Cache-Timing Attack Against HQC

Chitchanok Chuengsatiansup

The University of Melbourne, Australia

About Me

About Me

2023 **Senior Lecturer:** The University of Melbourne, Australia

2019 **Lecturer:** The University of Adelaide, Australia

About Me

2023 **Senior Lecturer:** The University of Melbourne, Australia

2019 **Lecturer:** The University of Adelaide, Australia

2017 **Post-doc:** Ecole Normale Supérieure de Lyon, France

2013 **PhD Student:** Technische Universiteit Eindhoven, The Netherlands

About Me

2023 **Senior Lecturer:** The University of Melbourne, Australia

2019 **Lecturer:** The University of Adelaide, Australia

2017 **Post-doc:** Ecole Normale Supérieure de Lyon, France

2013 **PhD Student:** Technische Universiteit Eindhoven, The Netherlands

2011 **Master's Student:** The University of Tokyo, Japan

2010 **Research Student:** The University of Tokyo, Japan

2006 **Bachelor's Student:** Chulalongkorn University, Thailand

Efficient and Secure Cryptosystems

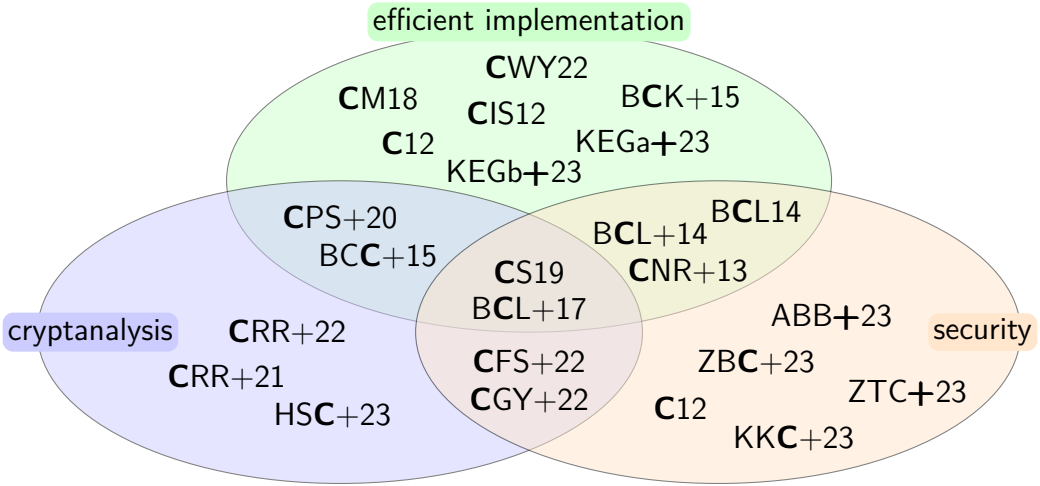
Efficient and Secure Cryptosystems

efficient implementation

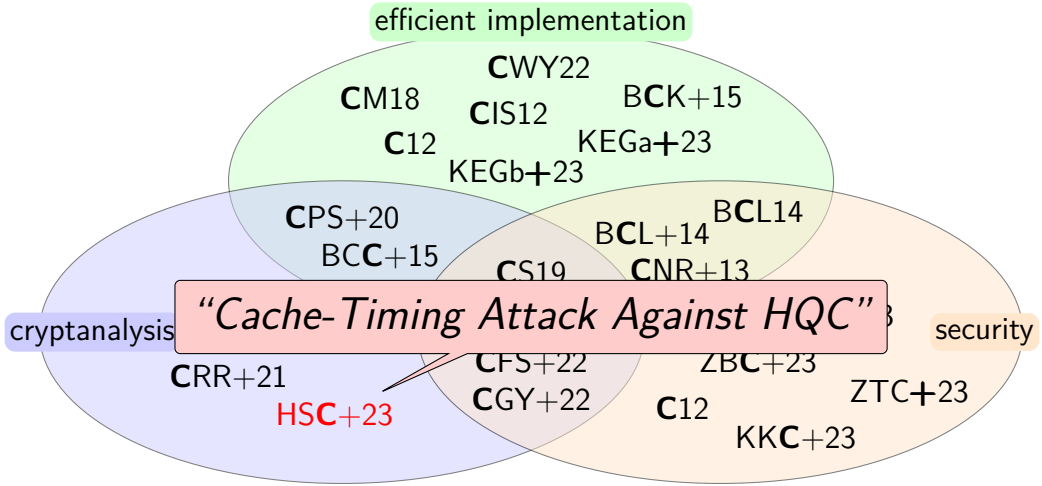
cryptanalysis

security

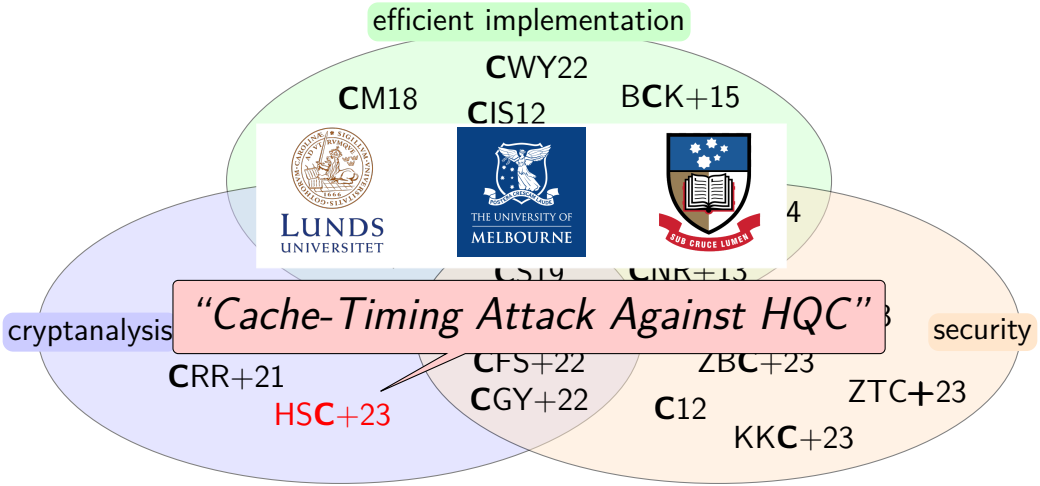
Efficient and Secure Cryptosystems



Efficient and Secure Cryptosystems



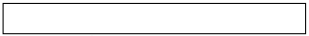
Efficient and Secure Cryptosystems



Joint work with Senyang Huang, Rui Qi Sim, Qian Guo, and Thomas Johansson

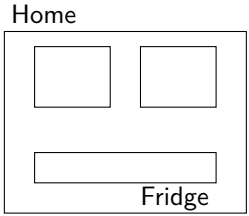
Optimization Strategy

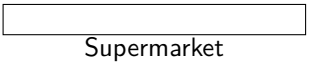
Optimization Strategy



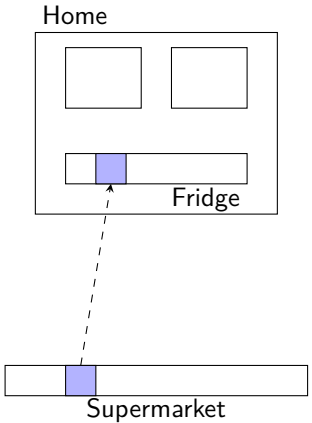
Supermarket

Optimization Strategy

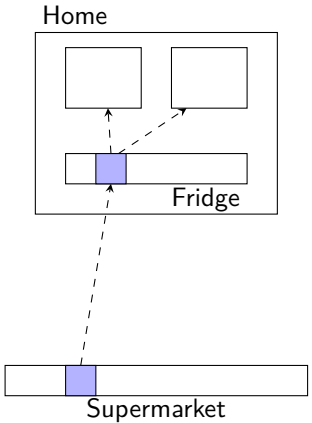




Optimization Strategy

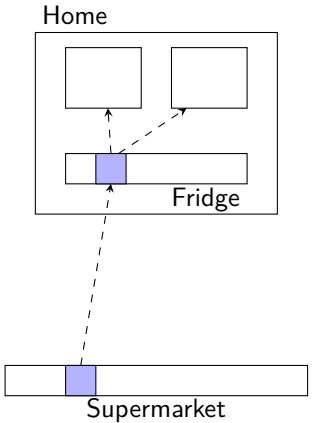


Optimization Strategy



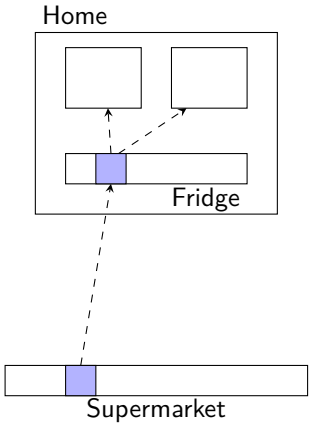
Optimization Strategy

- Exchanging products at a supermarket shortens customers' time



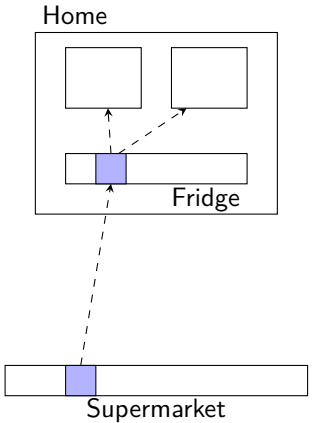
Optimization Strategy

- Exchanging products at a supermarket shortens customers' time
- Storing food in a fridge to avoid frequent visits to a supermarket



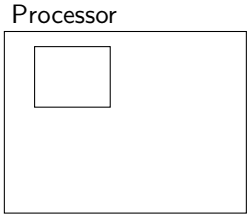
Optimization Strategy

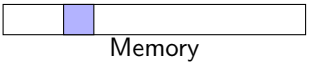
- Exchanging products at a supermarket shortens customers' time
- Storing food in a fridge to avoid frequent visits to a supermarket
- Observing shopping patterns can deduce food menus or recipes



Speeding Up Performance

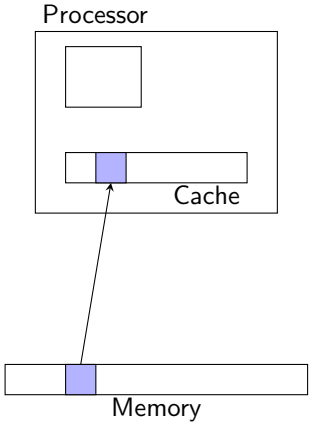
Speeding Up Performance





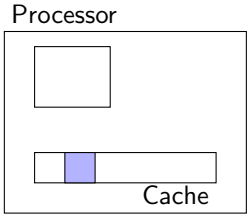
Speeding Up Performance

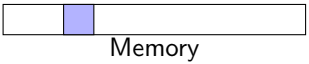
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache



Speeding Up Performance

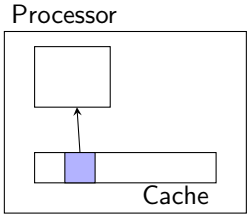
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache

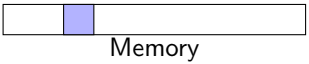




Speeding Up Performance

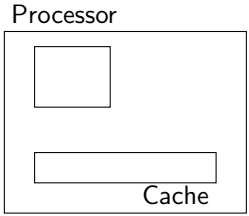
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache
- Check if data is in cache
  - cache hit: serve data from cache
  - cache miss: get data from memory and put in cache

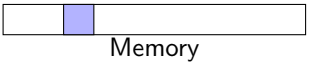




Speeding Up Performance

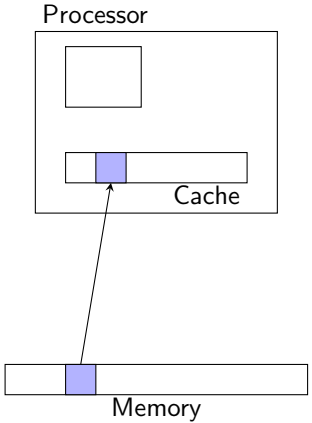
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache
- Check if data is in cache
  - cache hit: serve data from cache
  - cache miss: get data from memory and put in cache





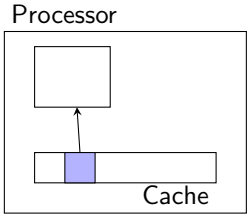
Speeding Up Performance

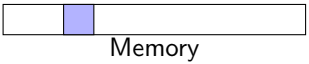
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache
- Check if data is in cache
  - cache hit: serve data from cache
  - cache miss: get data from memory and put in cache



Speeding Up Performance

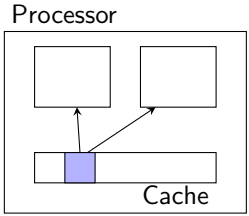
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache
- Check if data is in cache
  - cache hit: serve data from cache
  - cache miss: get data from memory and put in cache

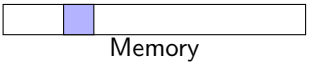




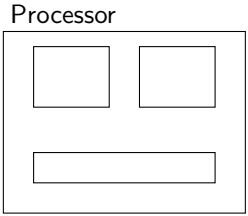
Speeding Up Performance

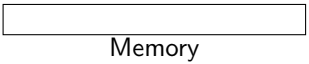
- Utilize locality
  - spatial: divide memory into lines
  - temporal: store recently used lines in cache
- Check if data is in cache
  - cache hit: serve data from cache
  - cache miss: get data from memory and put in cache
- Share cache
  - improve performance of multi-core processors





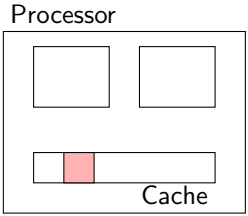
Maintaining Consistency

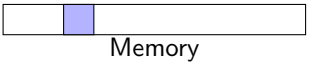




Maintaining Consistency

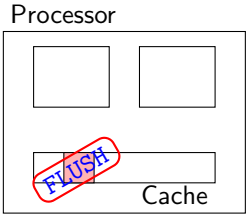
- Memory and cache can be inconsistent
  - rare but possible

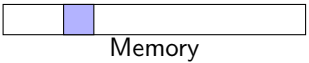




Maintaining Consistency

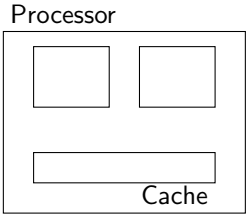
- Memory and cache can be inconsistent
  - rare but possible
- Solution: flush cache contents
  - ensure next load to serve from memory

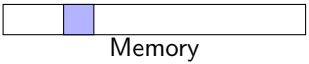




Maintaining Consistency

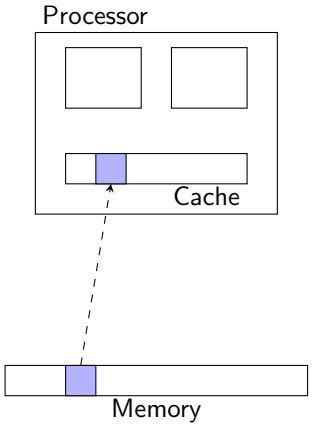
- Memory and cache can be inconsistent
  - rare but possible
- Solution: flush cache contents
  - ensure next load to serve from memory





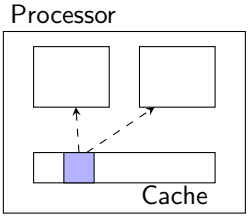
Maintaining Consistency

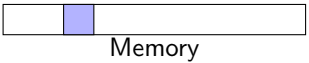
- Memory and cache can be inconsistent
  - rare but possible
- Solution: flush cache contents
  - ensure next load to serve from memory



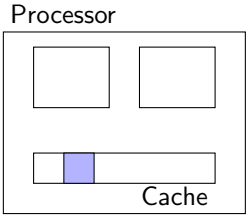
Maintaining Consistency

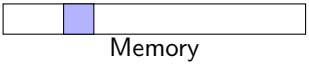
- Memory and cache can be inconsistent
  - rare but possible
- Solution: flush cache contents
  - ensure next load to serve from memory



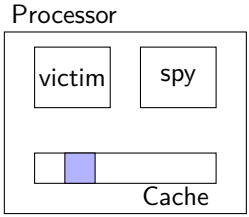


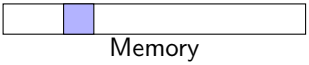
Flush + Reload [GBK11, YF14]





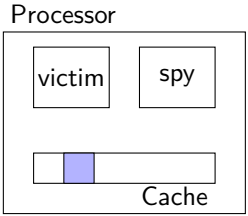
Flush + Reload [GBK11, YF14]

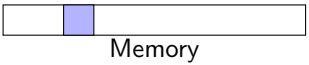




Flush + Reload [GBK11, YF14]

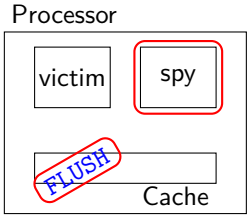
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)

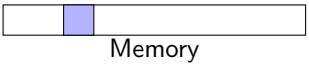




Flush + Reload [GBK11, YF14]

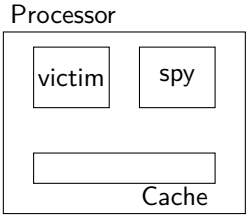
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)

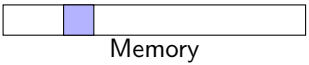




Flush + Reload [GBK11, YF14]

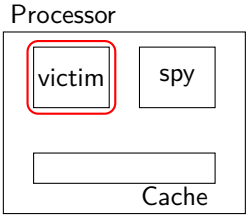
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)

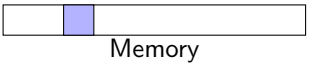




Flush + Reload [GBK11, YF14]

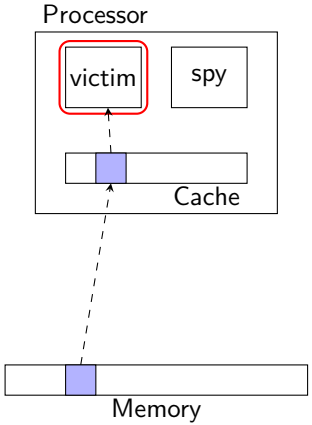
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)





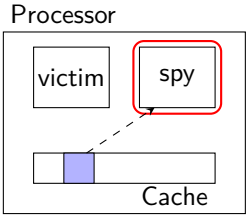
Flush + Reload [GBK11, YF14]

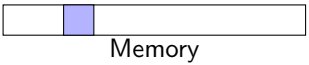
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)



Flush + Reload [GBK11, YF14]

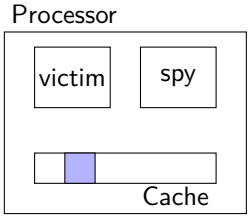
- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)

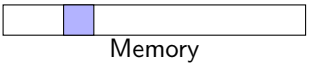




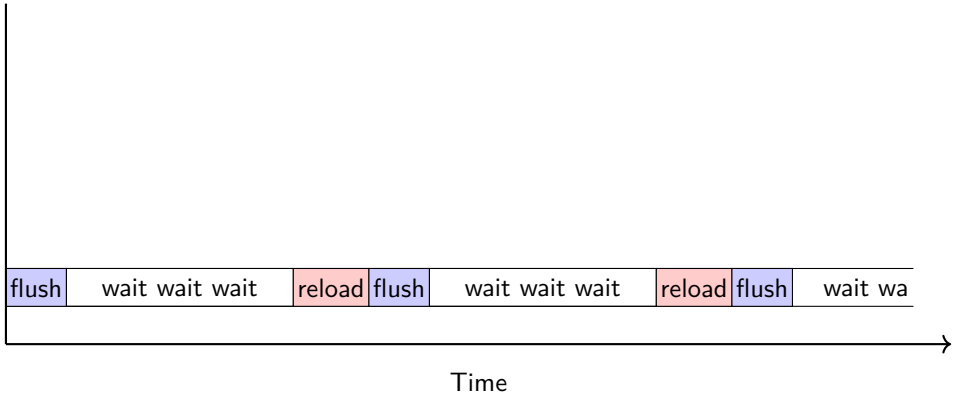
Flush + Reload [GBK11, YF14]

- **Flush** memory line
- Wait (victim executes)
- Measure time to **Reload** line
  - slow → no victim access
  - fast → victim accessed
- (Repeat)

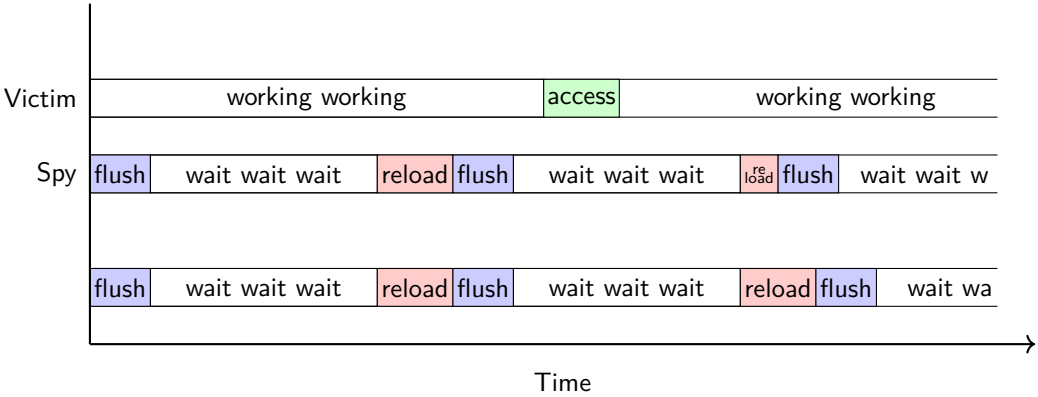




Flush + Reload: in action



Flush + Reload: in action



Flush + Reload: pros & cons

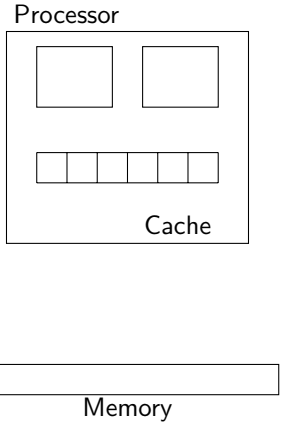
+ Simple
+ Very fews false positive
+ Resolution of memory line

Flush + Reload: pros & cons

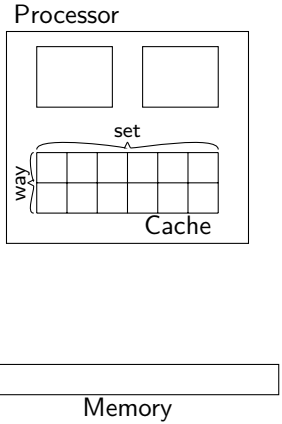
+ Simple
+ Very fews false positive
+ Resolution of memory line

- Only work with shared memory
- Require flush instruction

Set Associative Caches

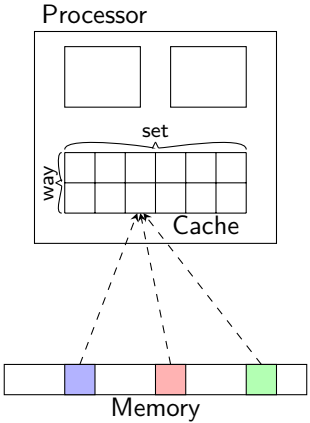


Set Associative Caches



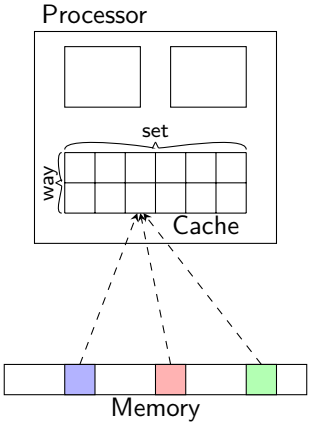
Set Associative Caches

- Memory lines map to cache sets



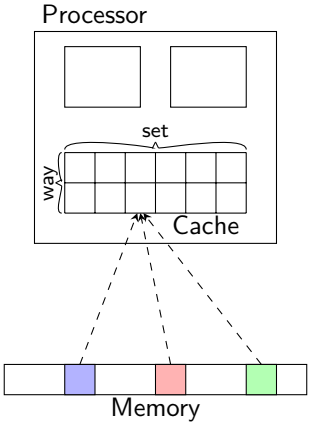
Set Associative Caches

- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways



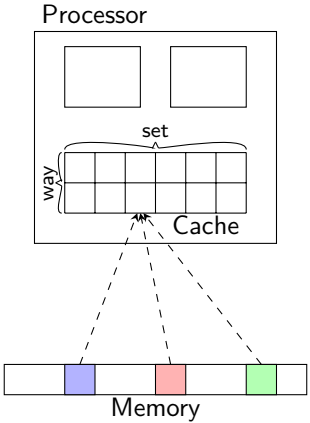
Set Associative Caches

- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set



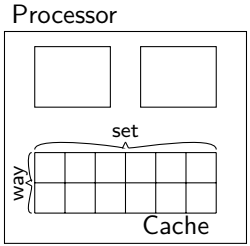
Set Associative Caches

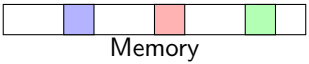
- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set
- When cache miss and that set is full, one of the lines in that set is evicted



Set Associative Caches

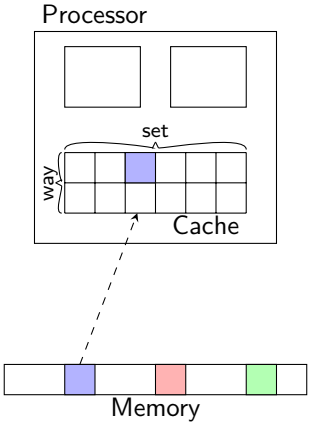
- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set
- When cache miss and that set is full, one of the lines in that set is evicted





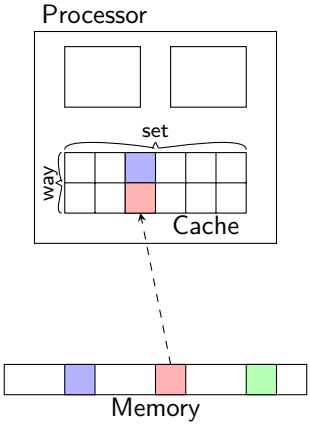
Set Associative Caches

- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set
- When cache miss and that set is full, one of the lines in that set is evicted



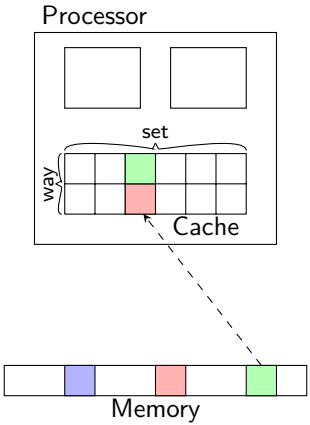
Set Associative Caches

- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set
- When cache miss and that set is full, one of the lines in that set is evicted



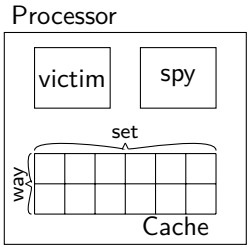
Set Associative Caches

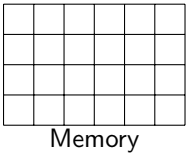
- Memory lines map to cache sets
- Each line maps to one particular set and can be stored in any of its ways
- Multiple lines map to the same set
- When cache miss and that set is full, one of the lines in that set is evicted



Prime + Probe [OST05, Per05]

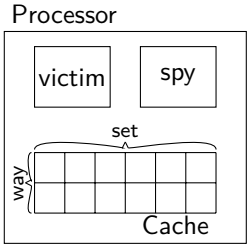
Prime + Probe [OST05, Per05]

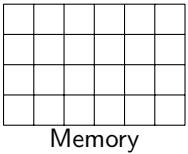




Prime + Probe [OST05, Per05]

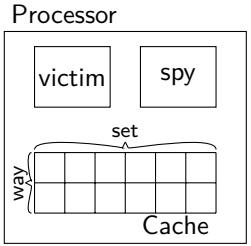
- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access

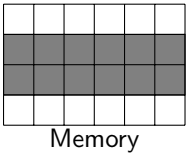




Prime + Probe [OST05, Per05]

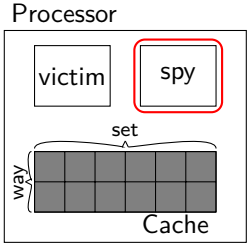
- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access

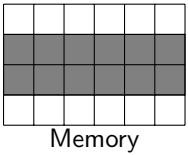




Prime + Probe [OST05, Per05]

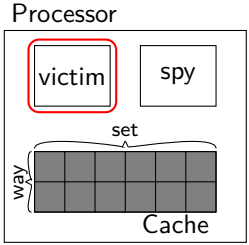
- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access

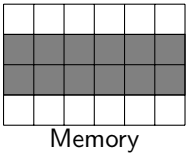




Prime + Probe [OST05, Per05]

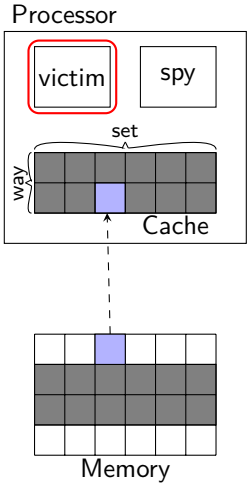
- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access





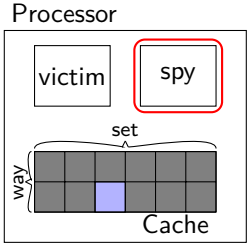
Prime + Probe [OST05, Per05]

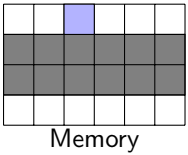
- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access



Prime + Probe [OST05, Per05]

- (Allocate cache-sized memory buffer)
- **Prime**: access all lines in buffer
  $\rightarrow$ fill cache with attacker's data
- Wait (victim executes)
  $\rightarrow$ may evict some cache lines
- **Probe**: measure time to access buffer
  - slow $\rightarrow$ cache line evicted
  - fast $\rightarrow$ no victim access





Limitations of Cache-Timing Attacks

Limitations of Cache-Timing Attacks

- Provide cache/memory access patterns
  - might not reveal actual secret values

Limitations of Cache-Timing Attacks

- Provide cache/memory access patterns
  - might not reveal actual secret values
- Provide partial information
  - might not reveal the full secret key

HQC

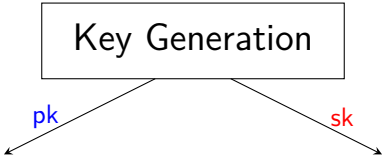
- HQC: Hamming Quasi Cyclic
- 4th-round candidate of the ongoing NIST PQC Standardization
- Code-based post-quantum key encapsulation mechanism (KEM)

HQC's KEM

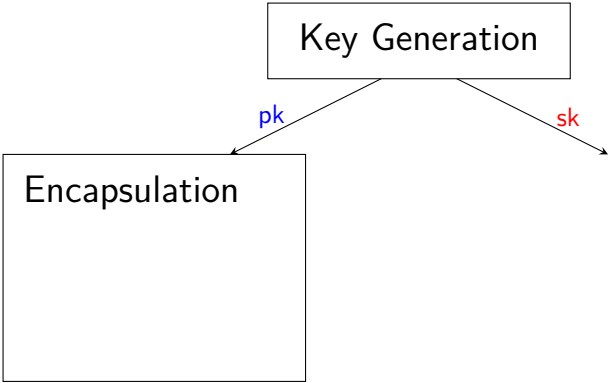
HQC's KEM

Key Generation

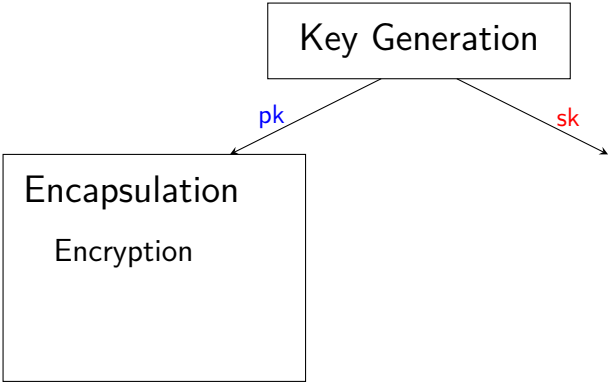
HQC's KEM



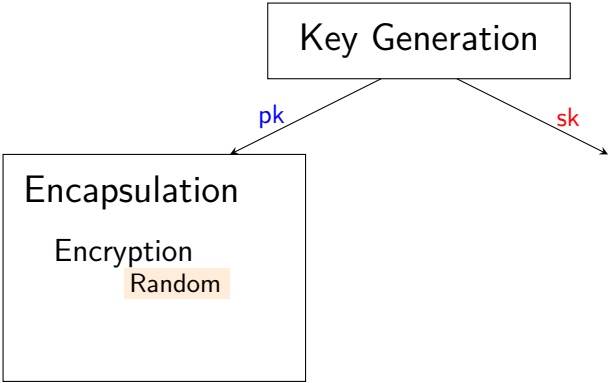
HQC's KEM



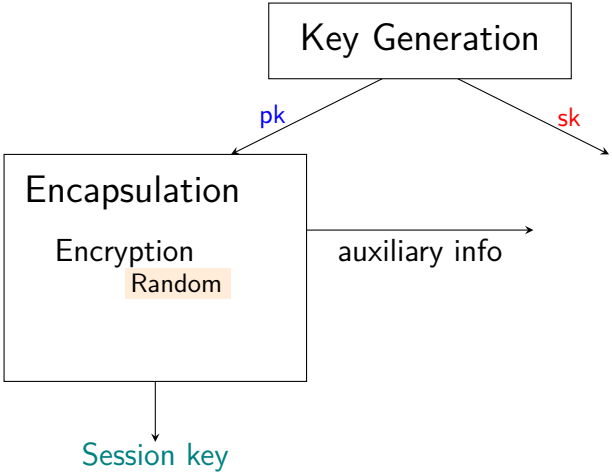
HQC's KEM



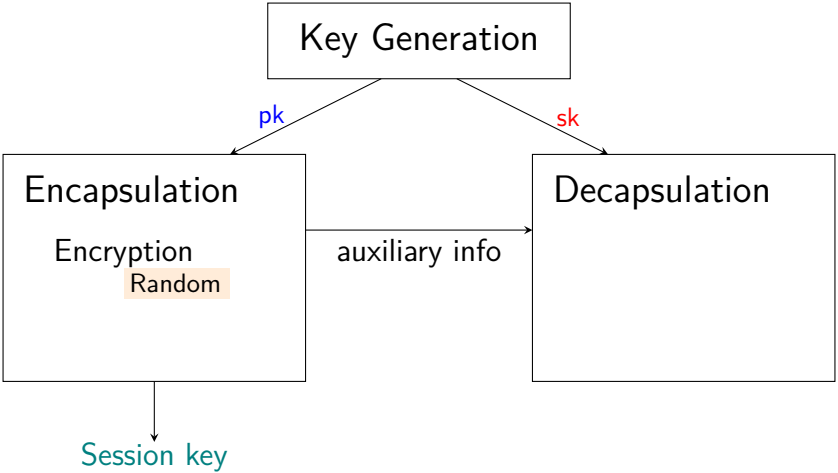
HQC's KEM



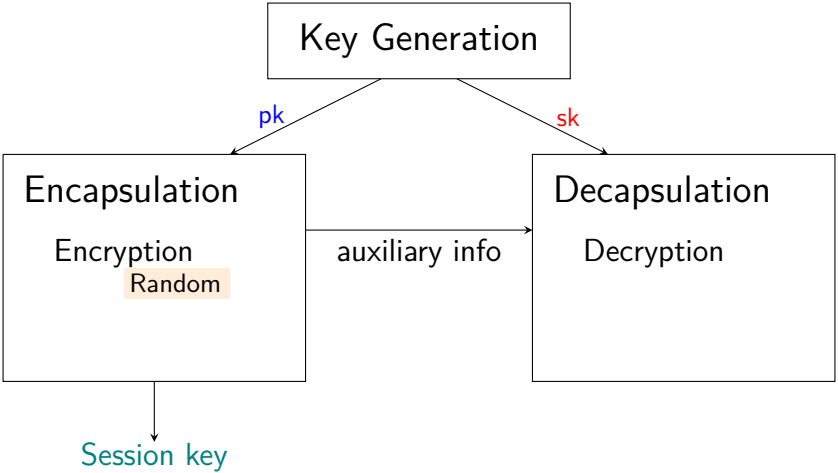
HQC's KEM


Key Generation
pk
sk
Encapsulation
Encryption
Random
auxiliary info
Session key

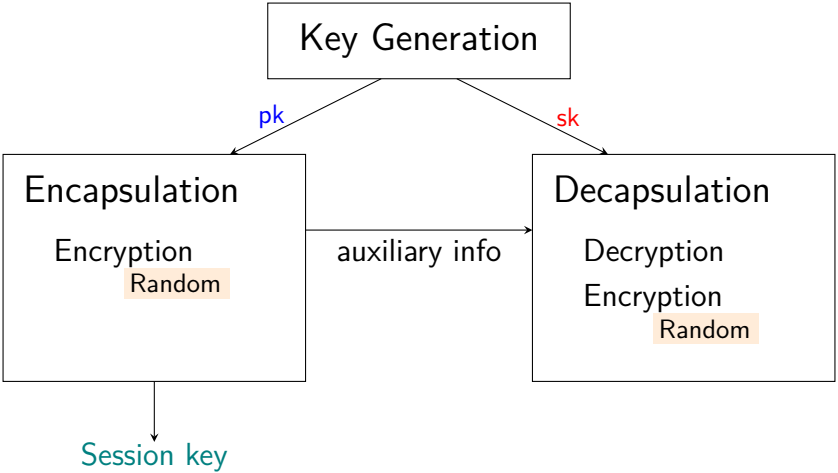
HQC's KEM


Key Generation
pk
sk
Encapsulation
Encryption
Random
auxiliary info
Decapsulation
Session key

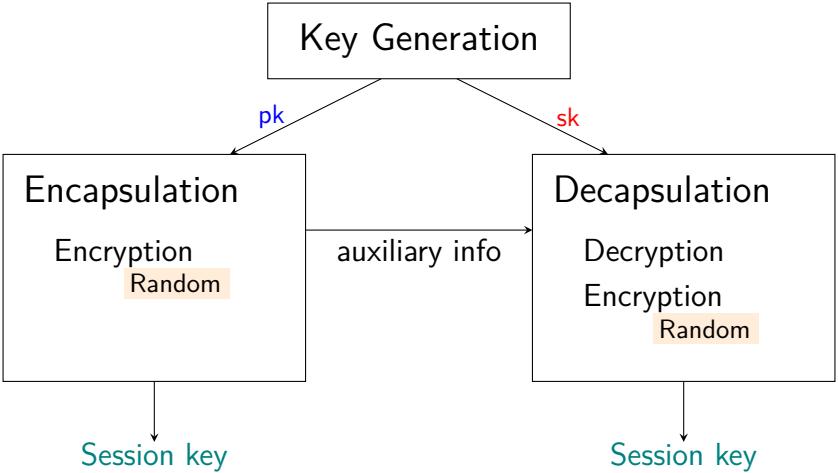
HQC's KEM


Key Generation
pk
sk
Encapsulation
Encryption
Random
auxiliary info
Decapsulation
Decryption
Session key

HQC's KEM


Key Generation
pk
sk
Encapsulation
Encryption
Random
auxiliary info
Decapsulation
Decryption
Encryption
Random
Session key

HQC's KEM


Key Generation
pk
sk
Encapsulation
Encryption
Random
auxiliary info
Decapsulation
Decryption
Encryption
Random
Session key
Session key

Randomness

Randomness

Encryption

Randomness

Encryption

$c = m + \text{error}$

Randomness

Encryption

$$c = m + \underbrace{\text{error}}_{\text{random+sk}}$$

Randomness

Encryption

$c = m + \underbrace{\text{error}}_{\text{random+sk}}$

Decryption

Randomness

Encryption

$c = m + \underbrace{\text{error}}_{\text{random+sk}}$

Decryption

c

Randomness

Encryption

$c = m + \underbrace{\text{error}}_{\text{random+sk}}$

Decryption

c

$\uparrow$

$m+\text{error}$

Randomness

Encryption

$c = m + \text{error}$

random+sk

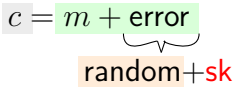
Decryption

$c - \text{error'}$

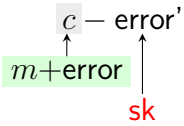
m+error

Randomness

Encryption



Decryption



Randomness

Encryption

$c = m + \text{error}$

random+sk

Decryption

$m' = c - \text{error'}$

m+error

sk

Randomness

Encryption

$c = m + \underbrace{\text{error}}_{\text{random+sk}}$

Decryption

$m' = c - \text{error'}$

$m + \text{error}$ ↑ c, sk ↑ error'

$m = m'$ vs $m \neq m'$

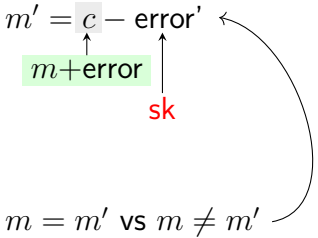
Randomness

Encryption

$c = m + \text{error}$

random+sk

Decryption



Cache Line Indicator

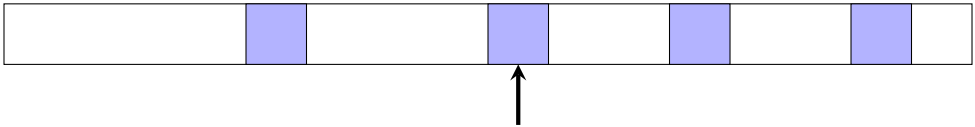
Cache Line Indicator

Cache Line Indicator

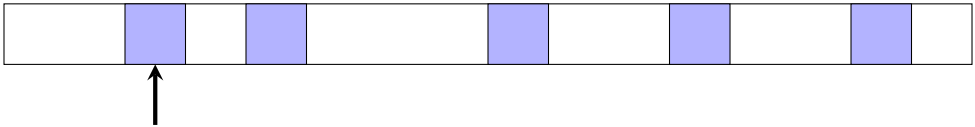
Cache Line Indicator

Cache Line Indicator

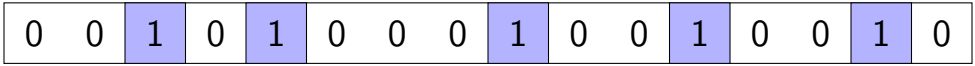
Cache Line Indicator



Cache Line Indicator

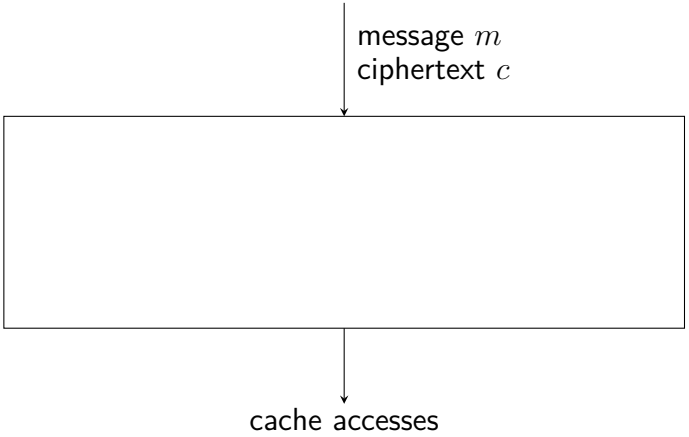


Cache Line Indicator

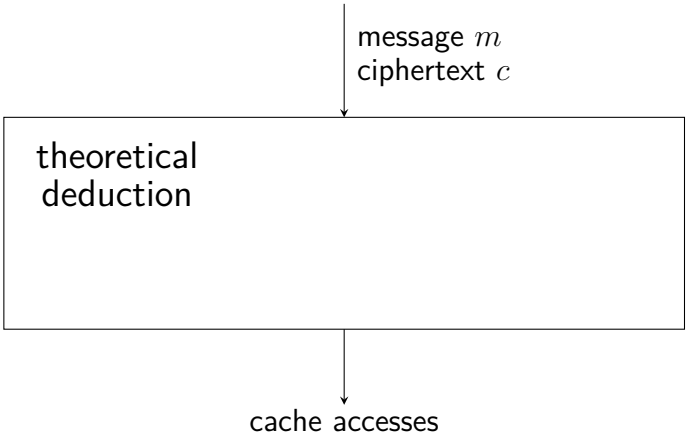


Plaintext-Checking Oracle

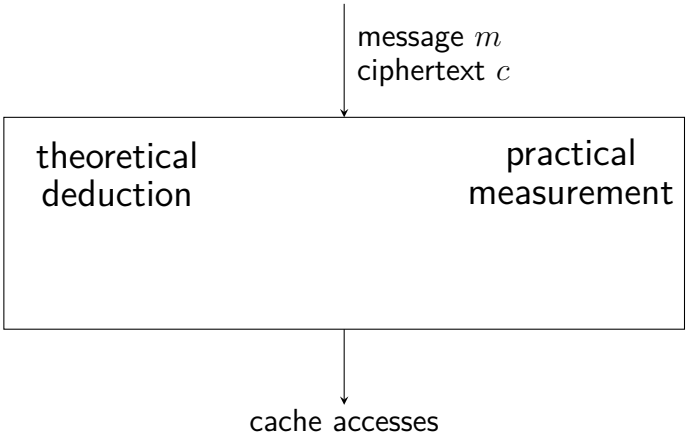
Plaintext-Checking Oracle



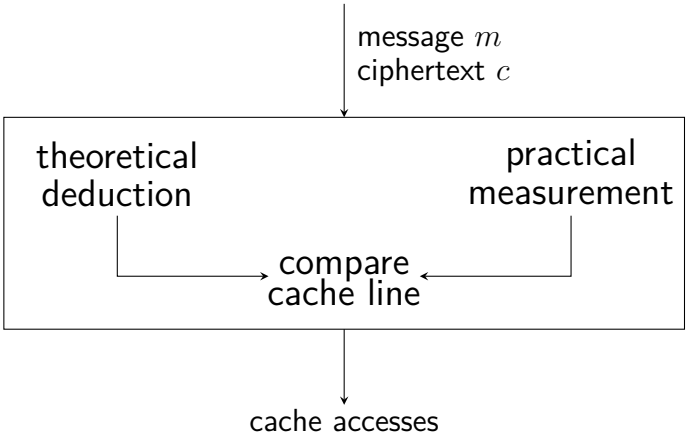
Plaintext-Checking Oracle



Plaintext-Checking Oracle

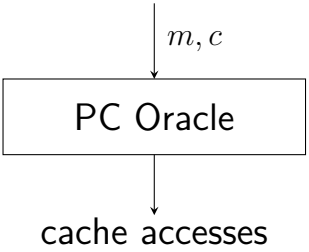


Plaintext-Checking Oracle

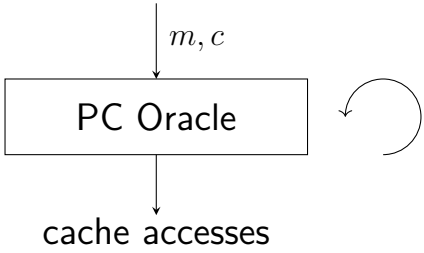


Our Attack

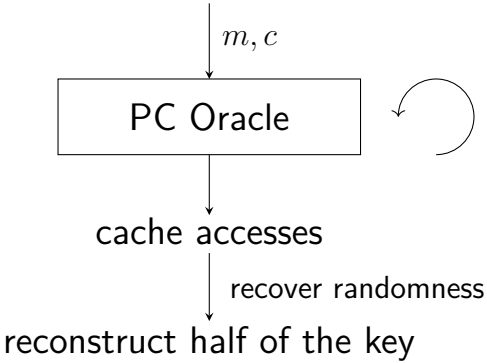
Our Attack



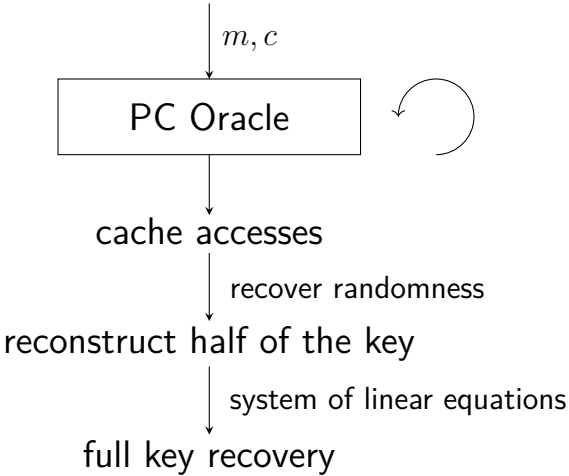
Our Attack



Our Attack


m, c
PC Oracle
cache accesses
recover randomness
reconstruct half of the key

Our Attack


m, c
PC Oracle
cache accesses
recover randomness
reconstruct half of the key
system of linear equations
full key recovery

Practical Experiment

Practical Experiment

- Code runs inside a secure Software Guard Extensions (SGX)

Practical Experiment

- Code runs inside a secure Software Guard Extensions (SGX)
- Assume a malicious OS under attacker's control

Practical Experiment

- Code runs inside a secure Software Guard Extensions (SGX)
- Assume a malicious OS under attacker's control
- Use SGX-Step to single-step an SGX enclave

Practical Experiment

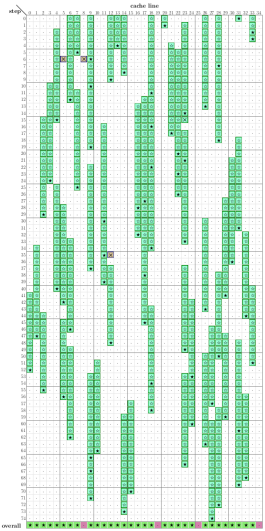
- Code runs inside a secure Software Guard Extensions (SGX)
- Assume a malicious OS under attacker's control
- Use SGX-Step to single-step an SGX enclave
- Aim: detect *whether* there is an access rather than *when* it occurs

Practical Experiment

- Code runs inside a secure Software Guard Extensions (SGX)
- Assume a malicious OS under attacker's control
- Use SGX-Step to single-step an SGX enclave
- Aim: detect *whether* there is an access rather than *when* it occurs
- Result: 99.9% accuracy

Captured Traces

Captured Traces



Comparison to ground truth

- ★ true positive
- · true negative
- × false positive
- ☆ speculatively accessed

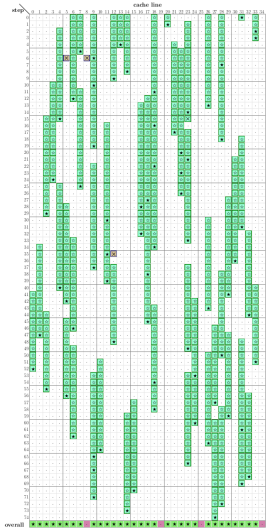
Simple analysis

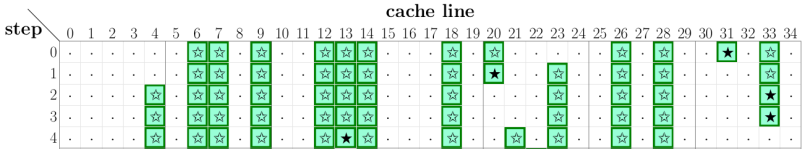
- accessed
- ignore access

Final classification

- accessed
- not accessed

Captured Traces





Comparison to ground truth

- ★ true positive
- · true negative
- ✕ false positive
- ☆ speculatively accessed

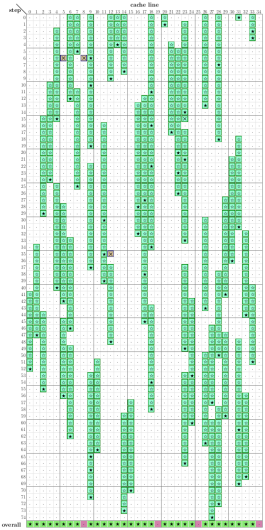
Simple analysis

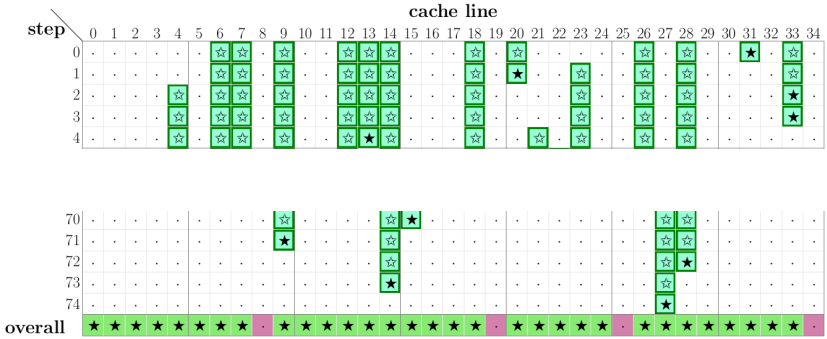
- accessed
- ignore access

Final classification

- accessed
- not accessed

Captured Traces





Comparison to ground truth

- ★ true positive
- · true negative
- × false positive
- ☆ speculatively accessed

Simple analysis

- accessed
- ignore access

Final classification

- accessed
- not accessed

Summary

Summary

- First cache-timing attack on HQC

Summary

- First cache-timing attack on HQC
- More practical than previous timing attack (54k vs 866k queries)

Summary

- First cache-timing attack on HQC
- More practical than previous timing attack (54k vs 866k queries)
- See paper for more details
  https://eprint.iacr.org/2023/102.pdf