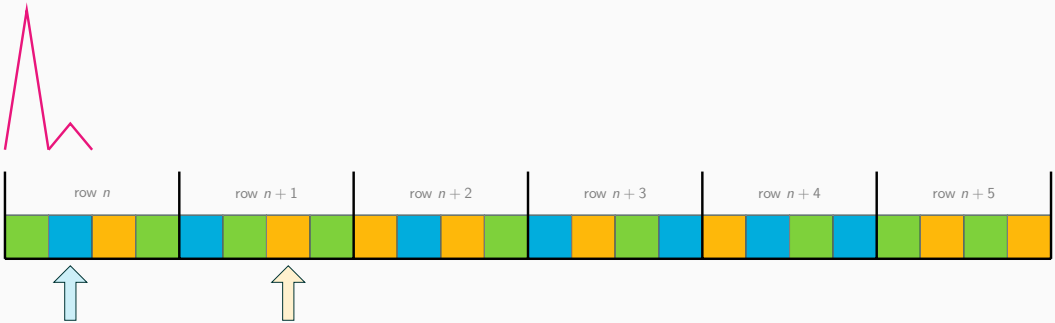


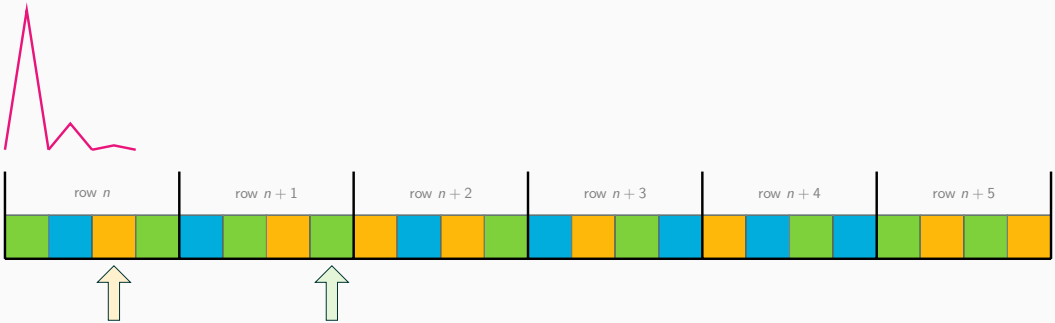

0
127
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (1)
4 kB Page #1
4095
8 kB row x in BG0 (1) and channel (1)
8 kB row x in BG0 (0) and channel (1)
8 kB row x in BG0 (1) and channel (0)
8 kB row x in BG0 (0) and channel (0)
Page #1
Page #2
Page #3
Page #4
Page #5
Page #6
Page #7
Page #8

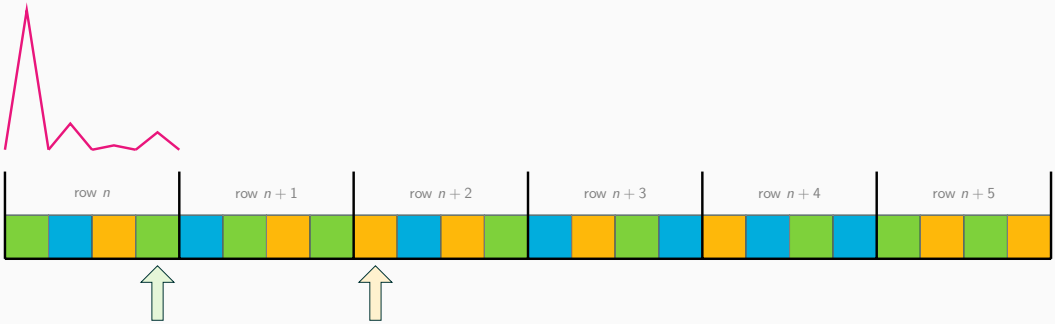
Physical Addresses

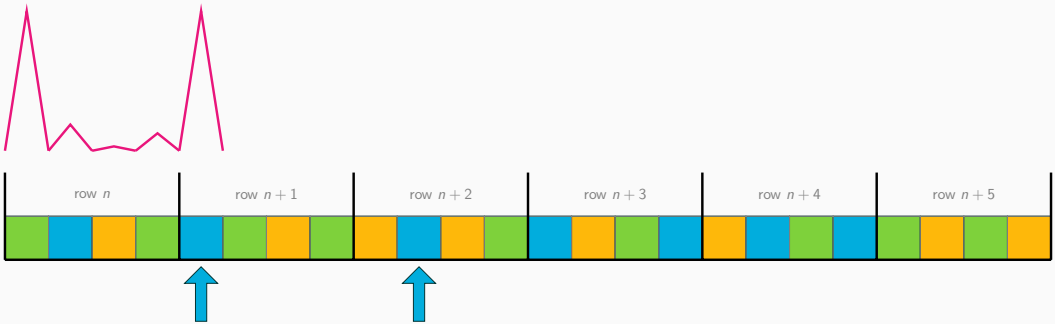


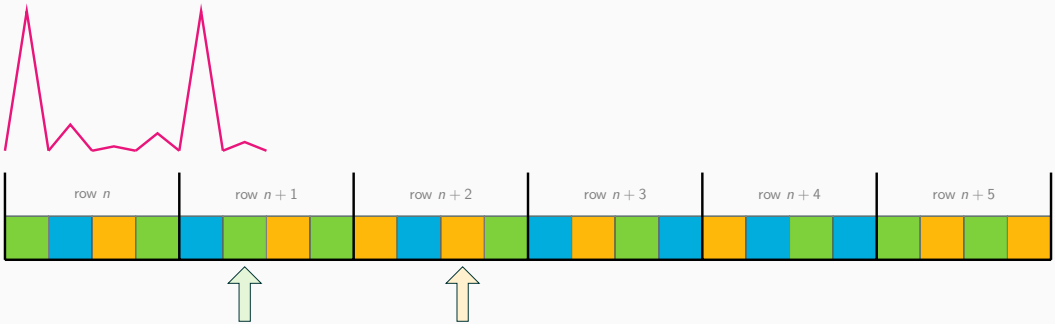
Physical Addresses



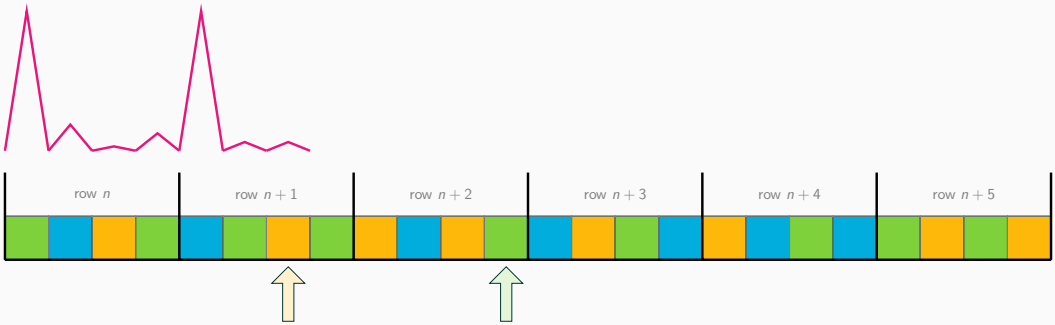

row n
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$

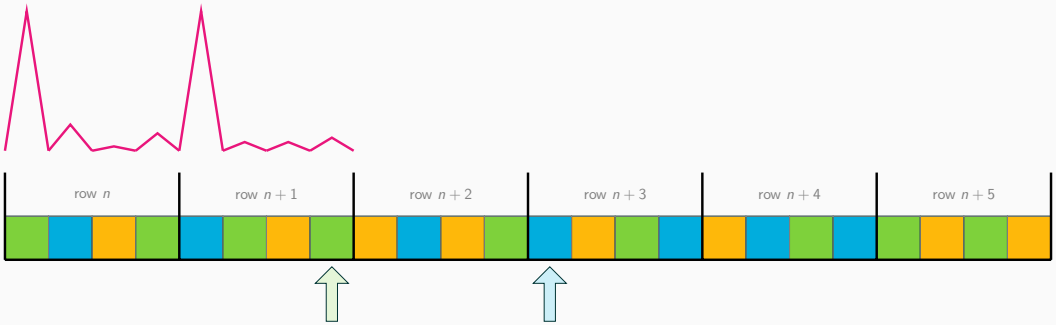

row n
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$


row n
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$

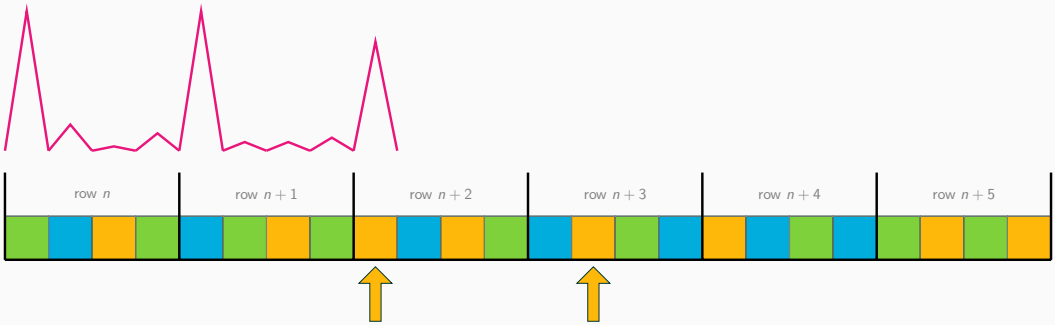

row n
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$

Physical Addresses

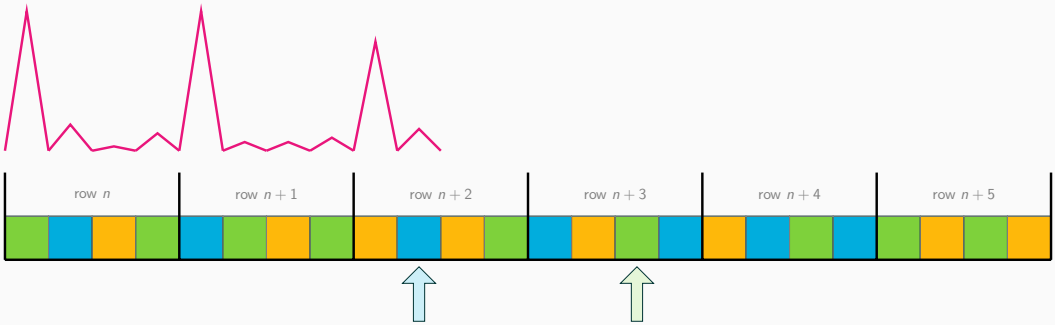



row n
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$

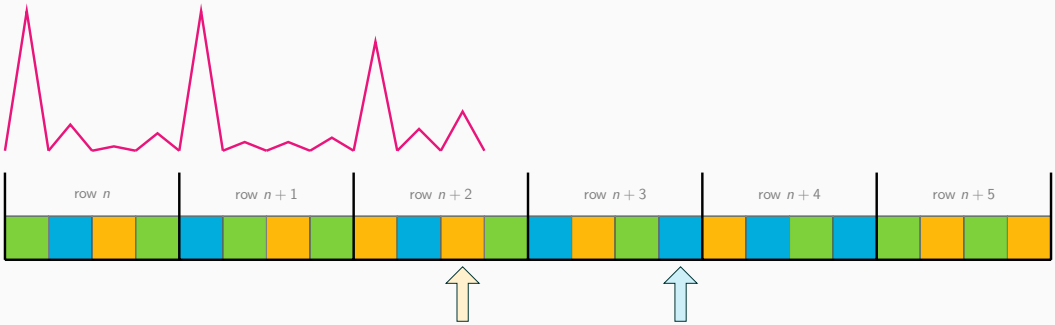
Physical Addresses



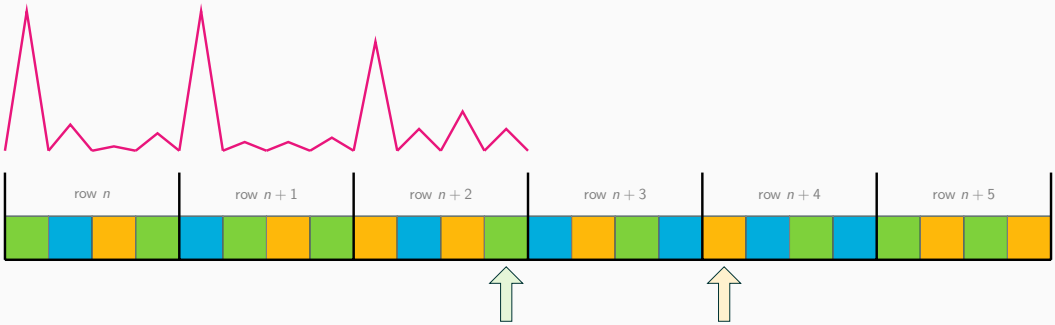
Physical Addresses



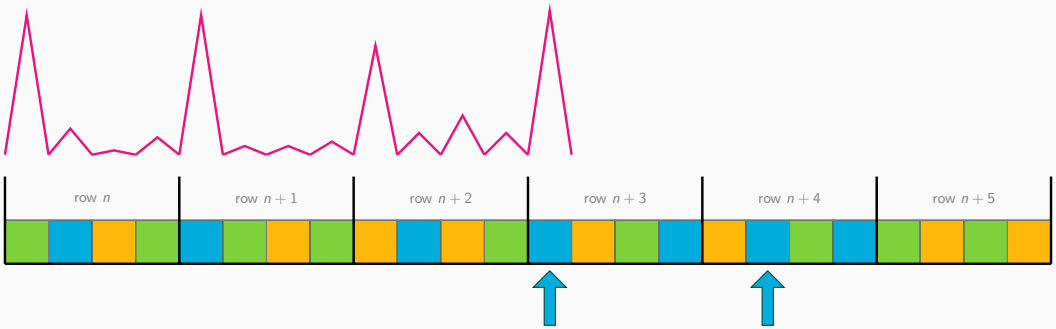
Physical Addresses



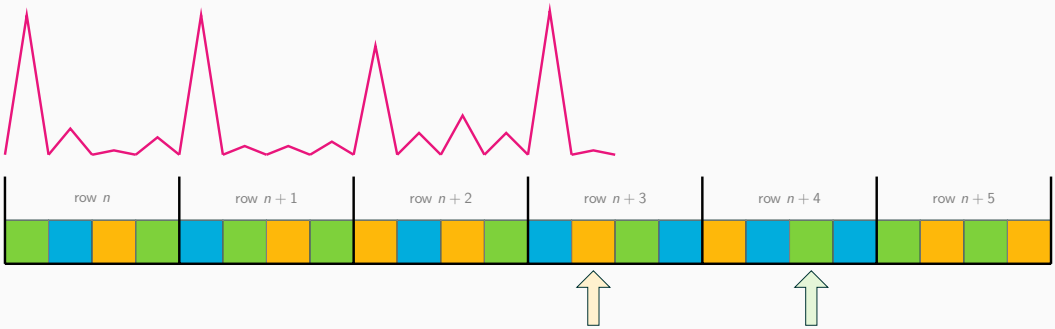
Physical Addresses



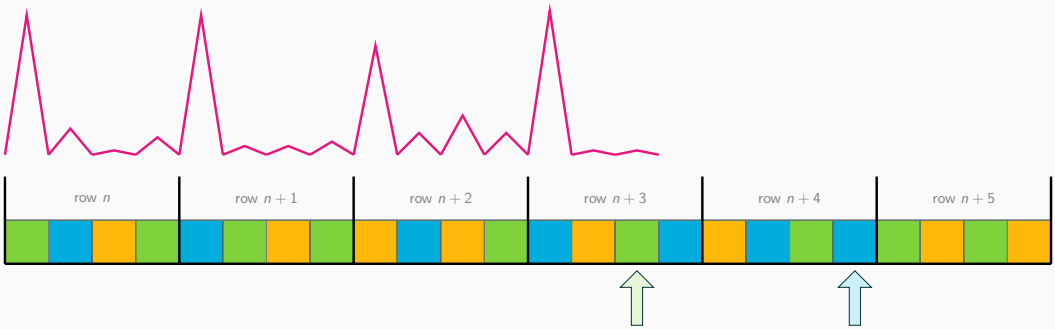
Physical Addresses



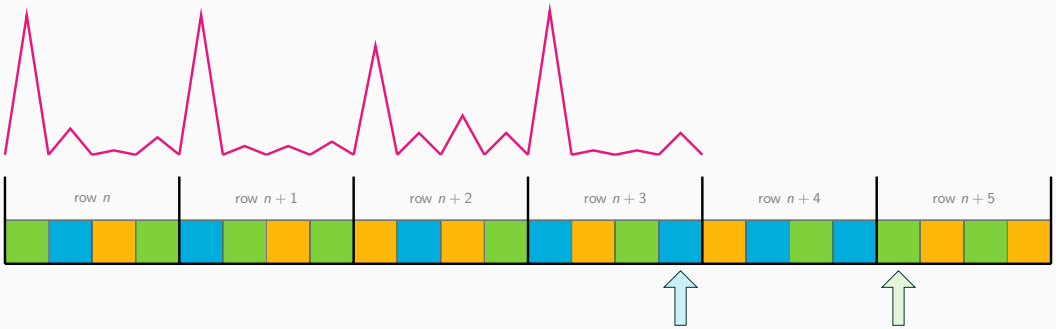
Physical Addresses



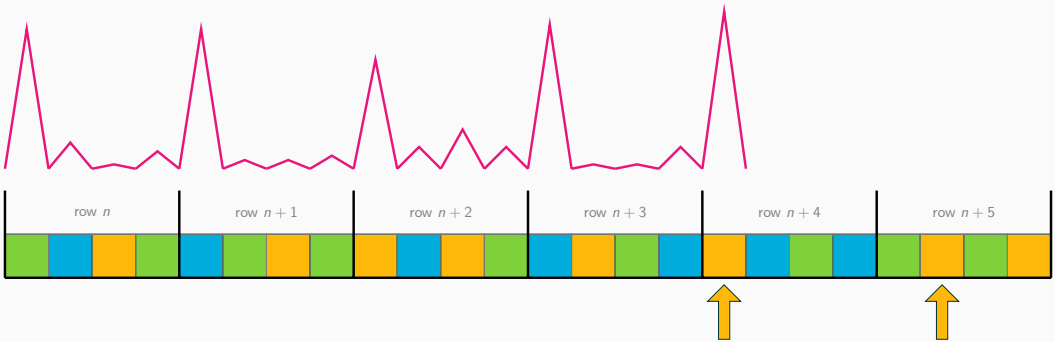
Physical Addresses



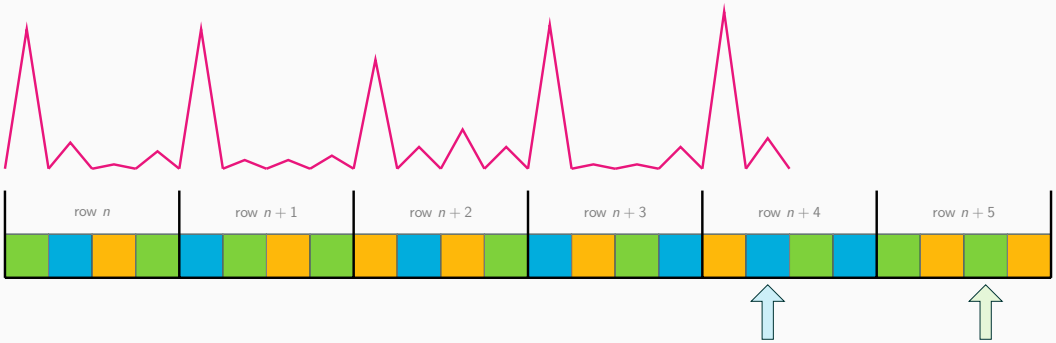
Physical Addresses



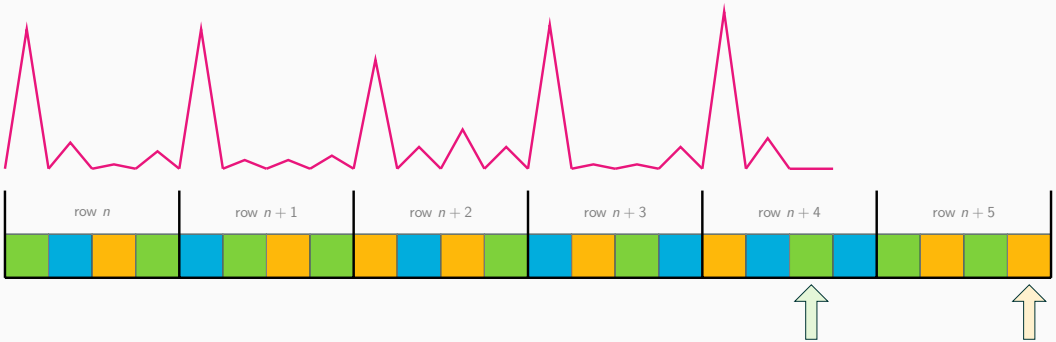
Physical Addresses



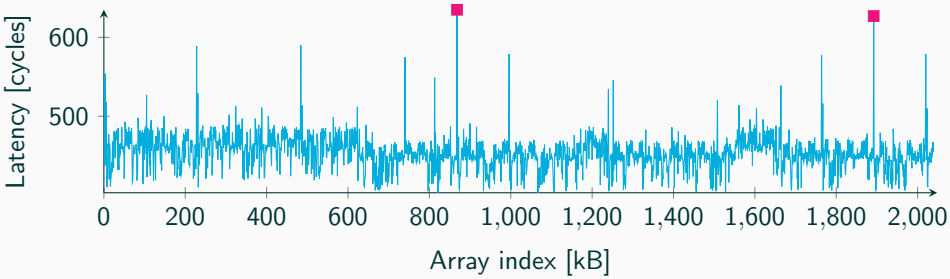
Physical Addresses

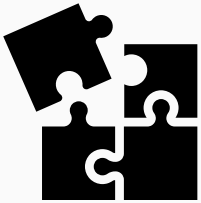


Physical Addresses



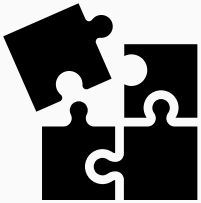
Result on an Intel i5-6200U





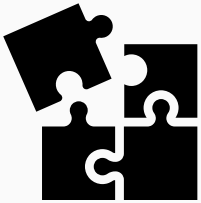
1. Use the counting primitive to measure DRAM accesses

Combining Everything



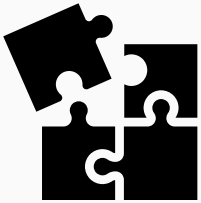
1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders

Combining Everything



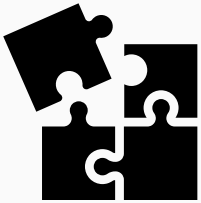
1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
3. Row borders have the 18 LSBs set to '0' $\rightarrow$ maps to cache set '0'

Combining Everything



1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
3. Row borders have the 18 LSBs set to '0' $\rightarrow$ maps to cache set '0'
4. Build the eviction set for the Prime+Probe attack

Combining Everything

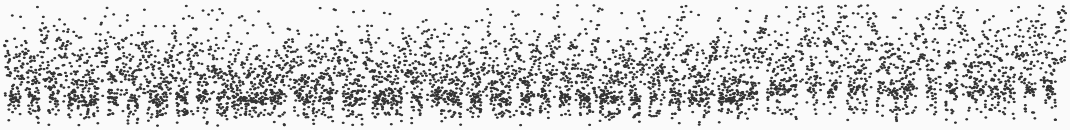


1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
3. Row borders have the 18 LSBs set to '0' $\rightarrow$ maps to cache set '0'
4. Build the eviction set for the Prime+Probe attack
5. Mount Prime+Probe on the buffer containing the multiplier

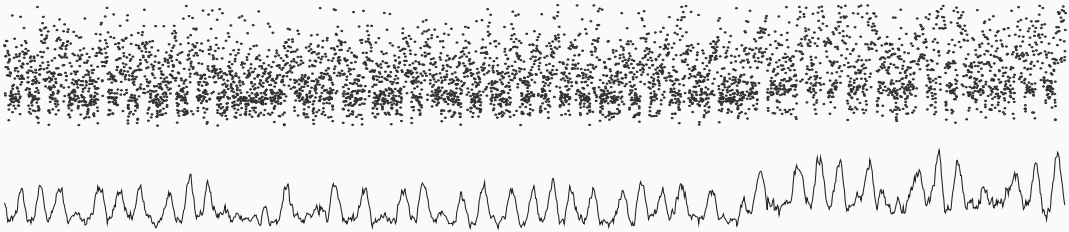
Results

Measured Trace

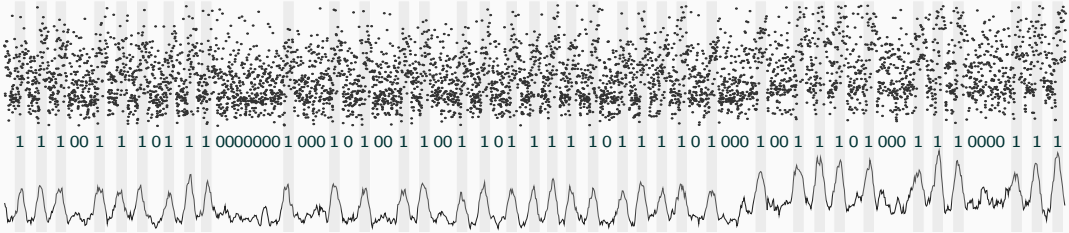
Raw Prime+Probe trace...



...processed with a simple moving average...

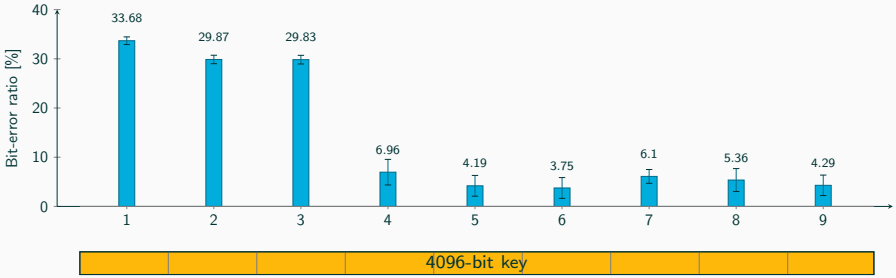


...allows to clearly see the bits of the exponent

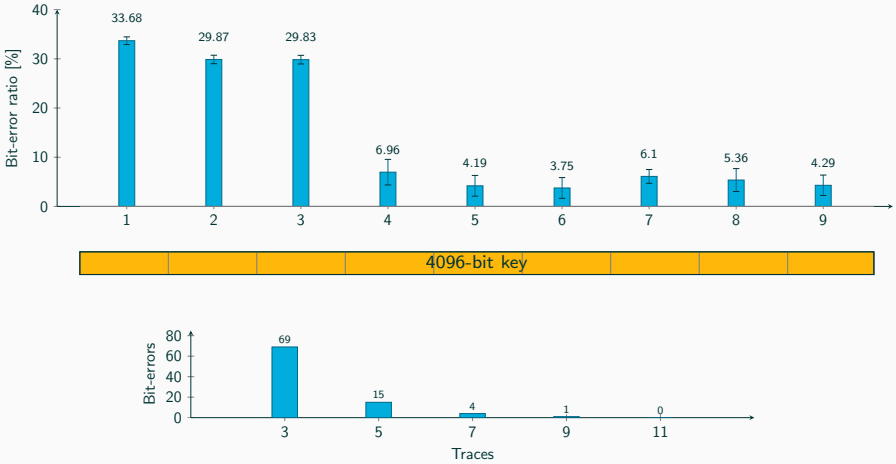


Error Probability

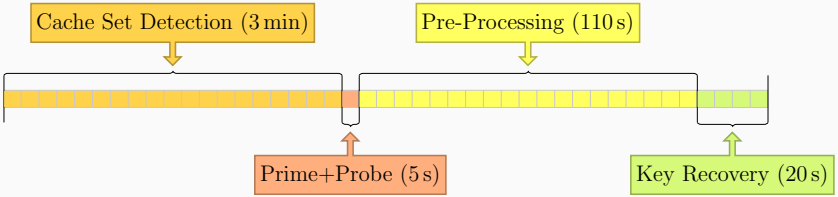
Error probability depends on which cache set of the key we attack



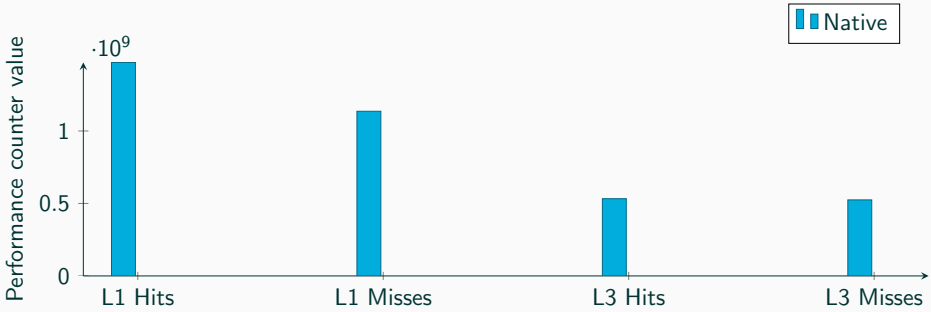
Error probability depends on which cache set of the key we attack



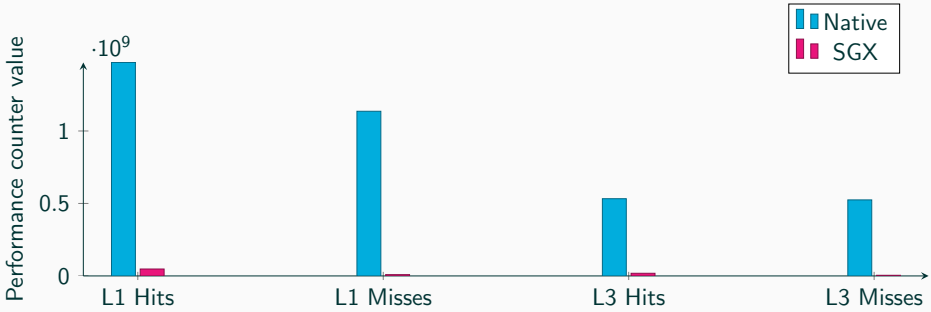
Full recovery of a 4096-bit RSA key in approximately 5 minutes



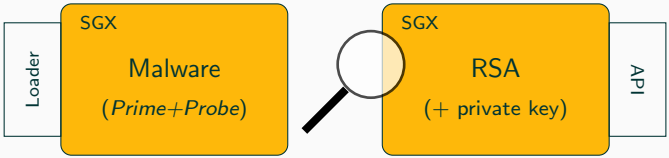
Performance Counters



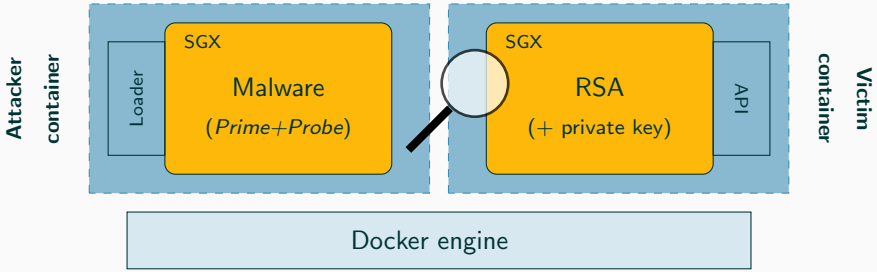
Performance Counters

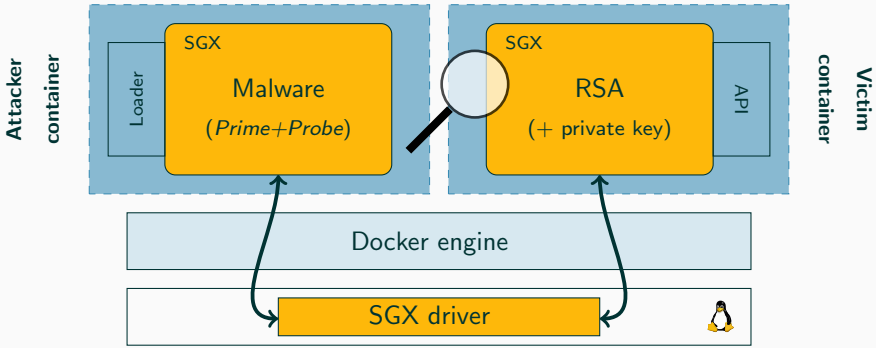


Bonus: Docker

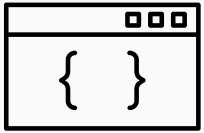


Bonus: Docker

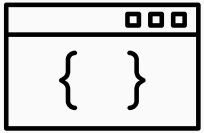



Attacker container
Loader
SGX
Malware
(Prime+Probe)
SGX
RSA
(+ private key)
API
Victim container
Docker engine
SGX driver

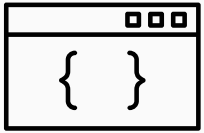
Countermeasures



- Cache attacks can be prevented on source level

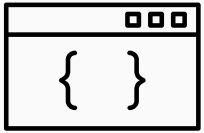


- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations



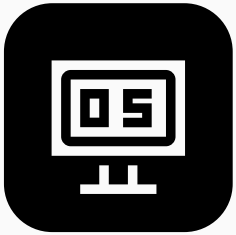
- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
- Exponent blinding for RSA prevents multi-trace attacks

Source Level



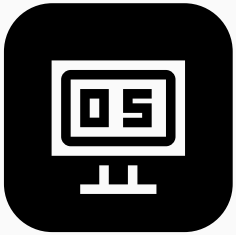
- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
- Exponent blinding for RSA prevents multi-trace attacks
- Bit-sliced implementations are not vulnerable to cache attacks

Operating System Level



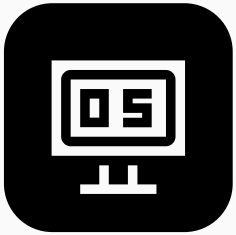
- Trusting the operating system weakens SGX threat model

Operating System Level



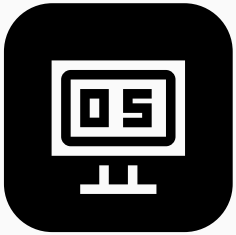
- Trusting the operating system weakens SGX threat model
- Method for the operating system to inspect enclave code

Operating System Level



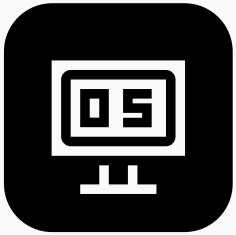
- Trusting the operating system weakens SGX threat model
- Method for the operating system to inspect enclave code
- Re-enable certain performance counters, such as L3 hits/misses

Operating System Level



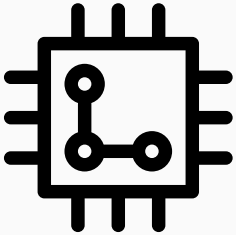
- Trusting the operating system weakens SGX threat model
- Method for the operating system to inspect enclave code
- Re-enable certain performance counters, such as L3 hits/misses
- Enclave coloring to prevent cross-enclave attacks

Operating System Level

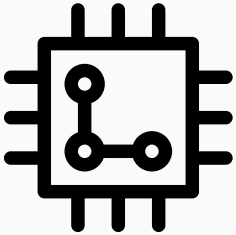


- Trusting the operating system weakens SGX threat model
- Method for the operating system to inspect enclave code
- Re-enable certain performance counters, such as L3 hits/misses
- Enclave coloring to prevent cross-enclave attacks
- Heap randomization to randomize cache sets

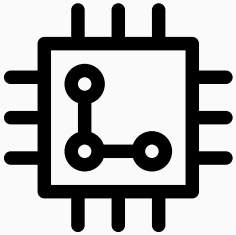
Hardware Level



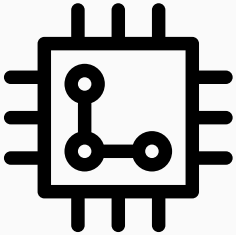
- Intel could prevent attacks by changing the hardware



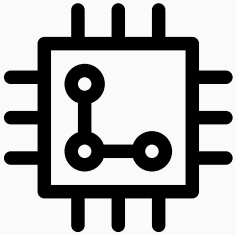
- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX



- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with `eenter`

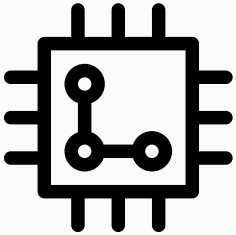


- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with `eenter`
  - Entering an enclave would automatically activate CAT for this core



- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with `eenter`
  - Entering an enclave would automatically activate CAT for this core
  - L3 is then isolated from all other enclaves and applications

Hardware Level



- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with `eenter`
  - Entering an enclave would automatically activate CAT for this core
  - L3 is then isolated from all other enclaves and applications
- Provide a non-shared secure memory element which is not cached

Conclusion

Conclusion

- We showed that attacks can be mounted from within SGX enclaves

Conclusion

- We showed that attacks can be mounted from within SGX enclaves
- We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution

Conclusion

- We showed that attacks can be mounted from within SGX enclaves
- We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
- Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces

Conclusion

- We showed that attacks can be mounted from within SGX enclaves
- We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
- Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces
- SGX allows to completely hide an attack from state-of-the-art detection techniques

Conclusion

- We showed that attacks can be mounted from within SGX enclaves
- We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
- Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces
- SGX allows to completely hide an attack from state-of-the-art detection techniques
- The attack showed that SGX is not a magic solution to make software safe

Thank you!

Malware Guard Extension: Using SGX to Conceal Cache Attacks

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard

July 6, 2017

Graz University of Technology