GoFetch: Breaking Constant-Time Cryptographic Implementations Using Data Memory-Dependent Prefetchers

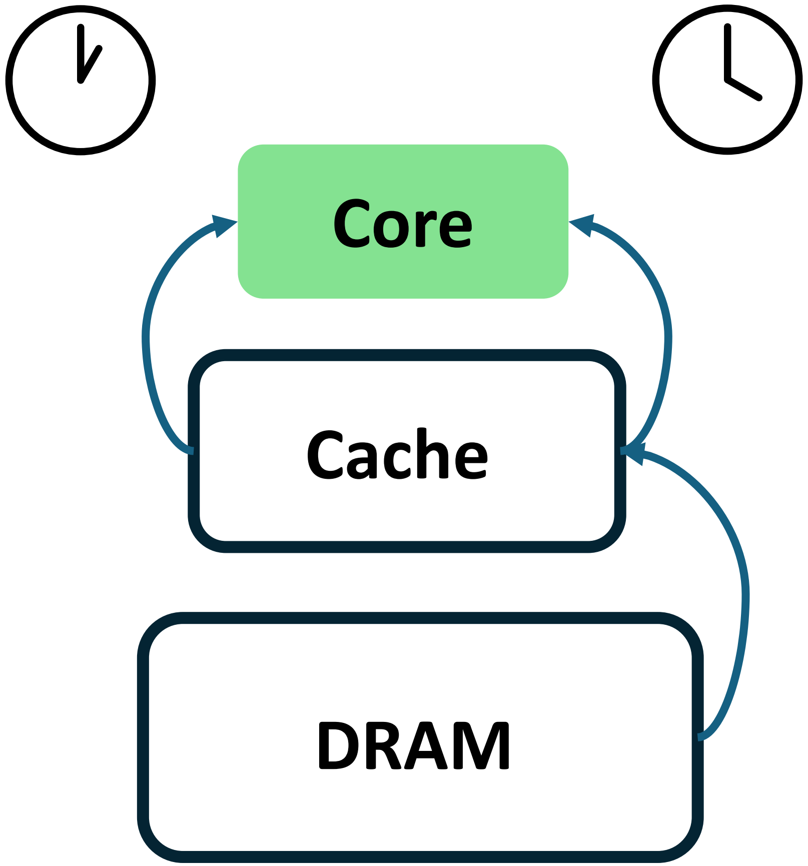
Boru Chen, Yingchen Wang, Pradyumna Shome,

Christopher Fletcher, David Kohlbrenner, Riccardo Paccagnella,

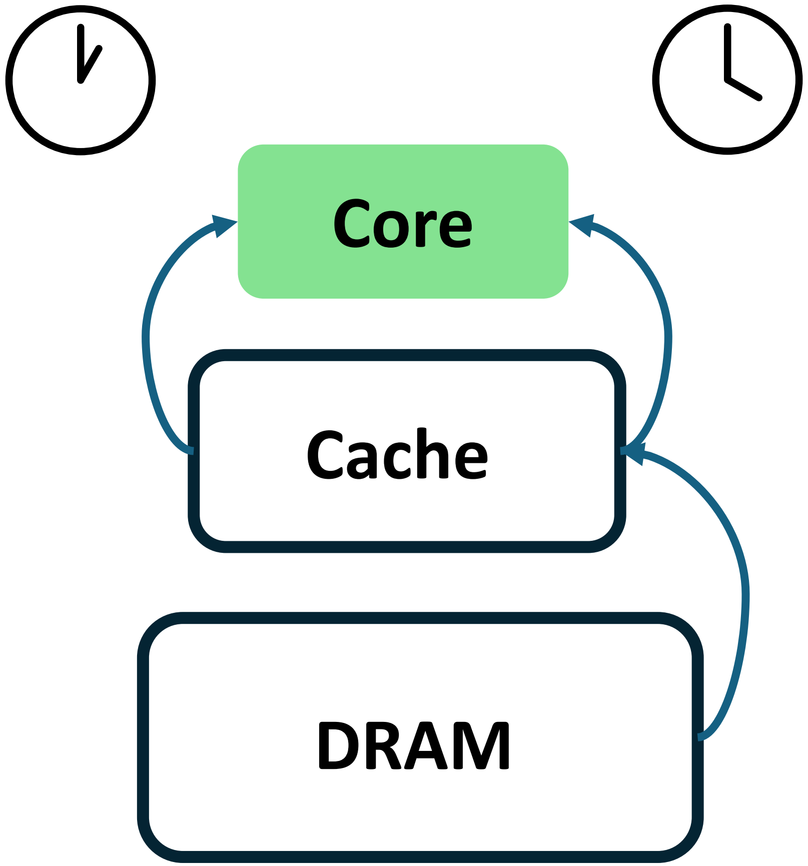
Daniel Genkin

Timing Attacks

Timing Attacks

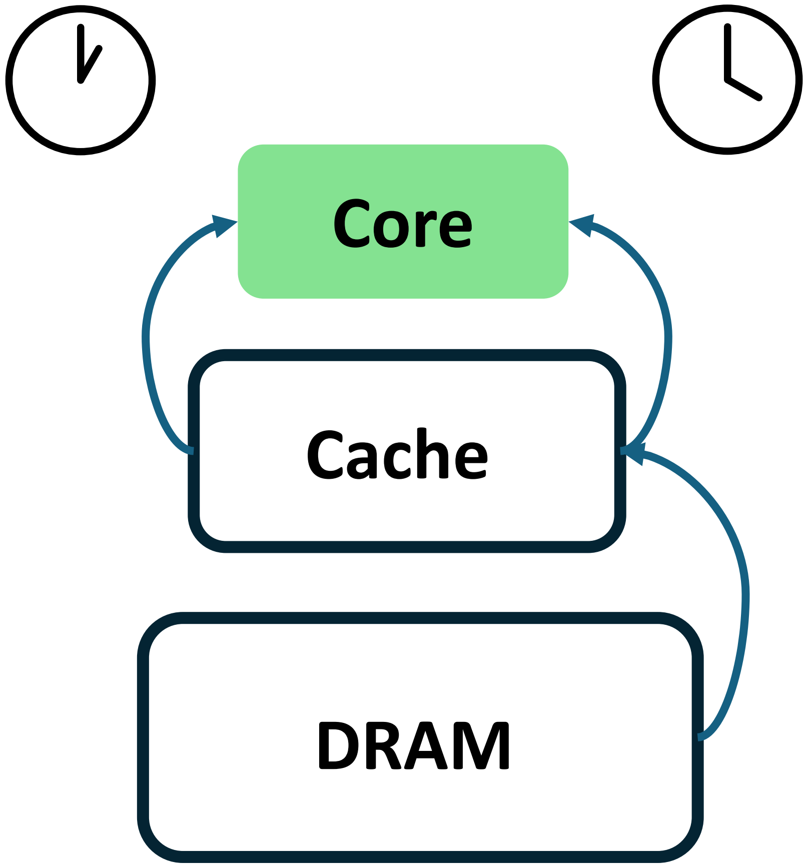


Timing Attacks

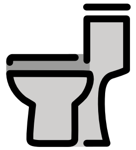


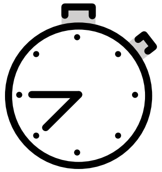

```
// secret = 1 or 0
if (secret)
{
  trash = *addr_A
}
```

Timing Attacks

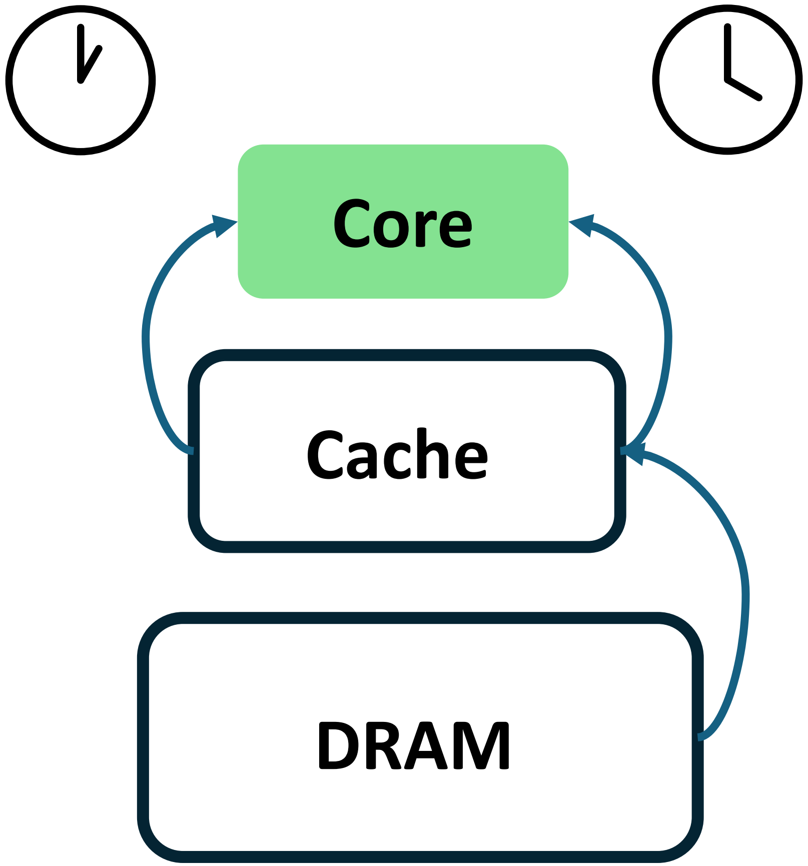



```
// secret = 1 or 0
if (secret)
{
   trash = *addr_A
}
```

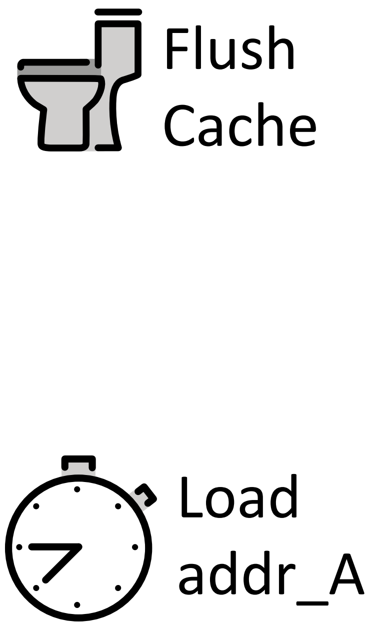


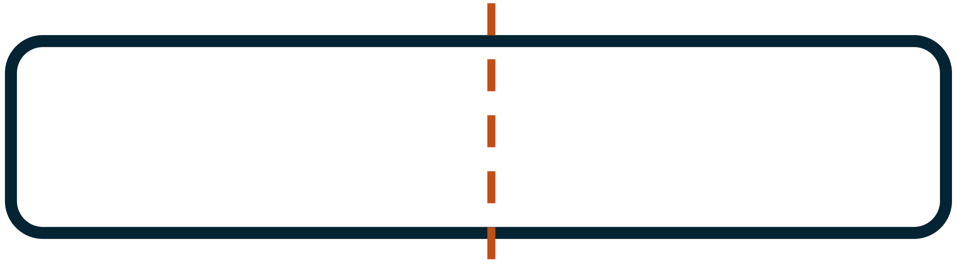


Timing Attacks



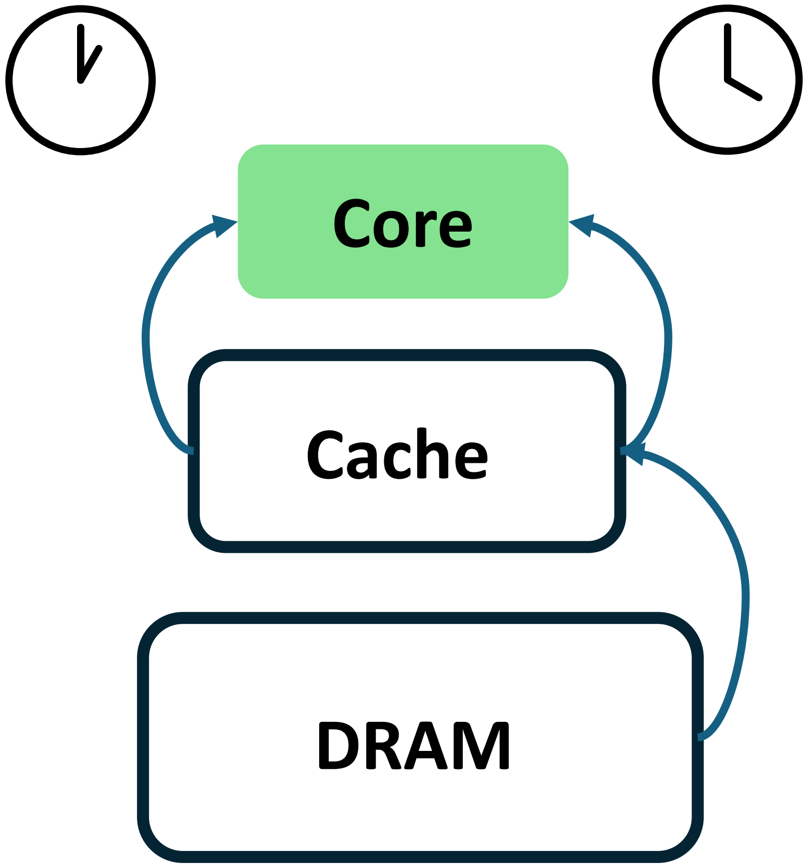

```
// secret = 1 or 0
if (secret)
{
  trash = *addr_A
}
```



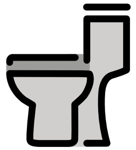


Cache

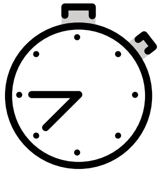
Timing Attacks



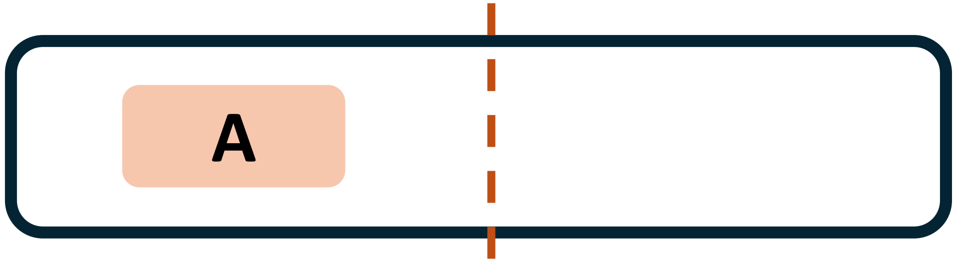

```
// secret = 1 or 0
if (secret)
{
   trash = *addr_A
}
```



Flush Cache



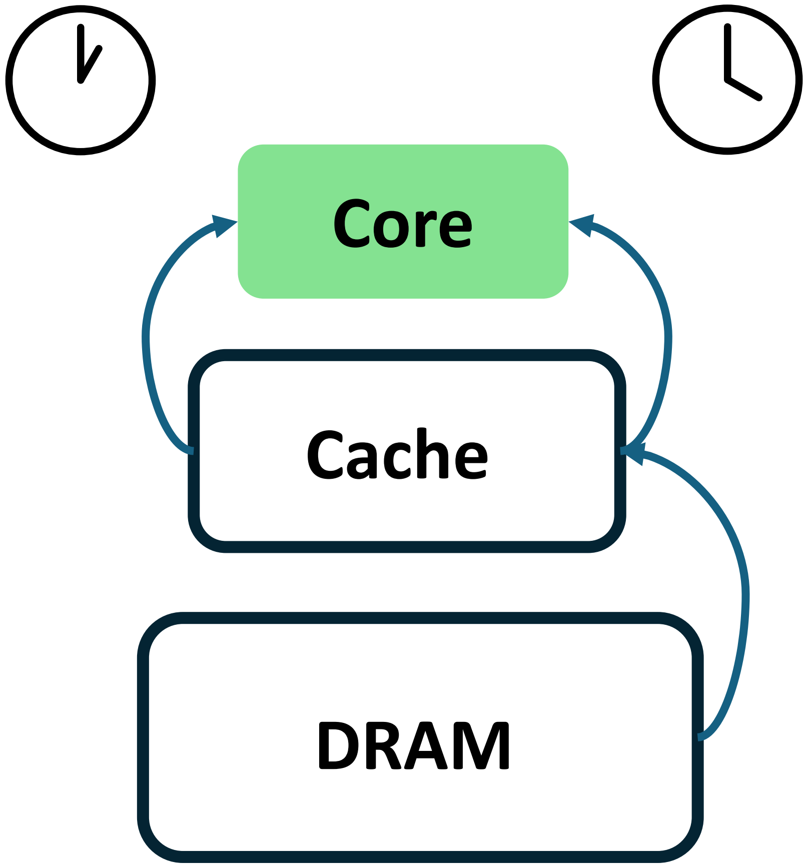
Load addr_A

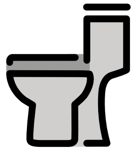


Cache

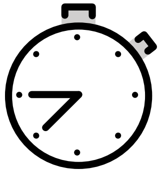
secret = 1

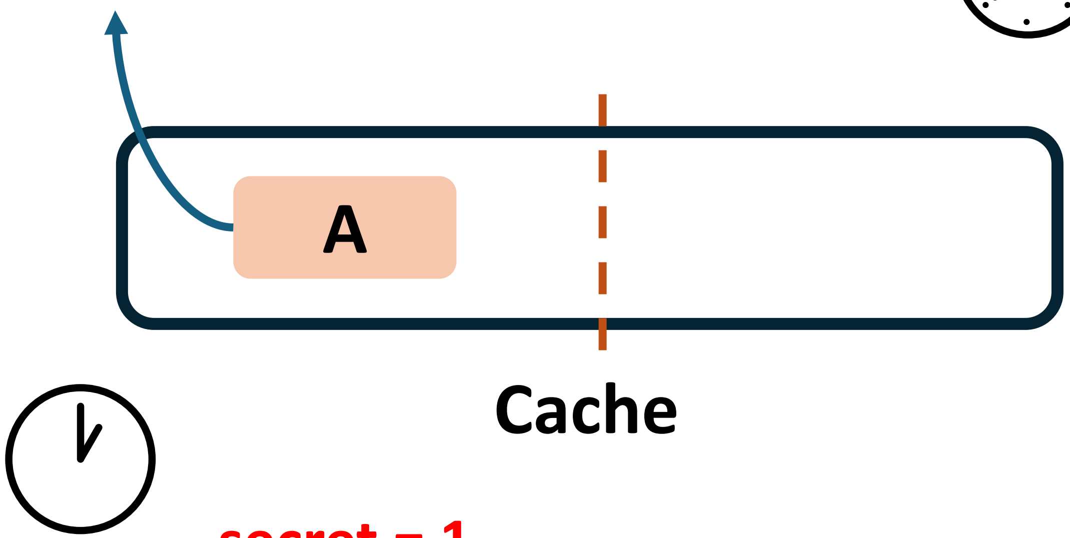
Timing Attacks





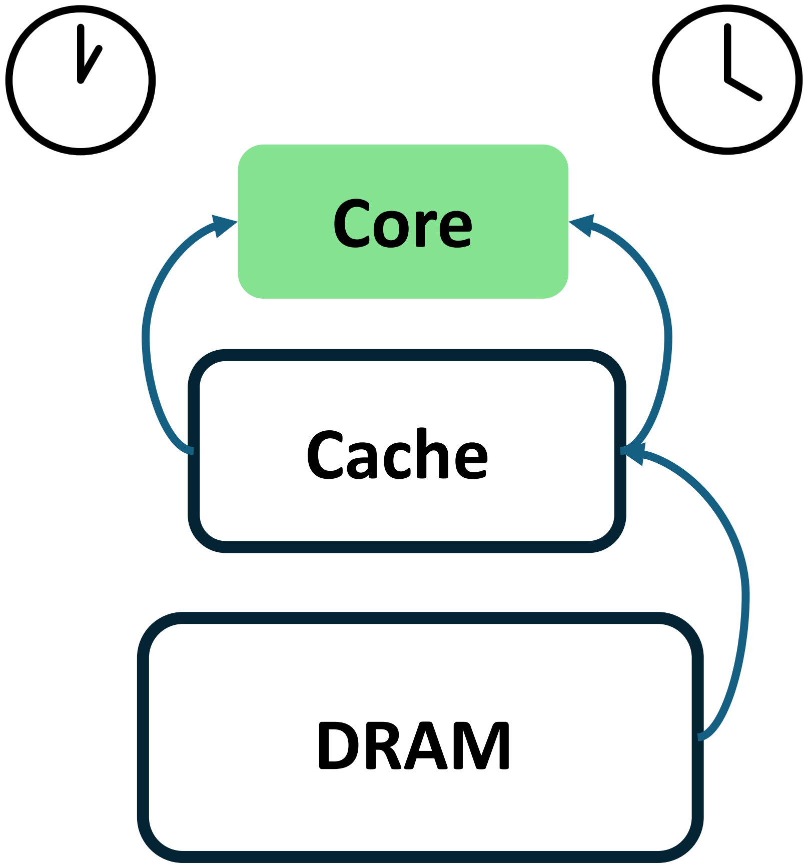

```
// secret = 1 or 0
if (secret)
{
   trash = *addr_A
}
```



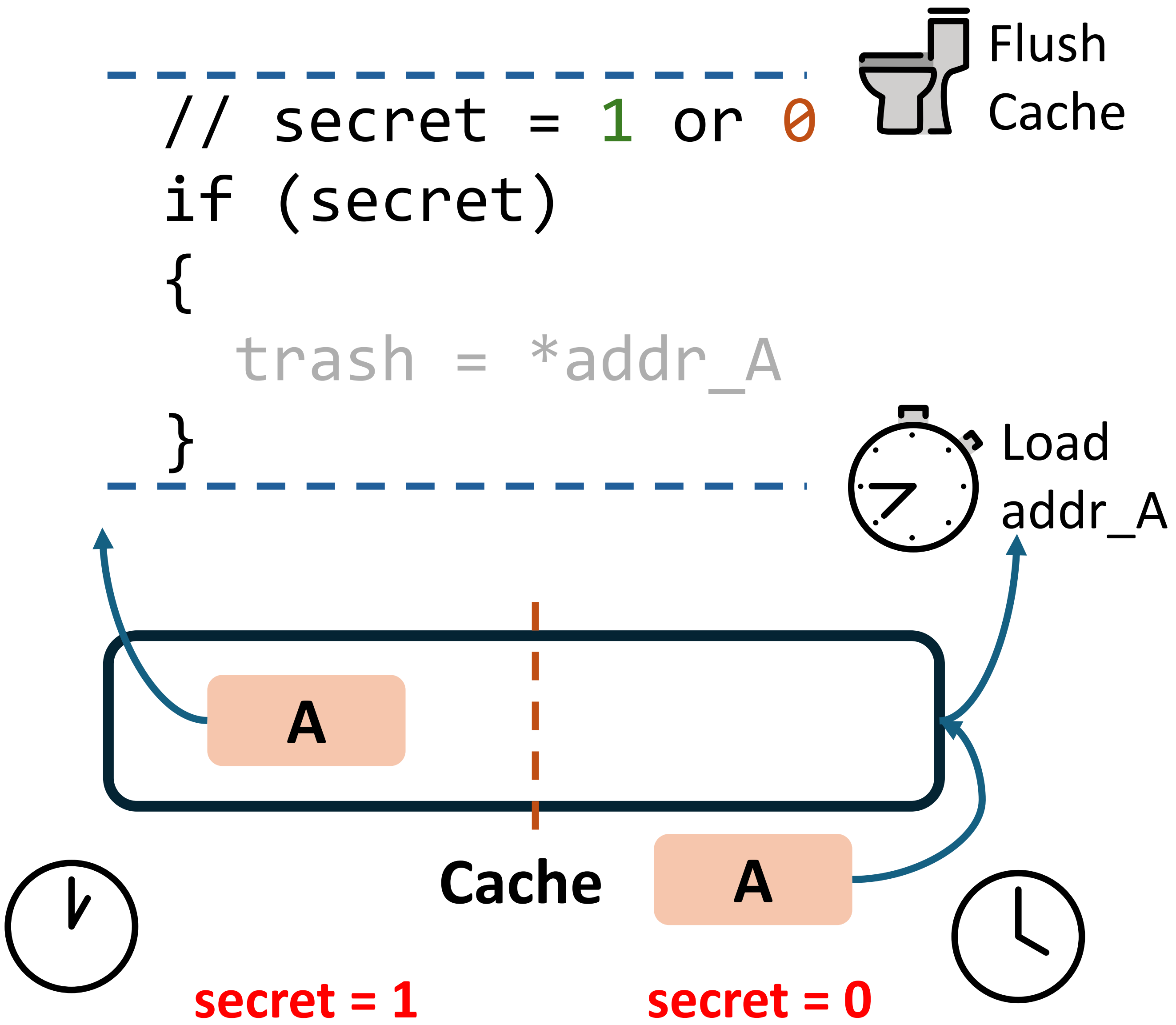


secret = 1

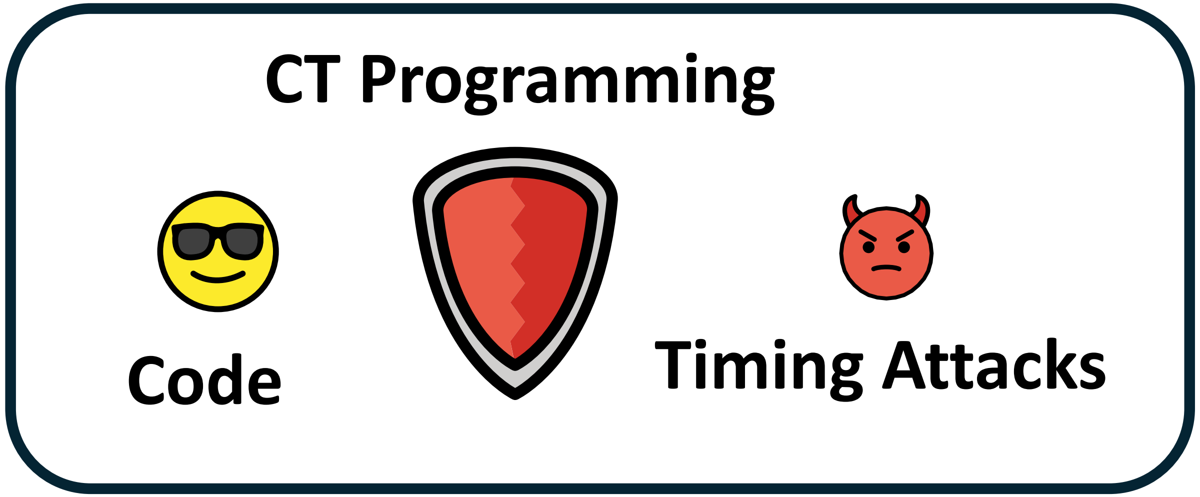
Timing Attacks



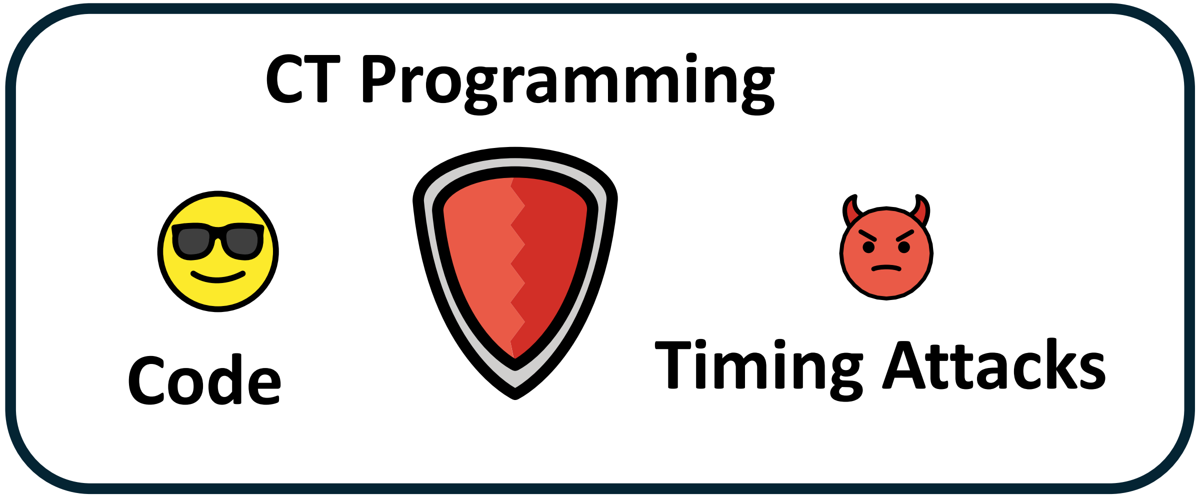

```
// secret = 1 or 0
if (secret)
{
   trash = *addr_A
}
```



Constant-Time Programming



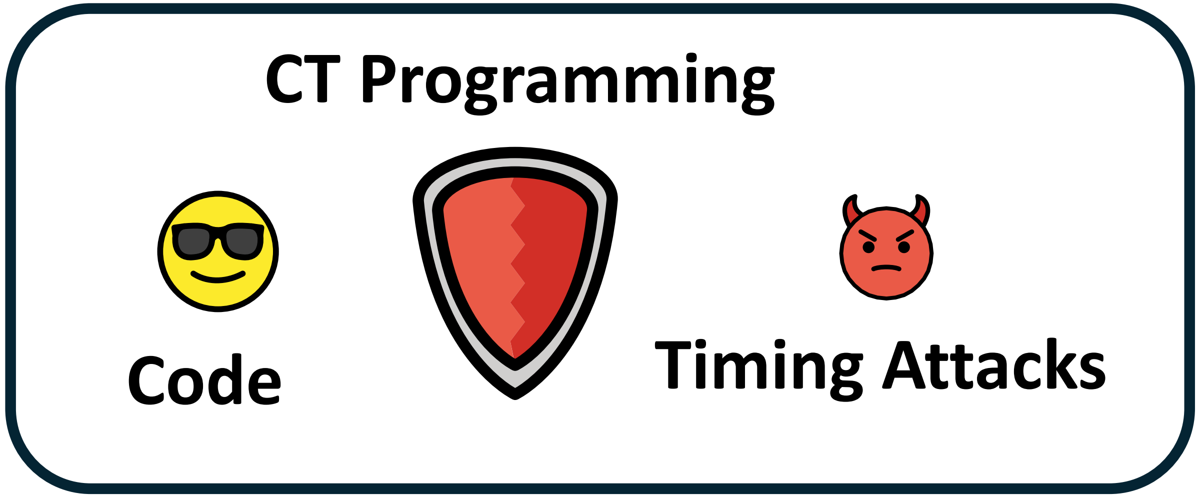
Constant-Time Programming

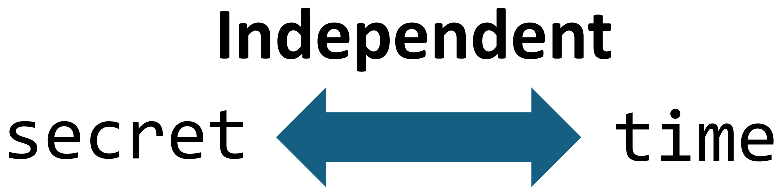


Independent

`secret` ⟷ `time`

Constant-Time Programming





Control-Flow

```
if (secret)
{
     ...
}
```

Time-Variable

```
secret / 100.0
```

Memory Address

```
trash = *secret
```

Constant-Time Programming

CT Programming

Code **Timing Attacks**

Independent

`secret` ⟷ `time`

Control-Flow

```
if (secret)
{
    ...
}
```

Time-Variable

```
secret / 100.0
```

Memory Address

```
trash = *secret
```

CT Cryptographic Implementation

Constant-Time Programming

<u>This talk:</u>
Show that these principles are insufficient.

Constant-Time Programming

This talk:
Show that these principles are insufficient.

Data Memory Dependent Prefetcher

Constant-Time Programming

<u>This talk:</u>

Show that these principles are insufficient.

Data Memory Dependent Prefetcher

```
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
```

program *without* secret-dependent load

Constant-Time Programming

<u>This talk:</u>

Show that these principles are insufficient.

Data Memory Dependent Prefetcher

```
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
```

program *without* secret-dependent load

*secret

*secret

Constant-Time Programming

<u>This talk:</u>
Show that these principles are insufficient.

Data Memory Dependent Prefetcher

```
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
secret = *non-sec-addr
```

program *without* secret-dependent load

*secret

*secret

Cache Timing Attacks

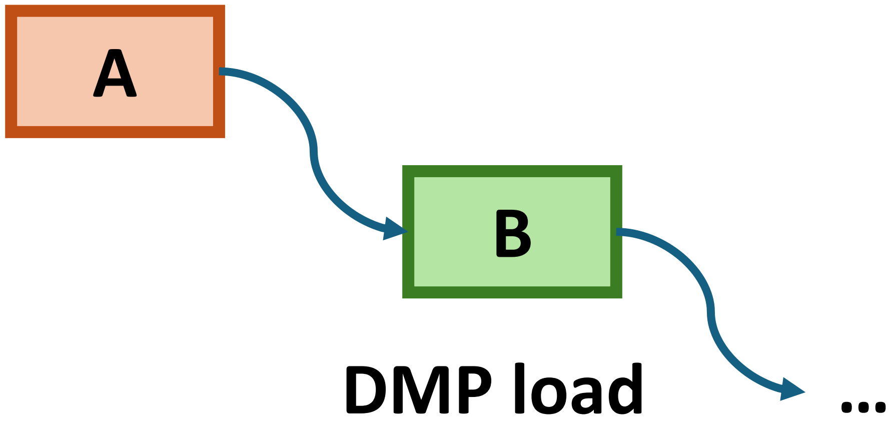
A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

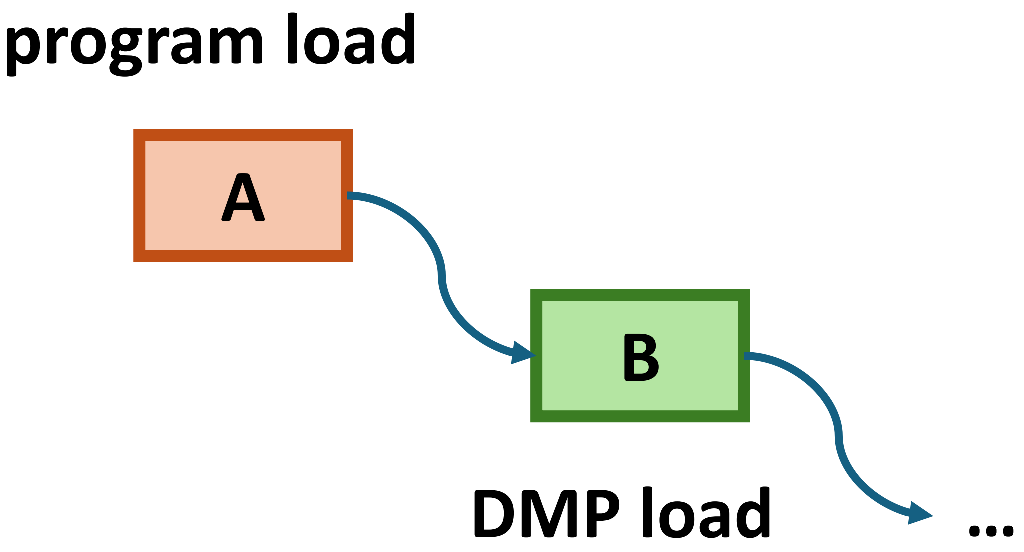
program load

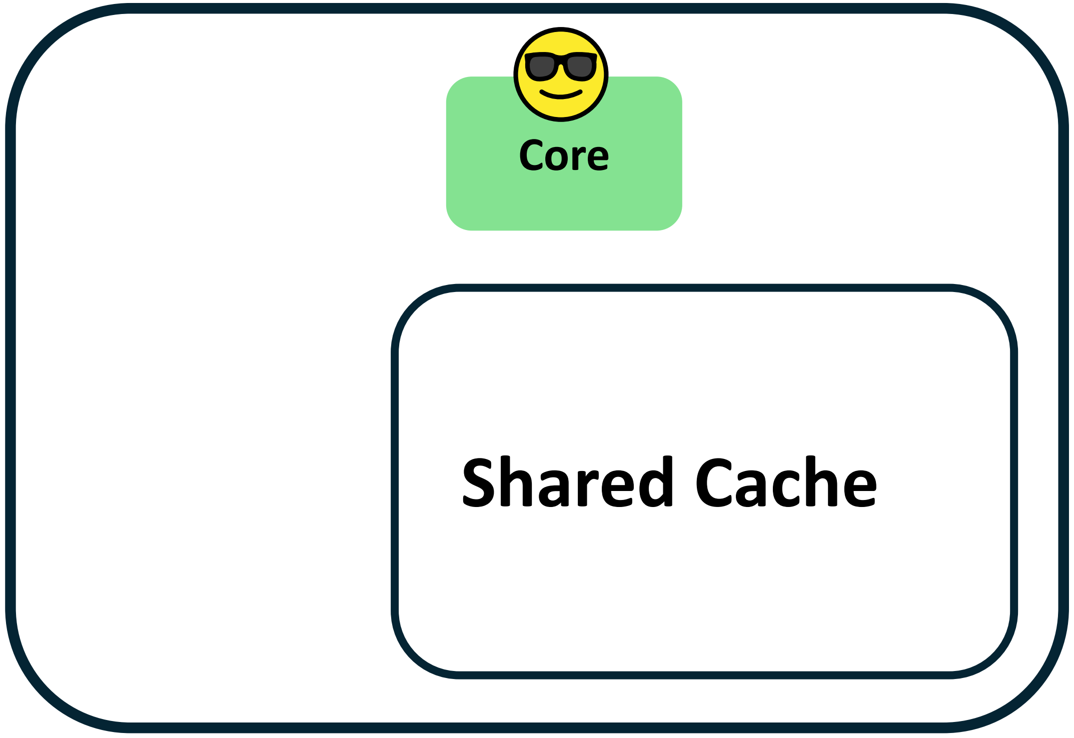


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```

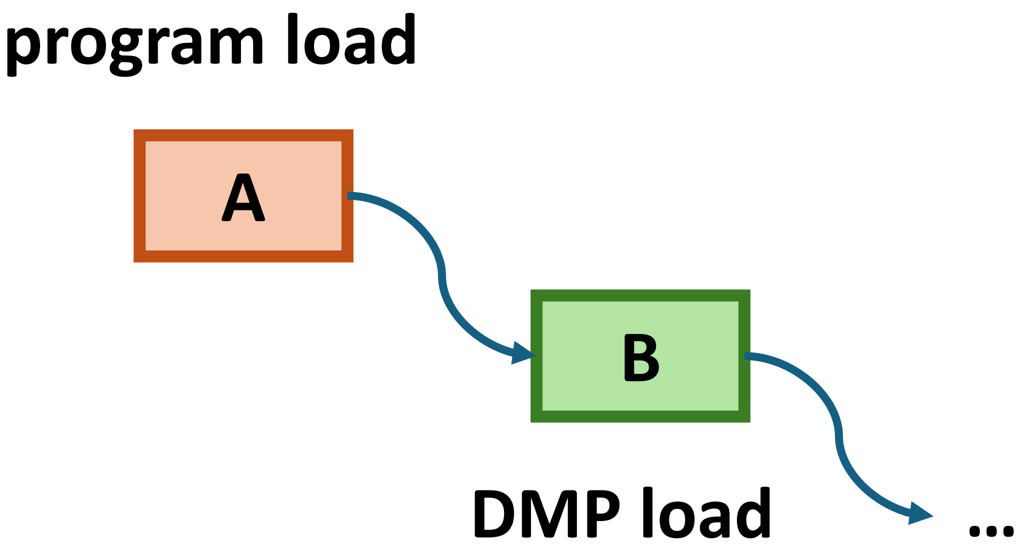


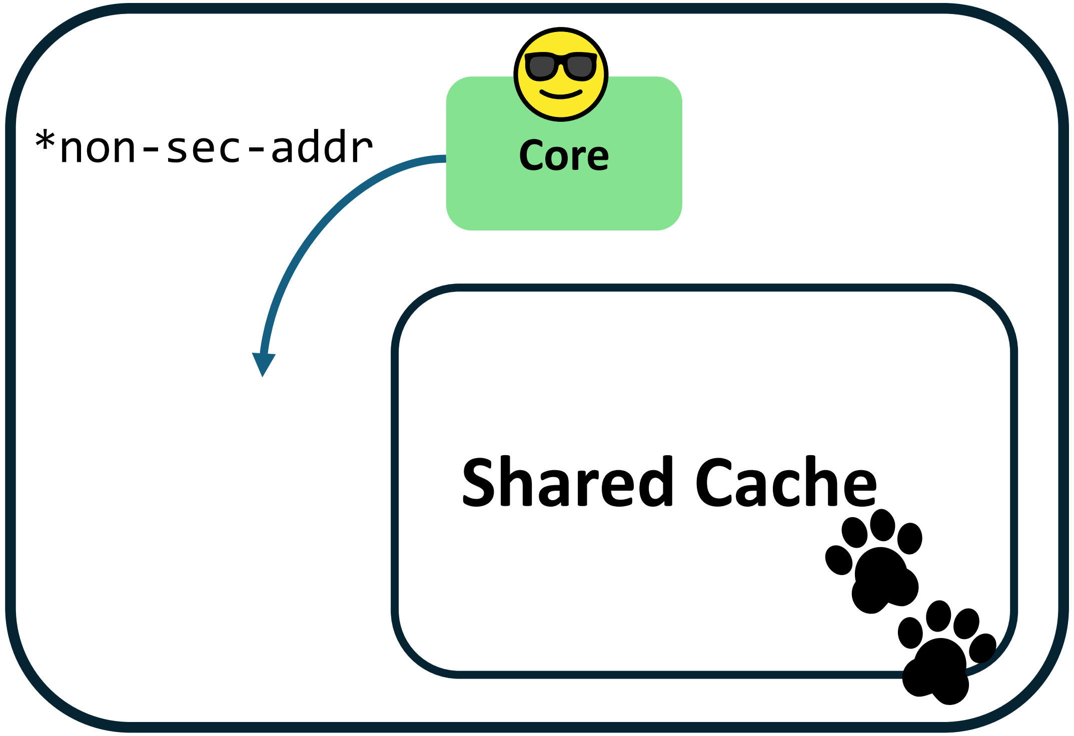


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```



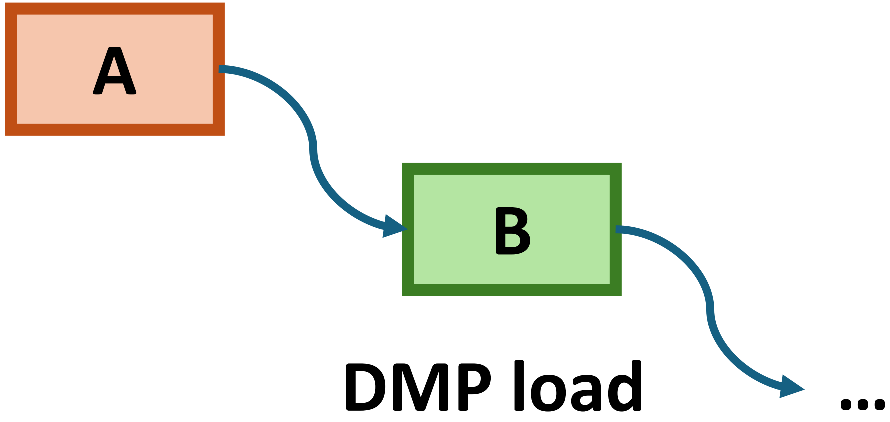


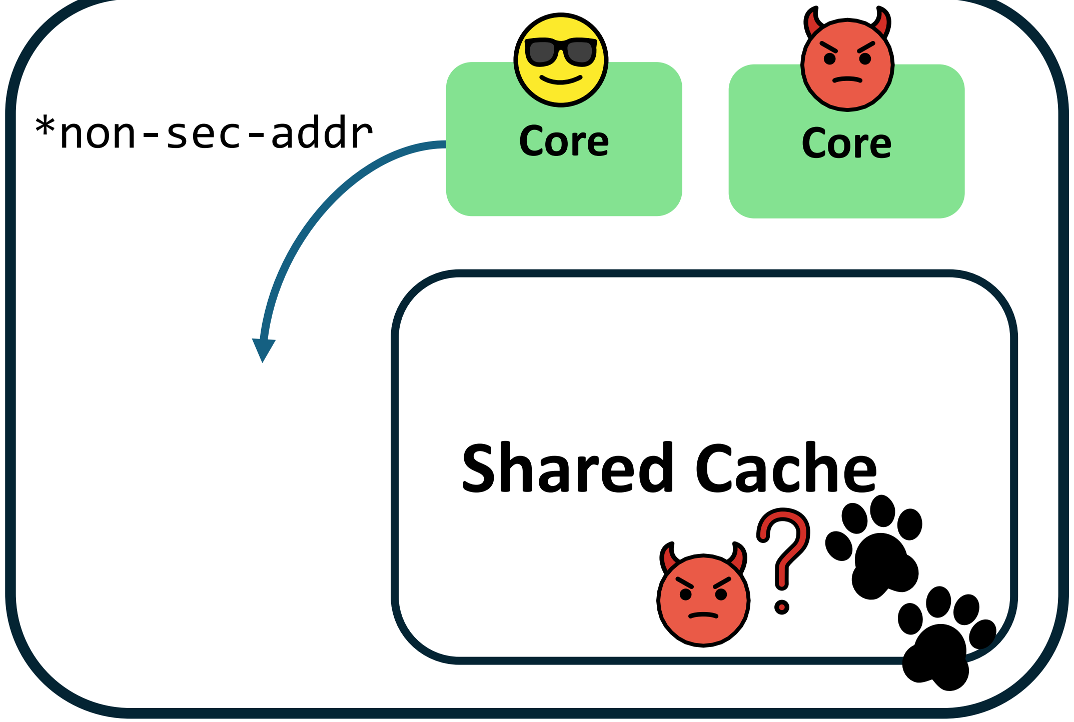
A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```

program load

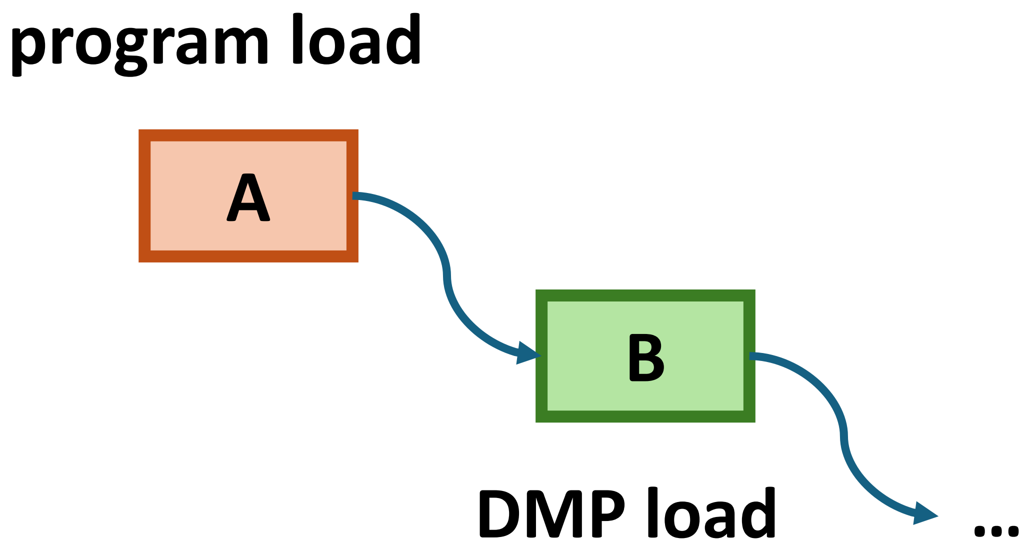


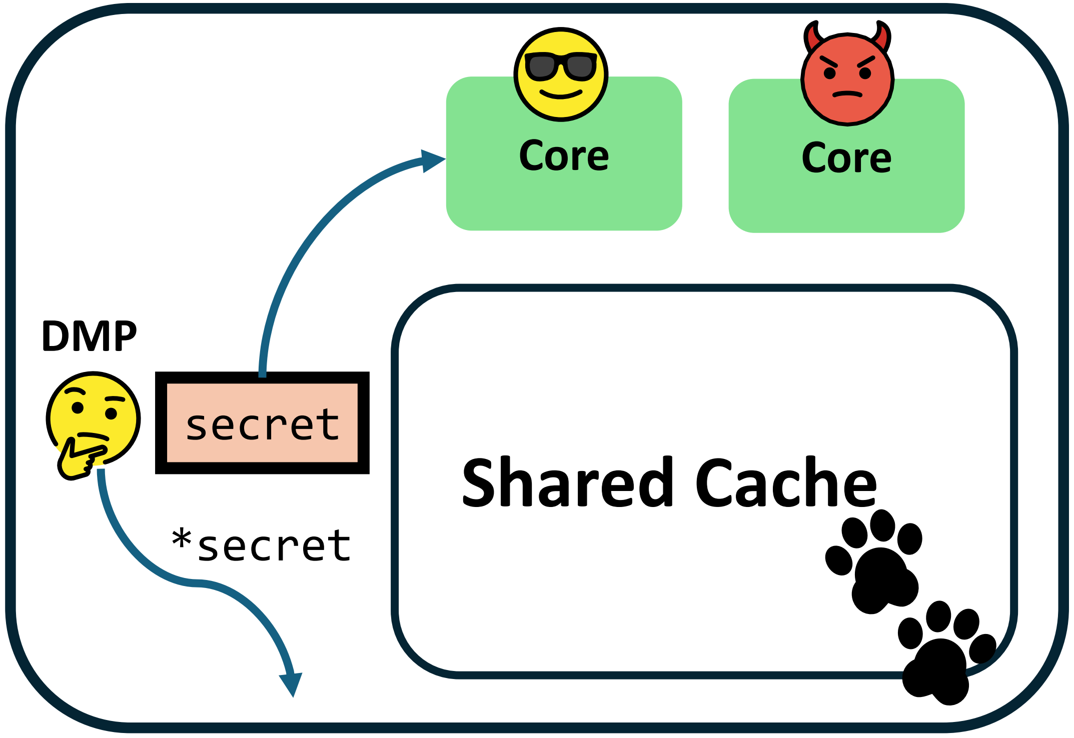


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```

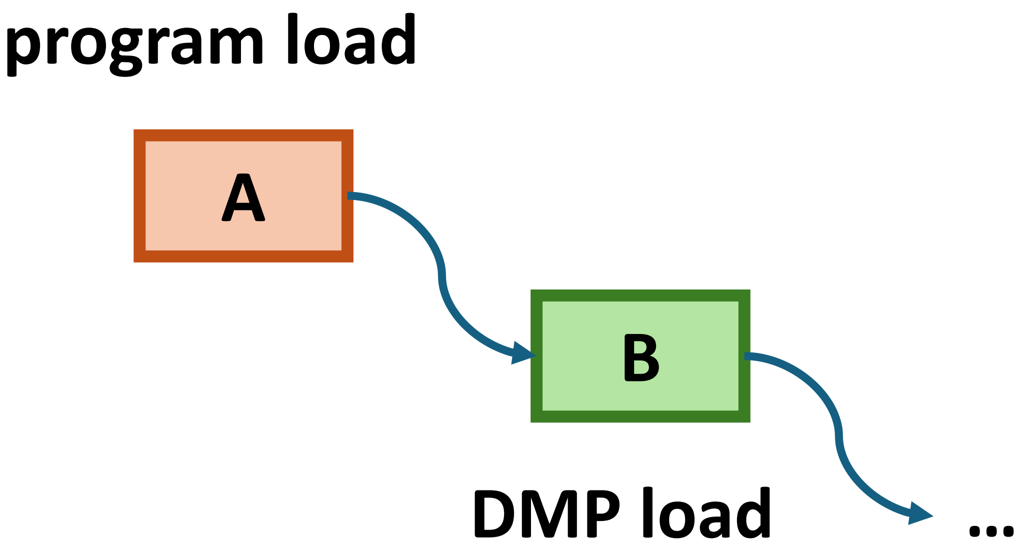


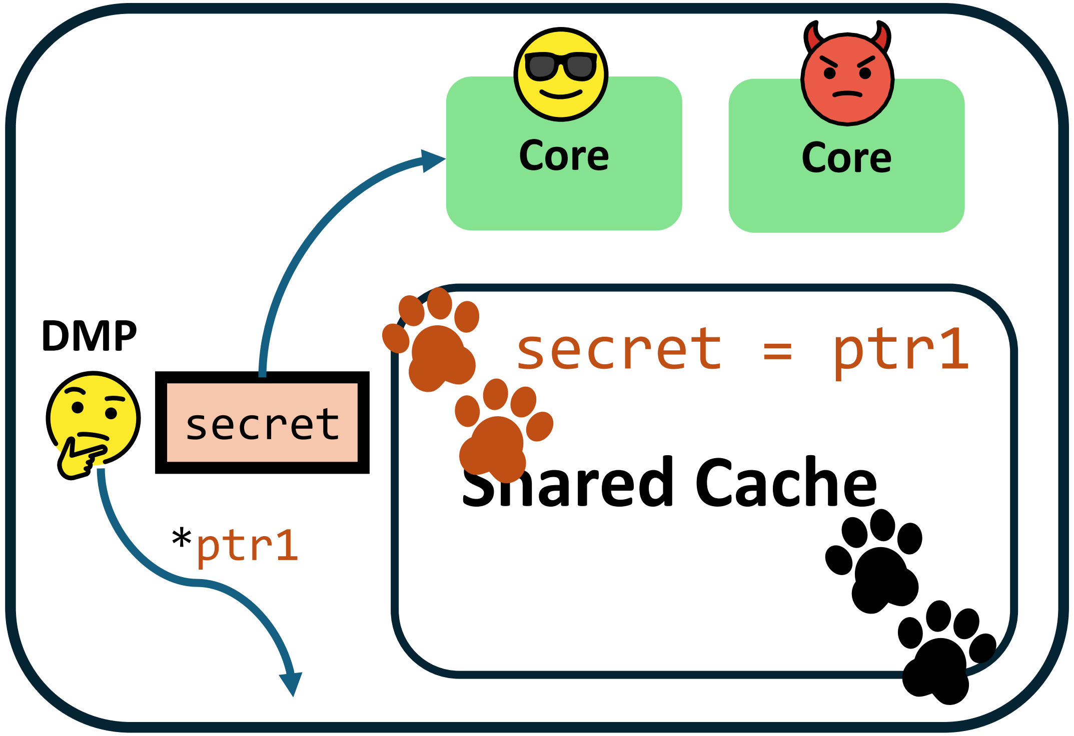


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```

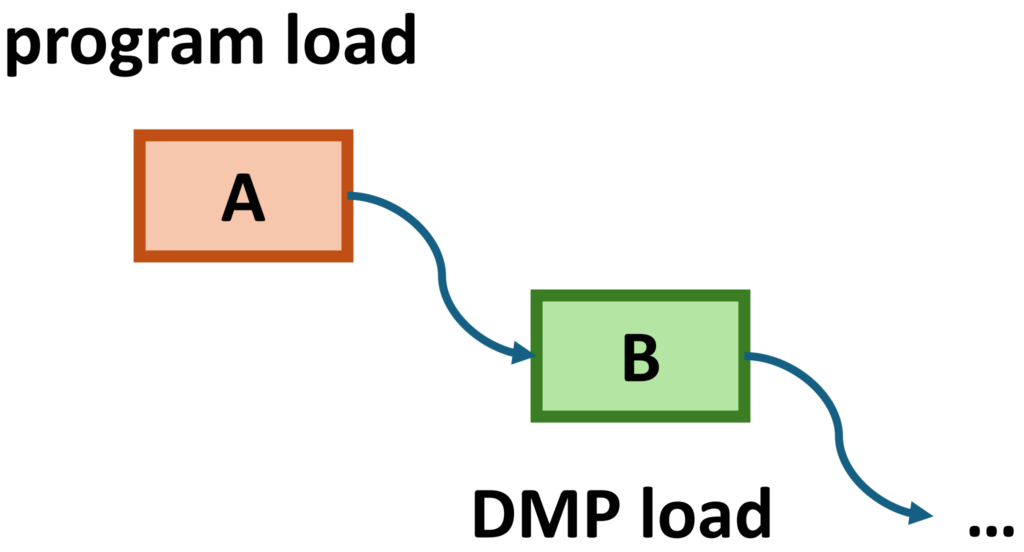


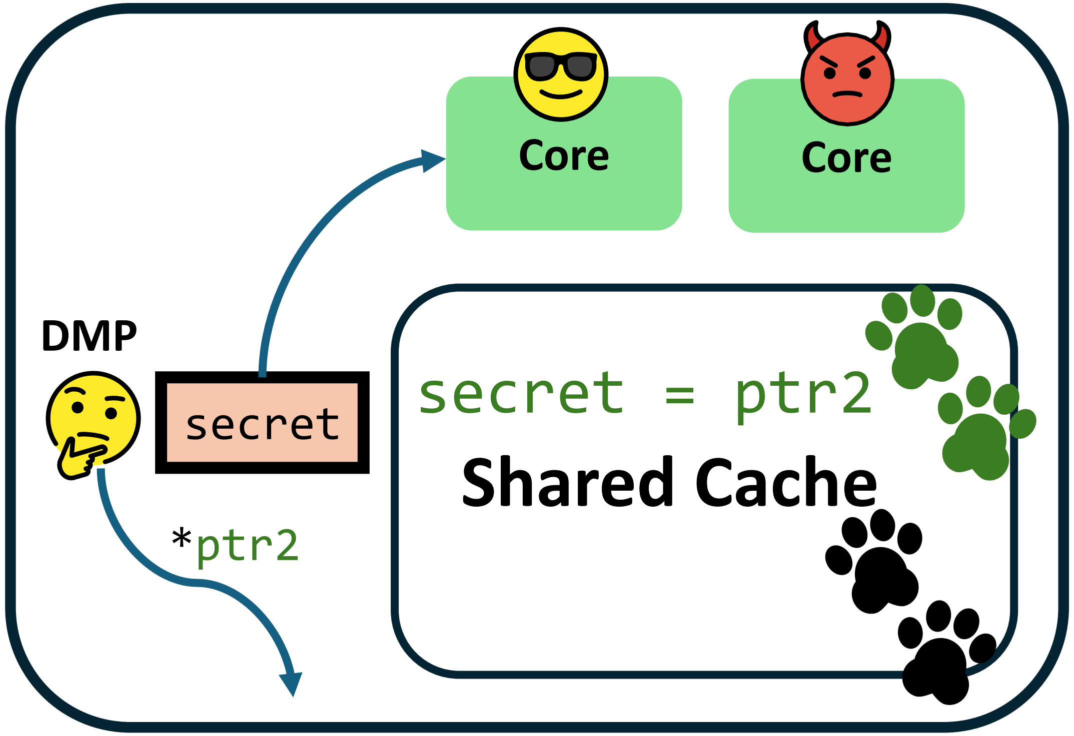


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```

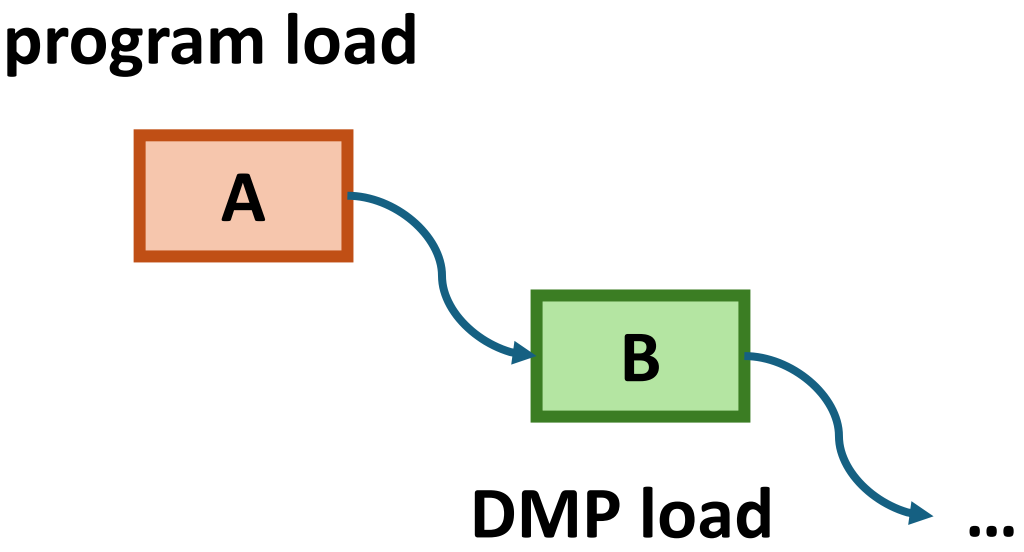


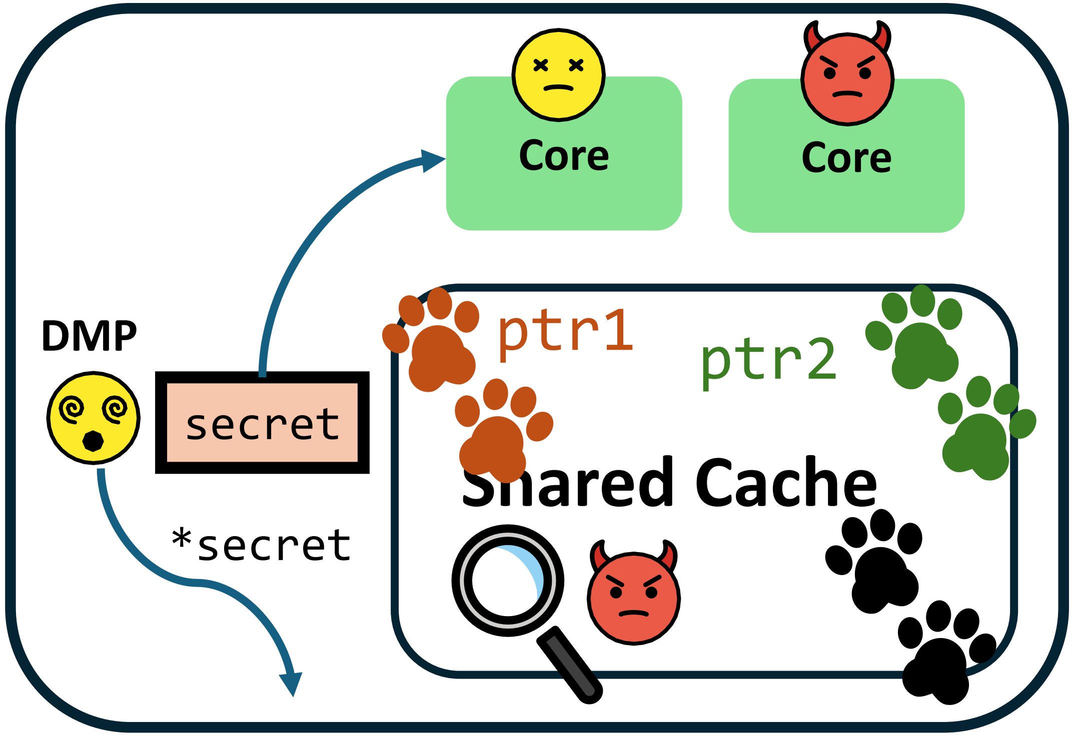


A Constant-Time Code Example

Apple DMP could **treat loaded data as memory address** and perform access.

```
// secret = ptr1 or ptr2
secret = *non-sec-addr
```





Contribution

Augury[1]

1 Vicarte, et al., "Augury: Using data memory-dependent prefetchers to leak data at rest", SP'22.

Contribution

- Comprehensive reverse engineering of Apple DMPs.

1 Vicarte, et al., "Augury: Using data memory-dependent prefetchers to leak data at rest", SP'22.

Contribution

- Comprehensive reverse engineering of Apple DMPs.
- Develop DMP-aided chosen-input attack framework.

1 Vicarte, et al., "Augury: Using data memory-dependent prefetchers to leak data at rest”, SP’22.

Contribution

- Comprehensive reverse engineering of Apple DMPs.
- Develop DMP-aided chosen-input attack framework.
- Undermine four cryptographic implementations in the wild or submitted to NIST PQC standardization.

1 Vicarte, et al., "Augury: Using data memory-dependent prefetchers to leak data at rest", SP'22.

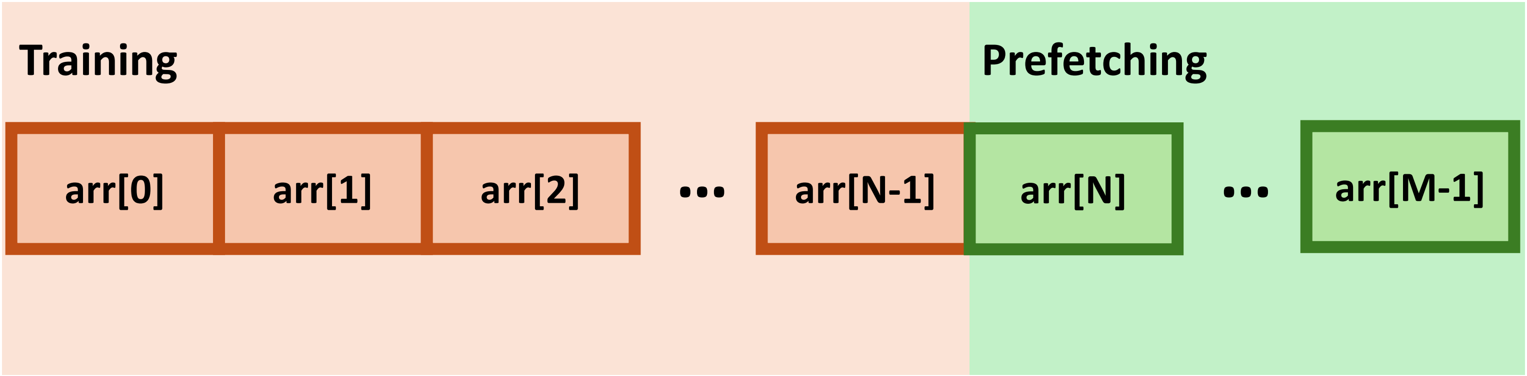
Key Observations of Apple DMP

How do classical prefetchers work?

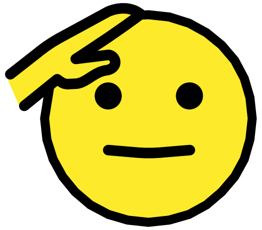
Key Observations of Apple DMP

How do classical prefetchers work?

```
// stride pattern
for (i = 0; i < M; i++)
    trash += arr[i];
```



The program reads arr[0], arr[1], …
The stride is 1! Prefetch arr[N], arr[N+1],…



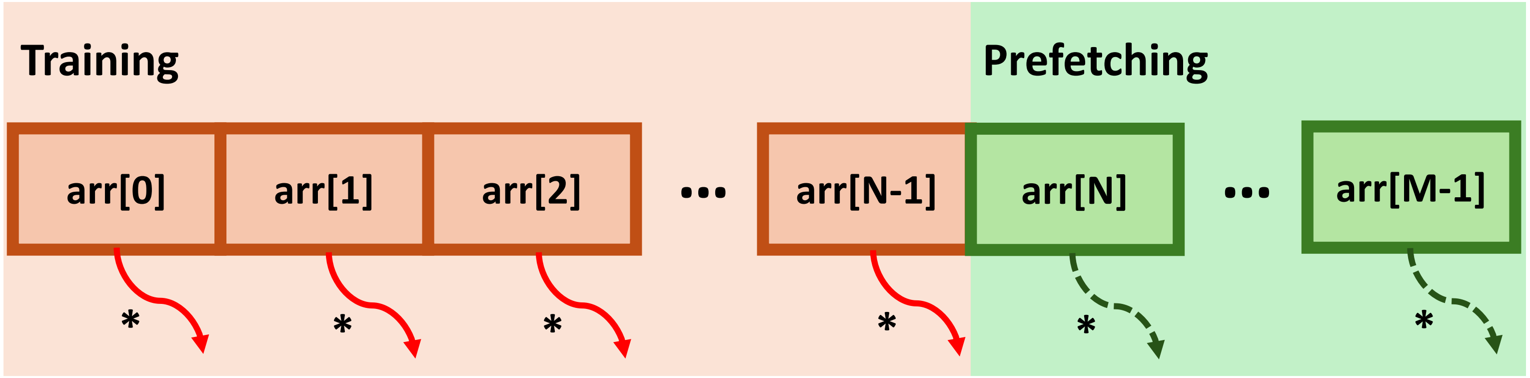
stride prefetcher

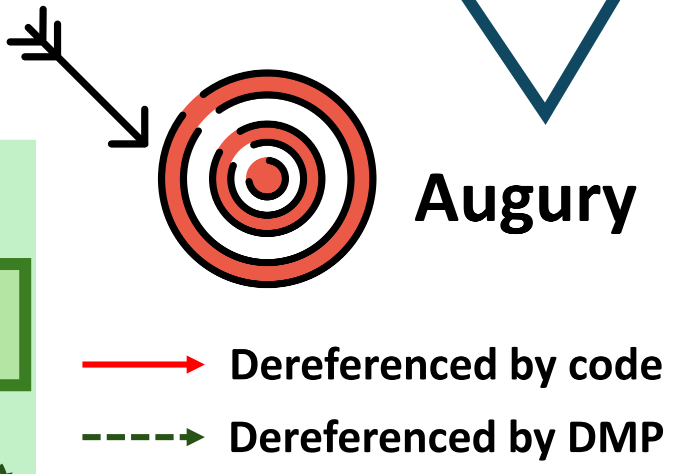
Key Observations of Apple DMP

What is the finding of prior work, Augury?

```
// Array-of-pointer pattern
for (i = 0; i < M; i++)
    trash += *arr[i];
```

DMP recognizes and prefetches Array-of-pointers access pattern!





Key Observations of Apple DMP

Does memory access pattern even matter?

```
// Array-of-pointer pattern
for (i = 0; i < M; i++)
    trash += *arr[i];
```

Really? Is it necessary?

GoFetch

Key Observations of Apple DMP

Does memory access pattern even matter?

Really? Is it necessary?

GoFetch

```
// Array-of-pointer pattern
for (i = 0; i < M; i++)
    trash += *arr[i];
```

```
// Single load
trash += arr[0];
```

Key Observations of Apple DMP

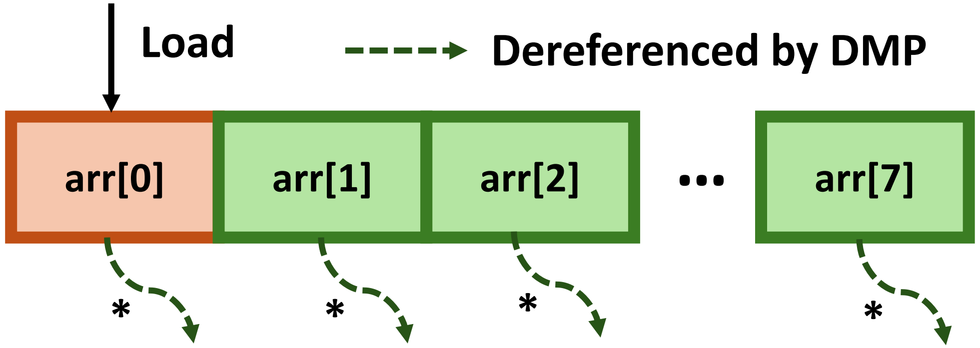
Does memory access pattern even matter?

Really? Is it necessary?

GoFetch

```
// Array-of-pointer pattern
for (i = 0; i < M; i++)
    trash += *arr[i];
```

```
// Single load
trash += arr[0];
```

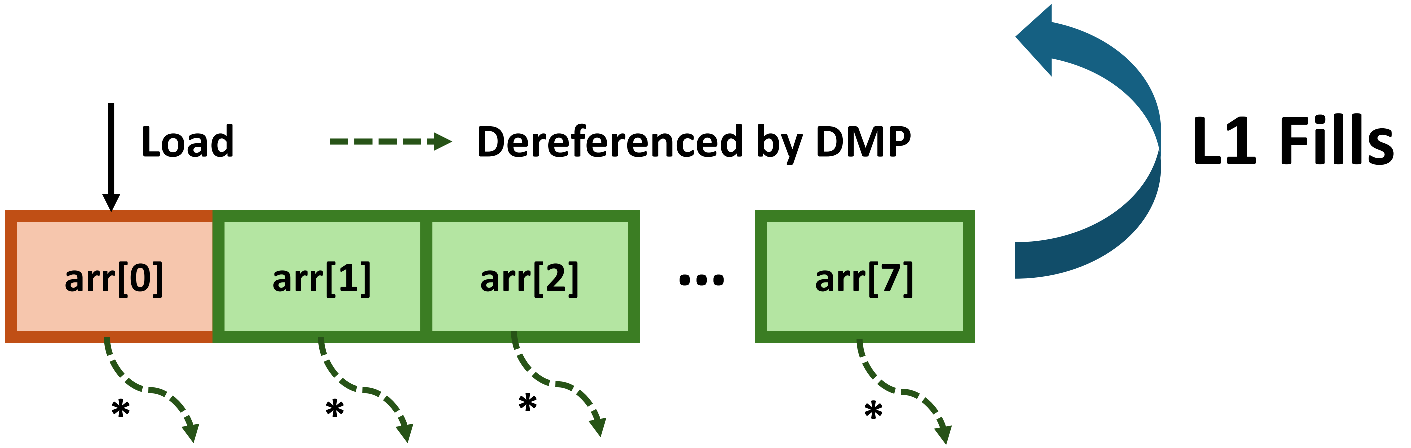


Cache Line Aligned

Key Observations of Apple DMP

Where does the DMP scan for pointers?

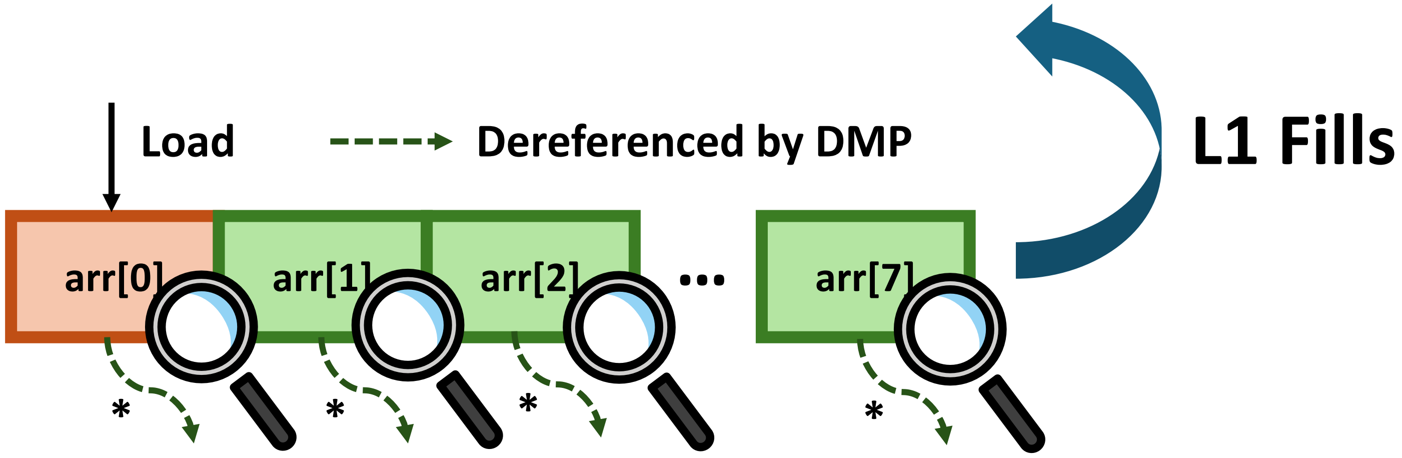
```
// Single load
trash += arr[0];
```



Cache Line Aligned

Key Observations of Apple DMP

How does DMP determine pointers to dereference in each line?



Cache Line Aligned

Key Observations of Apple DMP

History filter: ***how DMP avoids redundant dereference?***

4GByte region: ***heuristic of predicting pointer value.***

Do-not-scan hint: ***how DMP avoids redundant scan?***

Top byte ignore: ***how DMP synergizes with TBI?***

…

…

Check out the paper!

How to use DMP to break CT Crypto?

How to use DMP to break CT Crypto?

`cstate`: crypto state

`sec`: secret   `ci`: chosen input

ptr   **if sec=A**

No ptr   **if sec=B**

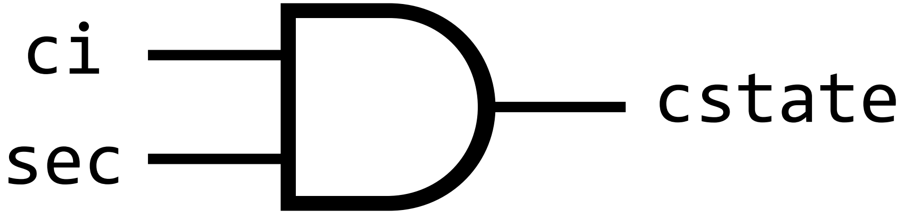
How to use DMP to break CT Crypto?

`cstate`: crypto state

`sec`: secret `ci`: chosen input

```
cstate = ci AND sec
```

```
sec { 0xffffffffffffffff
    { 0x0000000000000000
```



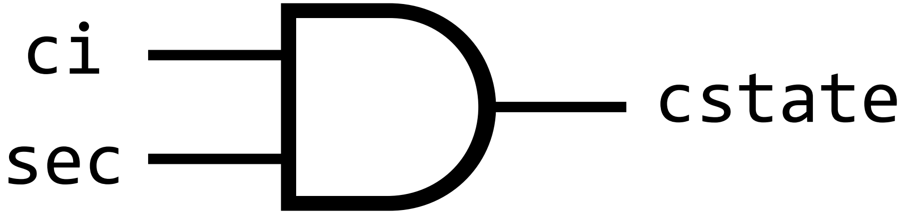
How to use DMP to break CT Crypto?

`cstate`: crypto state

`sec`: secret `ci`: chosen input

```
cstate = ci AND sec
```

```
sec { 0xffffffffffffffff
      0x0000000000000000
```



Choose `ci` as valid pointer!

```
cstate = ptr AND sec
if sec = 0xffffffffffffffff
   => cstate = ptr

if sec = 0x0000000000000000
   => cstate = 0
```

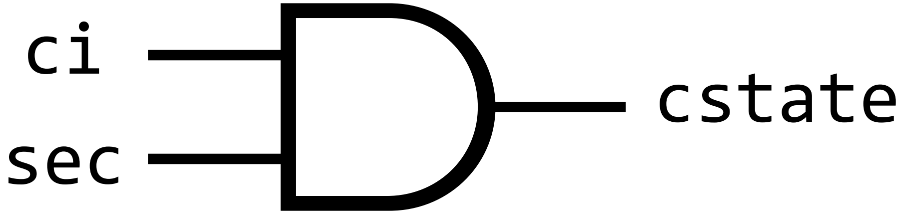
How to use DMP to break CT Crypto?

`cstate`: crypto state

`sec`: secret   `ci`: chosen input

```
cstate = ci AND sec
```

```
sec { 0xffffffffffffffff
      0x0000000000000000
```



Choose `ci` as valid pointer!

```
cstate = ptr AND sec
if sec = 0xffffffffffffffff
    => cstate = ptr
```

DMP → *ptr

```
if sec = 0x0000000000000000
    => cstate = 0
```

DMP

Proof-of-Concept Attacks

Cryptanalysis for DMP exploit + **End-to-end key extraction PoCs**

Proof-of-Concept Attacks

Cryptanalysis for DMP exploit ➕ **End-to-end key extraction PoCs**

OpenSSL DHKE

Proof-of-Concept Attacks

Cryptanalysis for DMP exploit + **End-to-end key extraction PoCs**

OpenSSL DHKE

Go RSA

Proof-of-Concept Attacks

Cryptanalysis for DMP exploit ➕ **End-to-end key extraction PoCs**

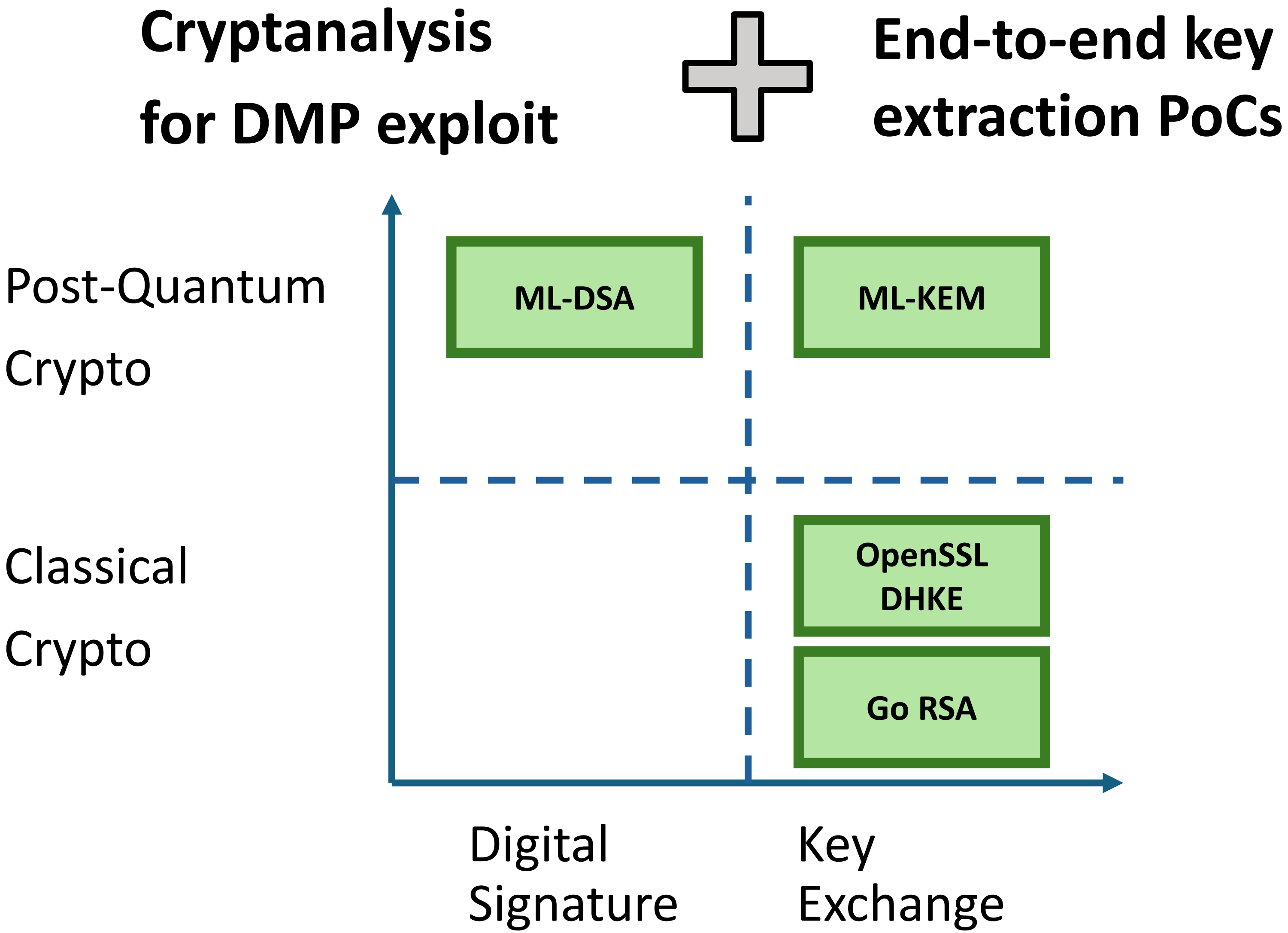
ML-DSA

ML-KEM

OpenSSL DHKE

Go RSA

Proof-of-Concept Attacks



Impact

Apple: Disable DMP with DIT=1

- Only works on M3.

```
bool set_DIT_enabled(void) {
    bool was_DIT_enabled = get_DIT_enabled();
    __asm__ __volatile__("msr dit, #1");
    return was_DIT_enabled;
}
```

Enable DIT for constant-time cryptographic operations

Impact

Apple: Disable DMP with DIT=1

- Only works on M3.

Go: Propose an opt-in DIT mode in Go binary.

proposal: runtime: implement a DIT/DOIT mode #66450

Open rolandshoemaker opened this issue on Mar 21 · 20 comments

Impact

Apple: Disable DMP with DIT=1

- Only works on M3.

Go: Propose an opt-in DIT mode in Go binary.

Asahi Linux

Asahi Linux: Found chicken bit to disable DMP on M1/M2.

Hector Martin
@marcan@treehouse.systems

Found the DMP disable chicken bit. it's `HID11_EL1<30>` (at least on M2).

So yeah, as I predicted, GoFetch is entirely patchable. I'll write up a patch for Linux to hook it up as a CPU security bug workaround.

(`HID4_EL1<4>` also works, but we have a name for that and it looks like a big hammer: `HID4_FORCE_CPU_OLDEST_IN_ORDER`)

Code here: github.com/AsahiLinux/m1n1/blo... (Thanks to @dkohlbre for the userspace C version this is based off of!)

One interesting finding is that the DMP is *already disabled* in EL2 (and presumably EL1), it only works in EL0. So it looks like the CPU designers already had *some* idea that it is a security liability, and chose to hard-disable it in kernel mode. This means kernel-mode crypto on Linux is already intrinsically safe.

Apr 08, 2024, 20:17 · Edited Apr 08, 20:24 · Web · 97 · 215

Impact

Apple: Disable DMP with DIT=1

- Only works on M3.

Go: Propose an opt-in DIT mode in Go binary.

Asahi Linux

Asahi Linux: Found chicken bit to disable DMP on M1/M2.

Pwine Awards: Best Cryptographic Attack winner.

Conclusion

- Data memory-dependent prefetchers (DMPs) performs secret-dependent memory access to leak data.
- Exploiting DMPs to perform key extraction attacks to constant-time cryptography is feasible.

GoFetch

Check our Website:

gofetch.fail

boruc2@illinois.edu