RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

SESSION ID: CRYP-T07

BREAKING ED25519 IN WOLFSSL

Niels Samwel[1], Lejla Batina[1], Guido Bertoni[2], Joan Daemen[1,2] and Ruggero Susella[2]

Radboud University[1]
STMicroelectronics[2]

Introduction

- ECDSA is a signature scheme that uses random ephemeral scalars
- Weak keys can be attacked with lattice attacks
- Ed25519 is a deterministic signature scheme that prevents weak keys
- When an adversary has access to a device this is becomes a weakness

Contributions

Contributions

- Side-channel attack on Ed25519.
- Side-channel attack on the message schedule of SHA512.
- Countermeasure for this attack.

The Attack Components

Three components

- Attacking Ed25519 to recover long term secret
- Attack on SHA512
- DPA on modular addition

Algorithm 1 Ed25519 key setup and signature generation

Key setup.

Hash k such that $H(k) = (h_0, h_1, \ldots, h_{2b-1}) = (a, b)$

$a = (h_0, \ldots, h_{b-1})$, Private scalar

$b = (h_b, \ldots, h_{2b-1})$, Auxiliary key

Compute public key: $A = aB$.

Signature generation.

Compute ephemeral private key: $r = H(b, M)$.

Compute ephemeral public key: $R = rB$.

Compute $h = H(R, A, M)$ and convert to integer.

Compute: $S = (r + ha) \mod l$.

Signature pair: (R, S).

Attacking Ed25519

Using auxiliary key b that was successfully recovered:

1. Compute $r = H(b, M)$.
2. Compute $h = H(R, A, M)$.
3. Compute $a = (S - r)h^{-1} \mod l$.

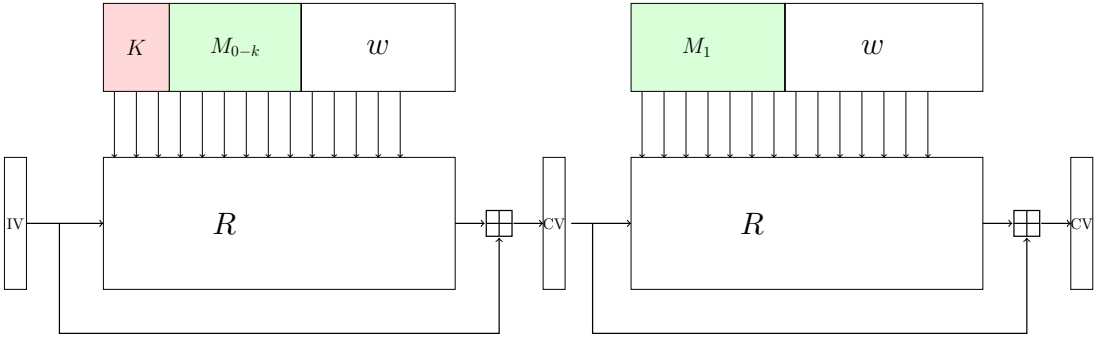
Attack on SHA512

Algorithm 2 Merkle Damgård

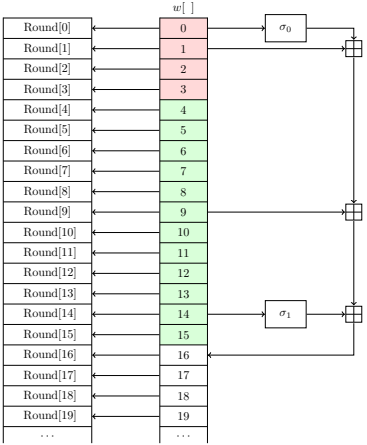
Input: Message M with $0 \leq$ bit-length $< 2^{128}$
Output: Hash value of M

1: Pad message M by appending an encoding of the message length
2: Initialize chaining value CV with constant IV
3: Split padded message M into blocks
4: **for all** blocks M_i **do**
5: $\quad CV_{i+1} \leftarrow CF(CV_i, M_i)$
6: **end for**
7: **return** H $\leftarrow$ CV

Attack on SHA512



Attack on SHA512

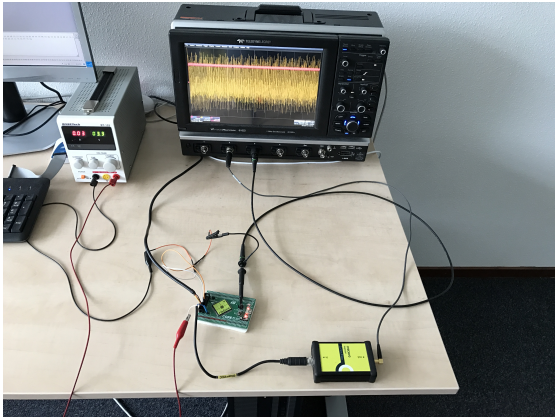


DPA on modular addition

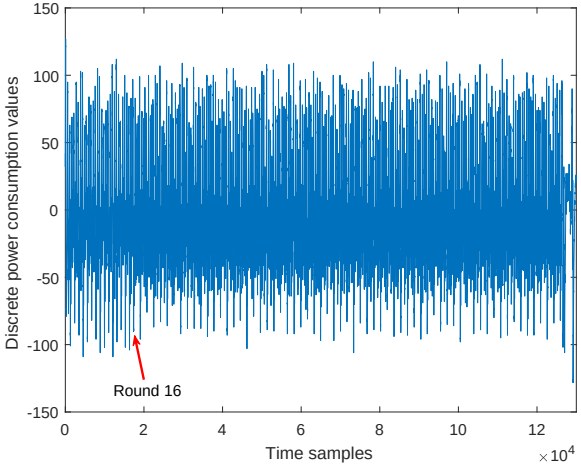
$$w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0]$$
$$w[17] \leftarrow \sigma_1(w[15]) + w[10] + \sigma_0(w[2]) + w[1]$$
$$w[18] \leftarrow \sigma_1(w[16]) + w[11] + \sigma_0(w[3]) + w[2]$$
$$w[19] \leftarrow \sigma_1(w[17]) + w[12] + \sigma_0(w[4]) + w[3]$$

DPA on modular addition

Setup

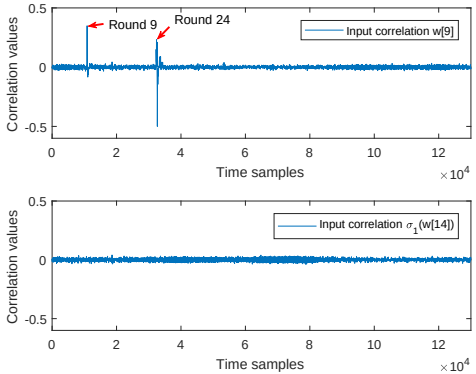


DPA on modular addition

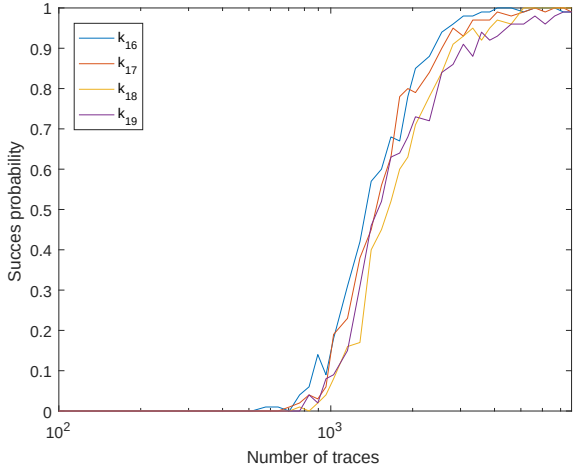


DPA on modular addition

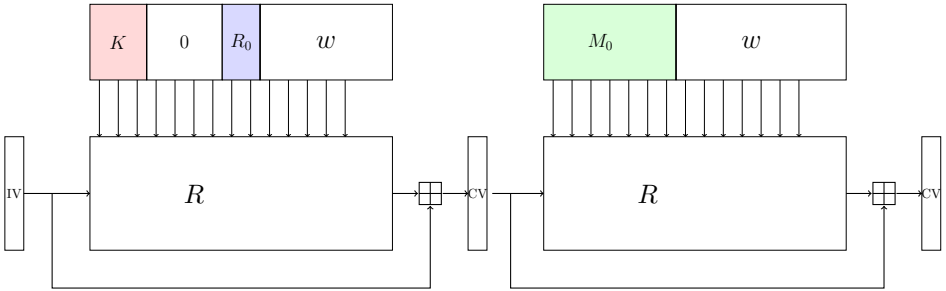
$$w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0]$$



DPA on modular addition



Countermeasure

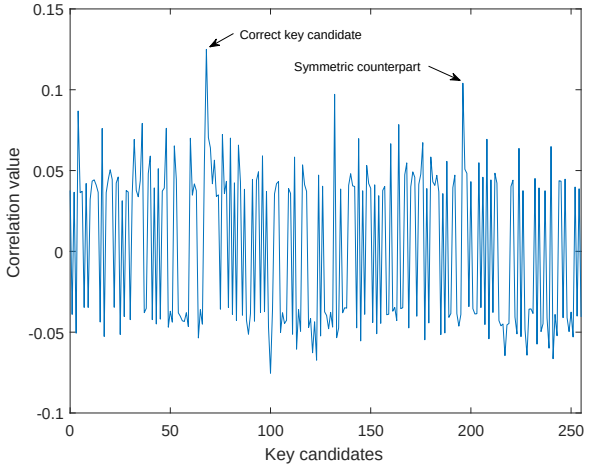


Conclusion

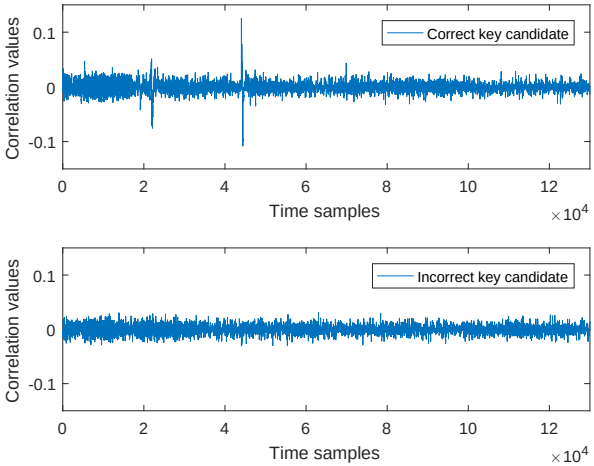
Side-channel attack on Ed25519

- High success rate
- Inexpensive countermeasure

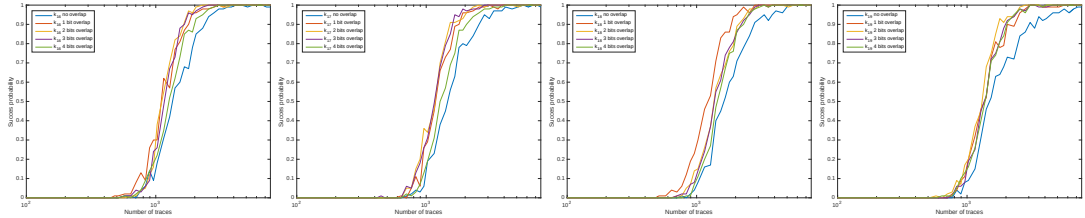
DPA on modular addition



DPA on modular addition



DPA on modular addition



RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

SESSION ID: CRYP-T07

MEMJAM: A FALSE DEPENDENCY ATTACK AGAINST CONSTANT-TIME CRYPTO IMPLEMENTATIONS IN SGX

Daniel Moghimi

Ph.D. Student
Worcester Polytechnic Institute
@danielmgmi

MemJam: A False Dependency Attack against Constant-Time Crypto Implementations in SGX

Ahmad "Daniel" Moghimi
Thomas Eisenbarth
Berk Sunar

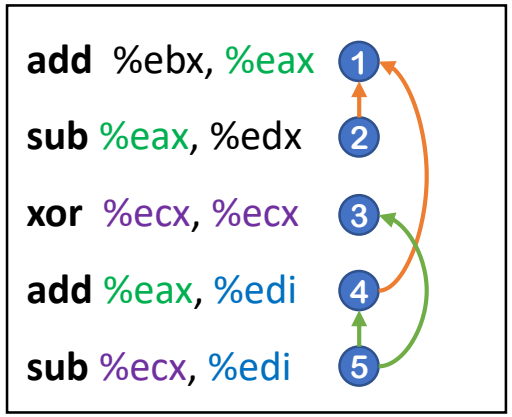
April 17, 2018

CT-RSA 2018 - San Francisco, CA

WPI

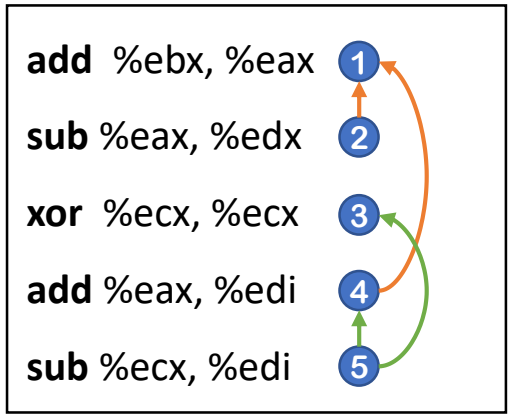
Data Dependency

- Data dependency: Instruction → Data of a preceding instruction



Data Dependency – Pipelined Execution

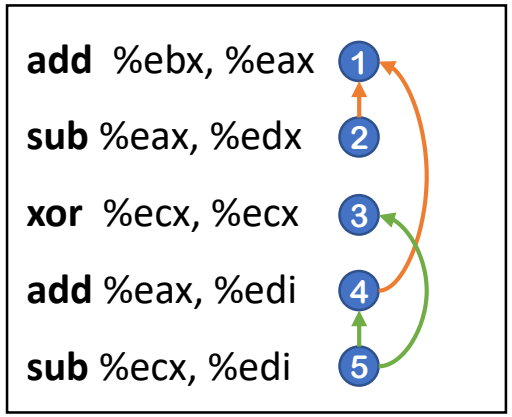
- Data dependency: Instruction → Data of a preceding instruction



IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

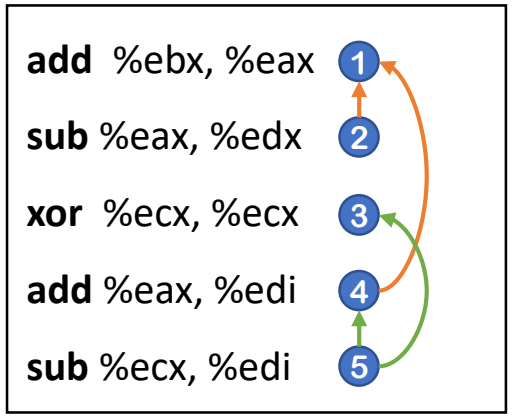


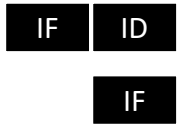
IF

IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

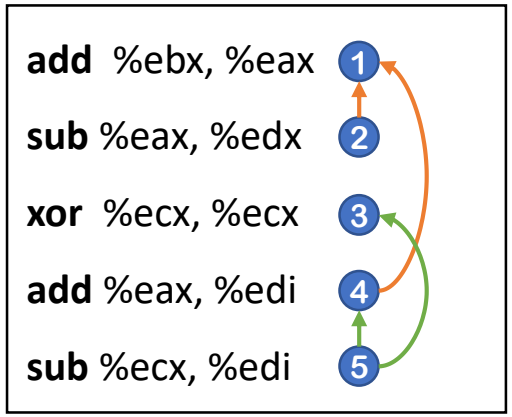


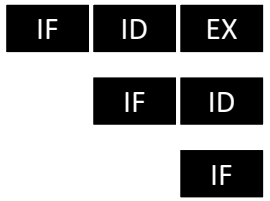


IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

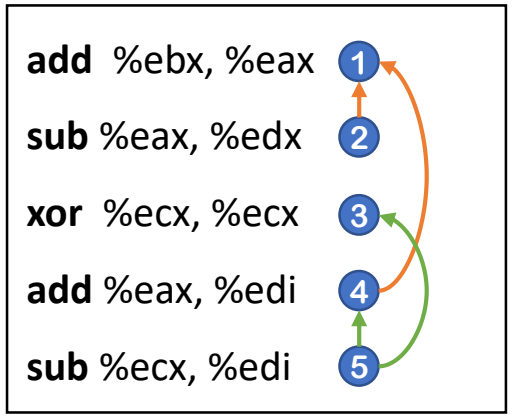


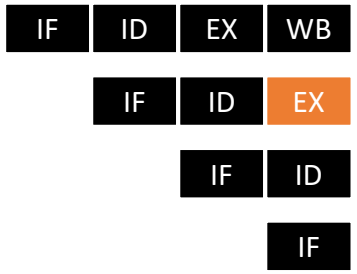


IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

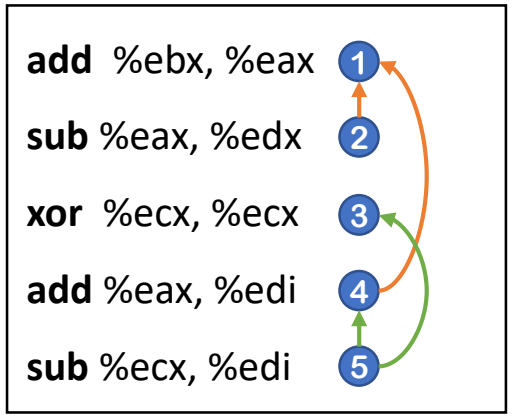


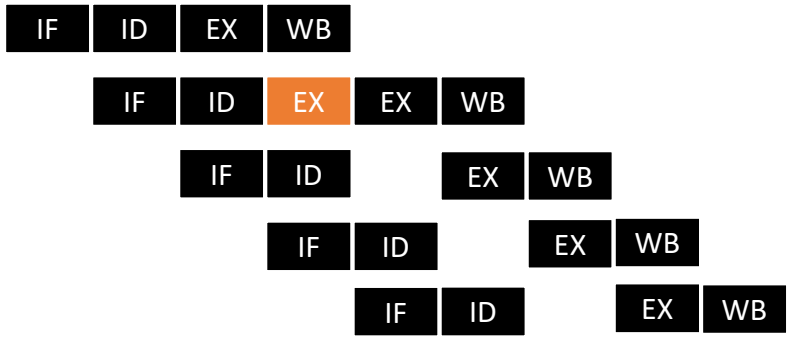


IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

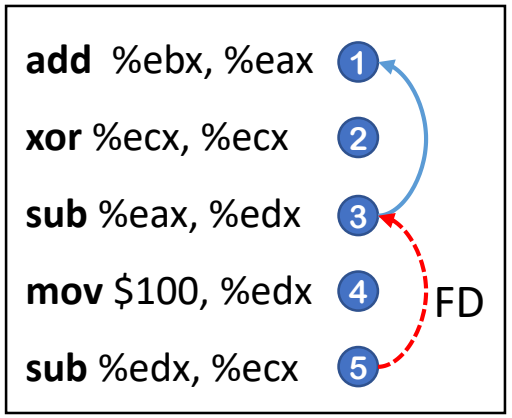




IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

~~Data~~ False Dependency

- Pipeline stalls without true dependency.
- Reasons:
  - Register Reuse
  - Limited Address Space

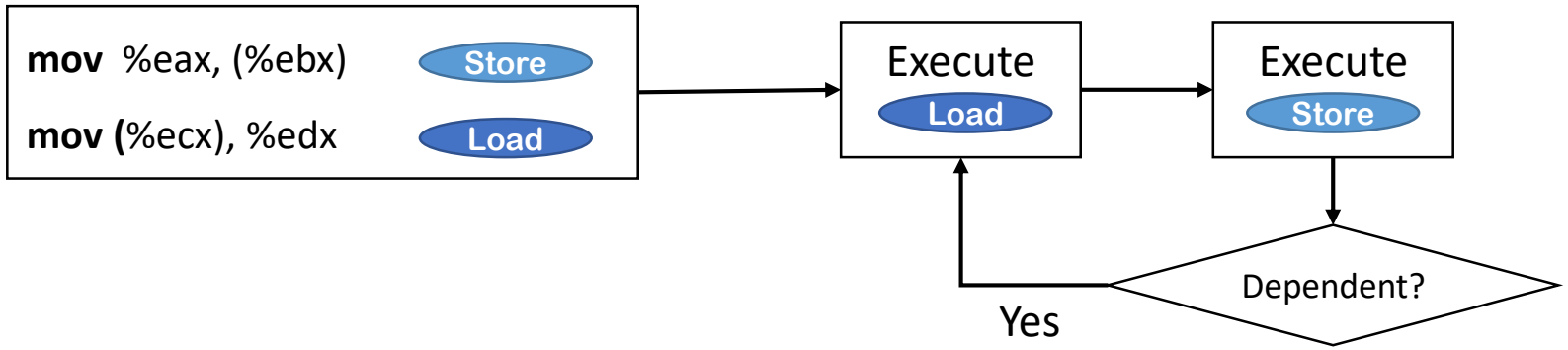


~~Data~~ False Dependency – Register Renaming

- Pipeline stalls without true dependency.
- Reasons:
  - Register Reuse
  - Limited Address Space

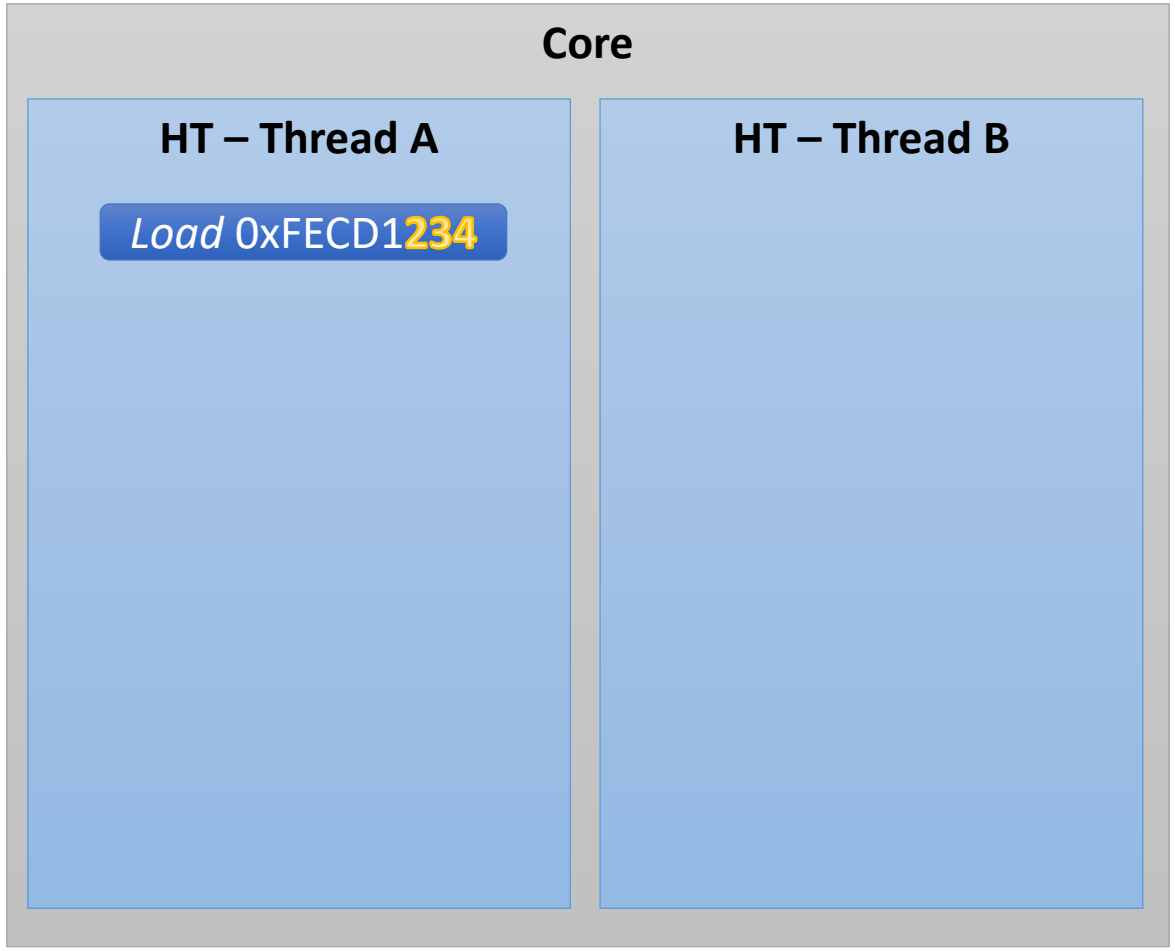
Memory False Dependency – 4K Aliasing

- Memory loads/stores are executed out of order and speculatively.
- The dependency is verified after the execution!

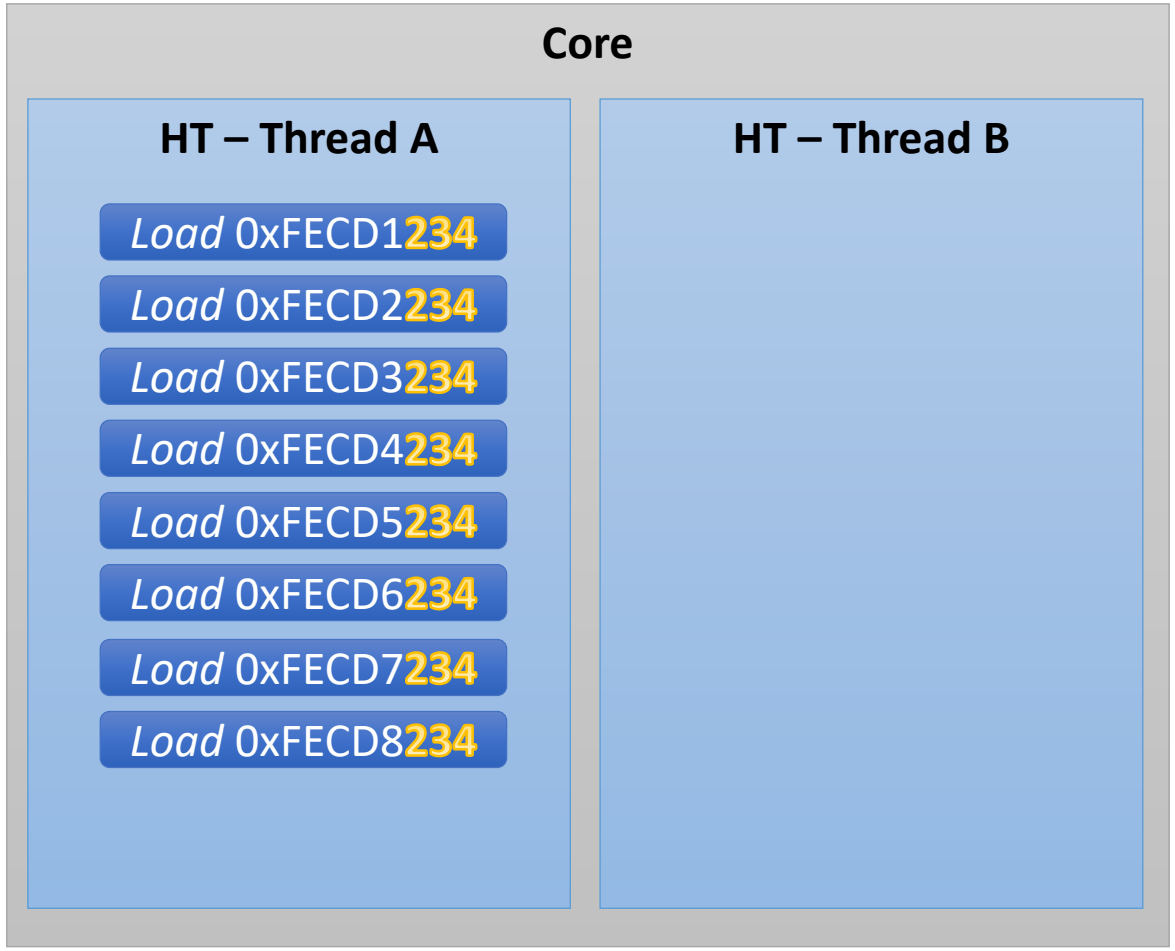


- 4K Aliasing: Addresses that are 4K apart are assumed dependent.
- Re-execute the **load** and corresponding instructions due to false dependency.
- Virtual-to-physical address translation → Memory disambiguation

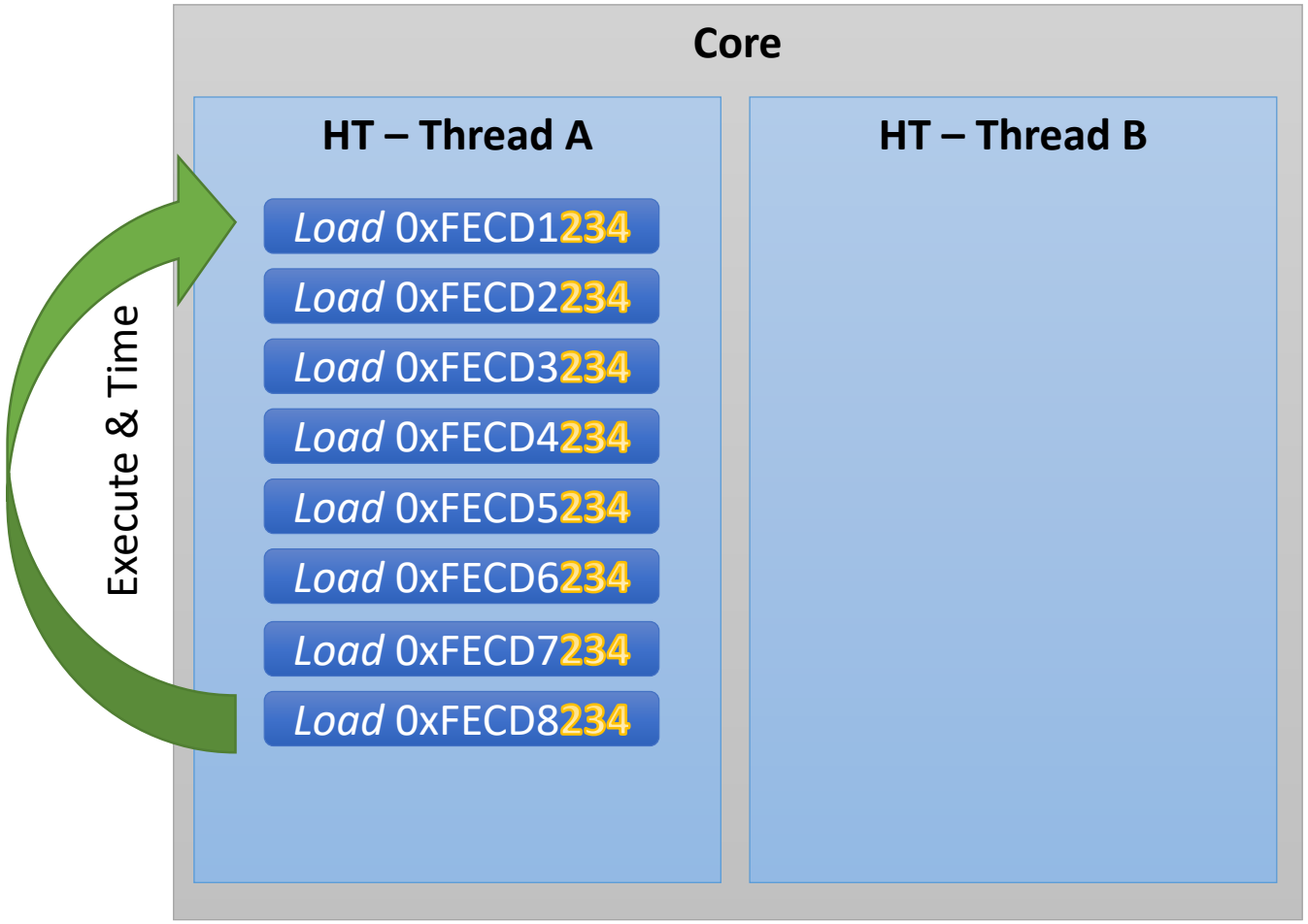
4K Aliasing – Hyperthreading



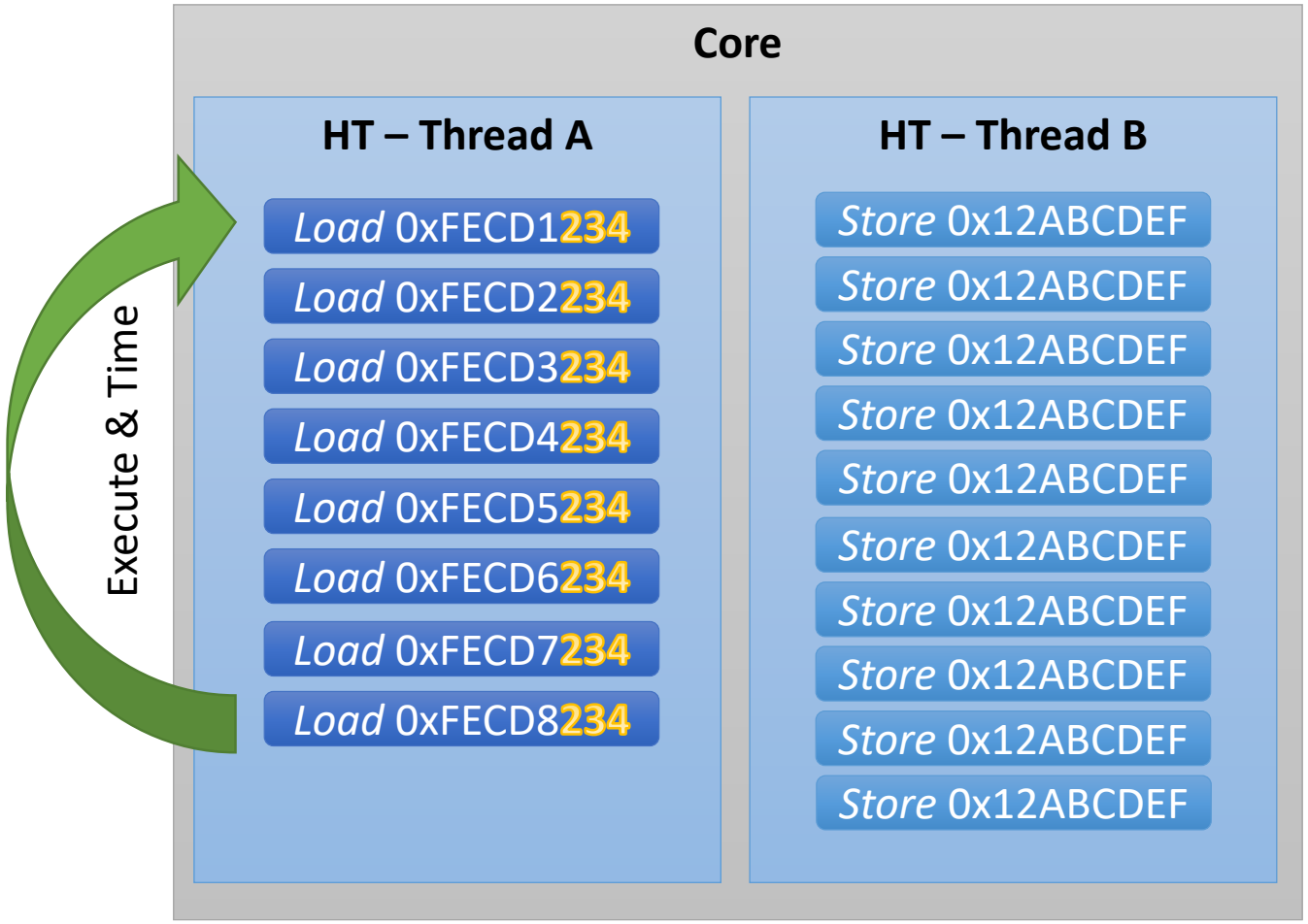
4K Aliasing – Hyperthreading

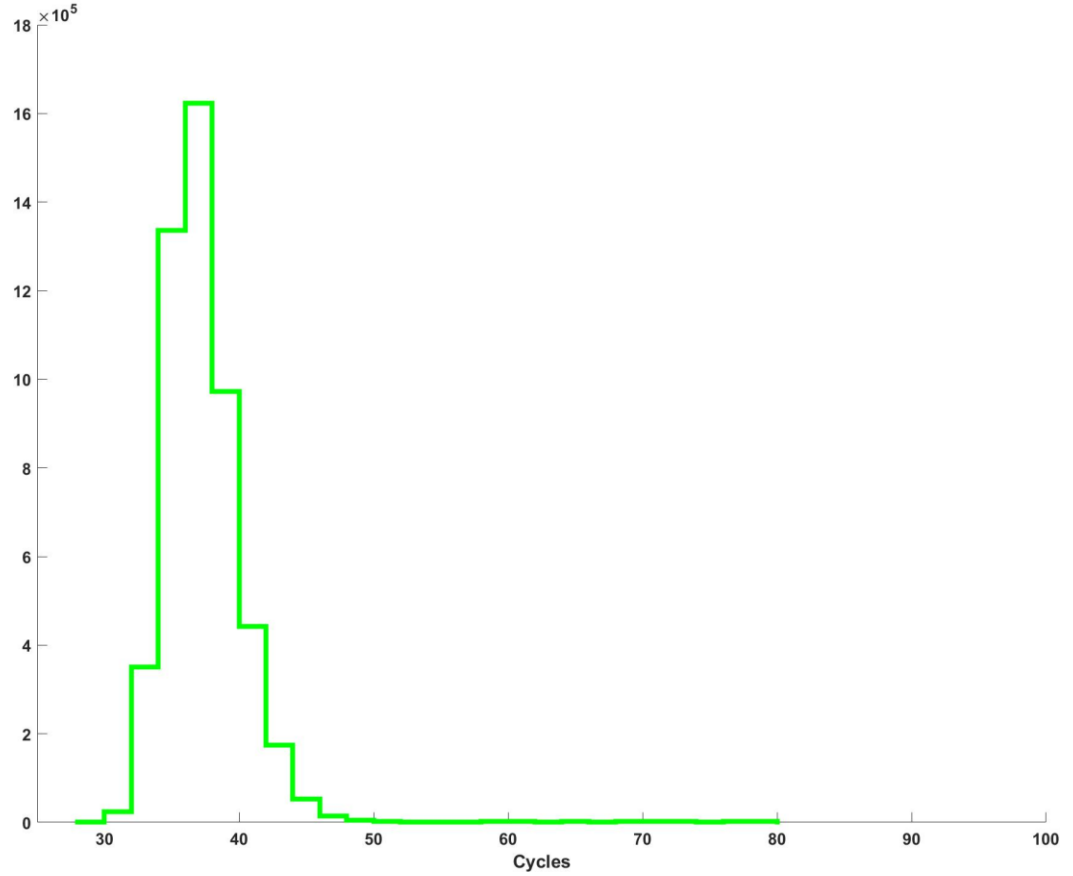


4K Aliasing – Hyperthreading

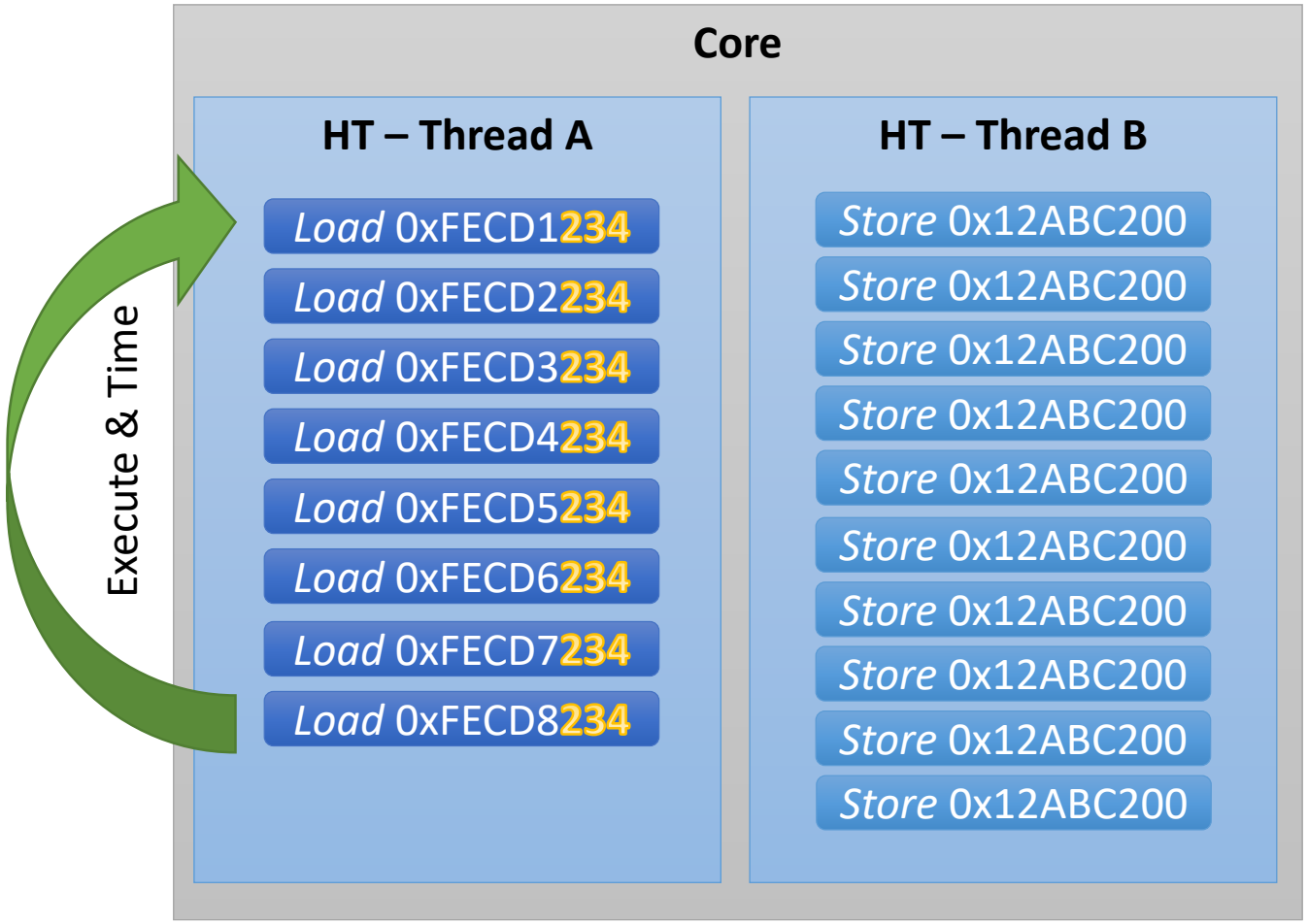


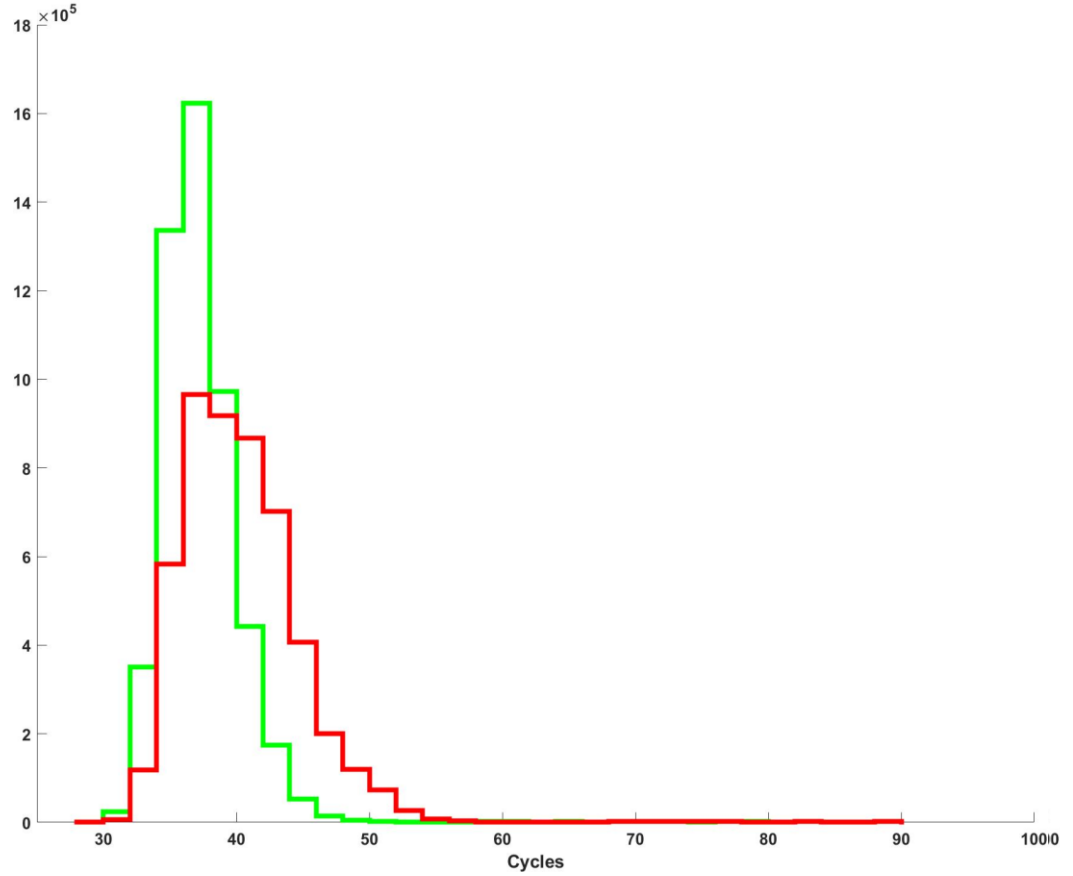
4K Aliasing – Hyperthreading



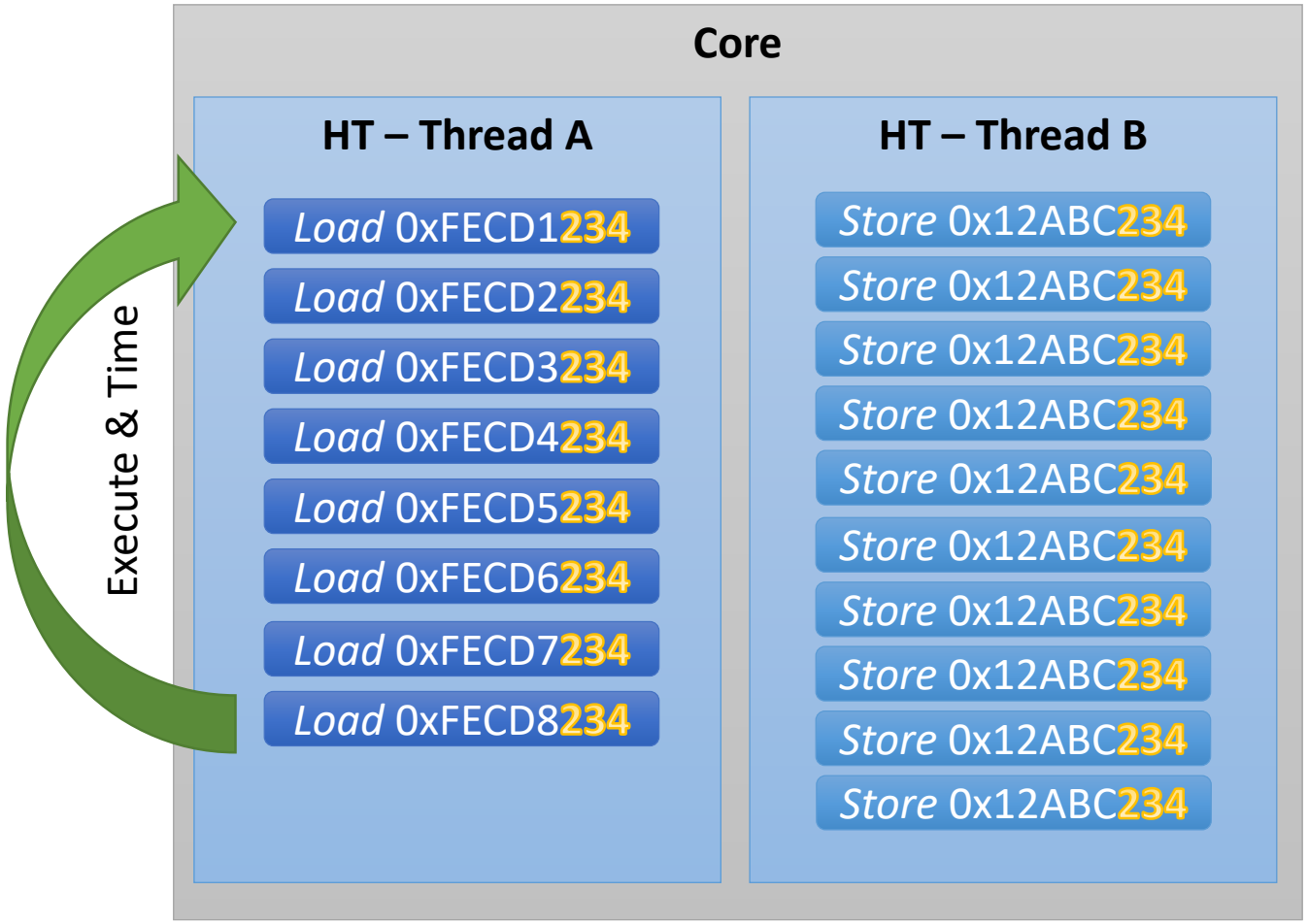


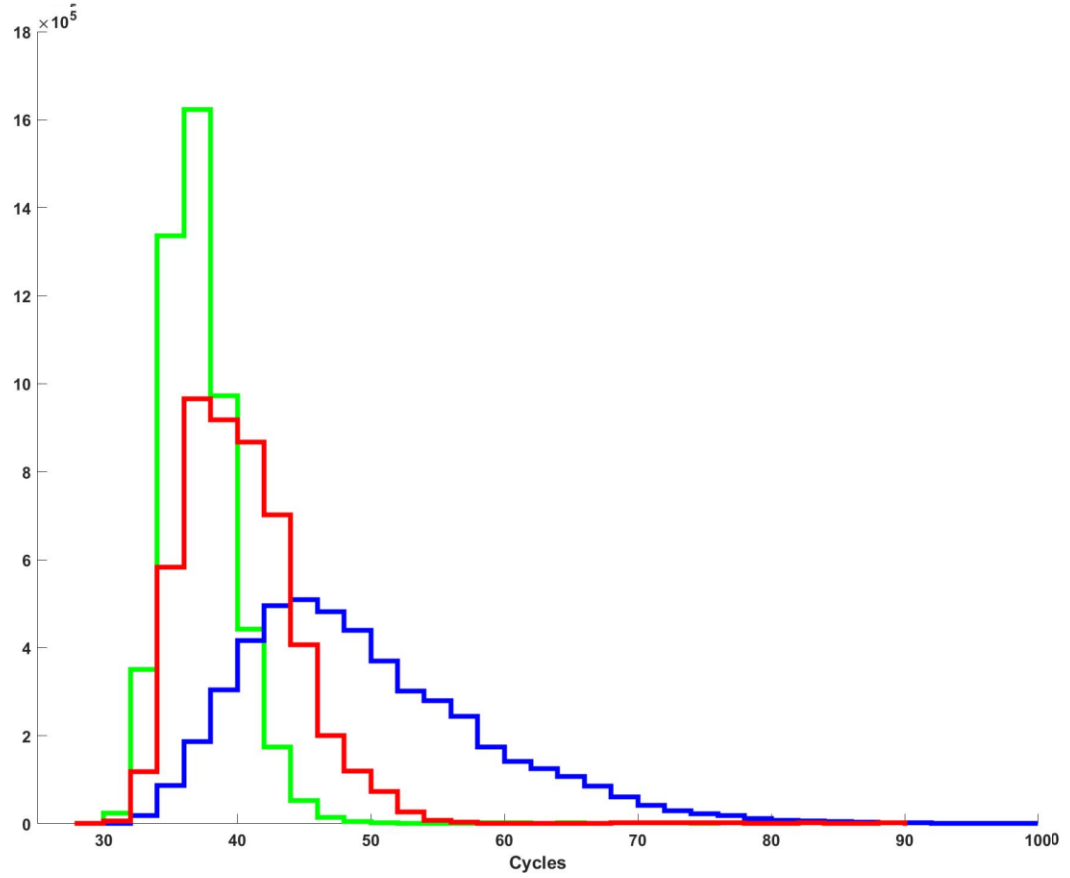
4K Aliasing – Hyperthreading





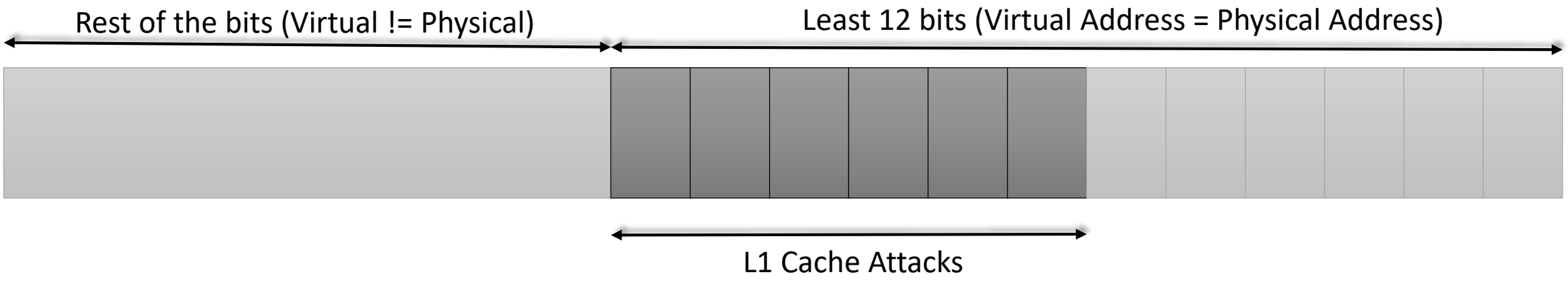
4K Aliasing – Hyperthreading



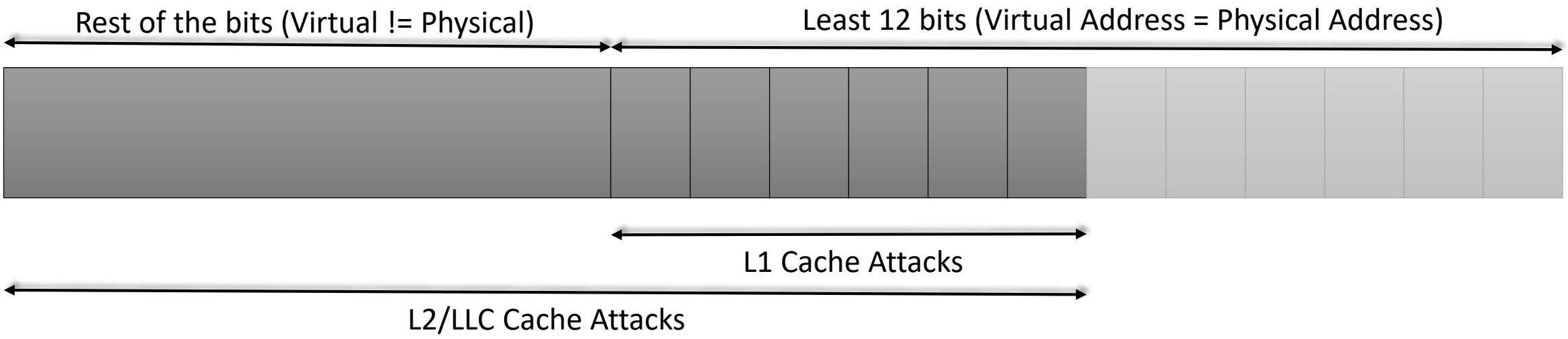


MemJam – Intra Cache Line Resolution

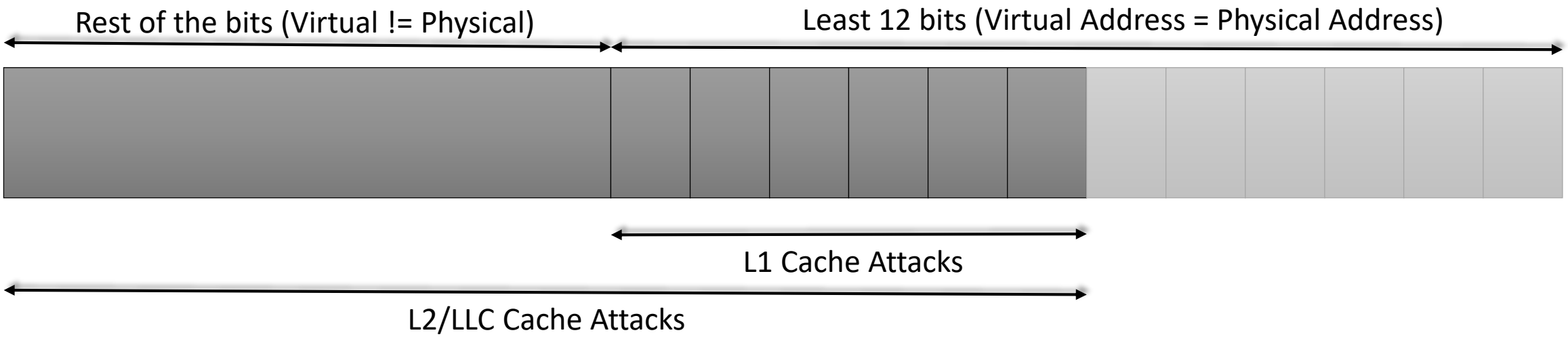
MemJam – Intra Cache Line Resolution



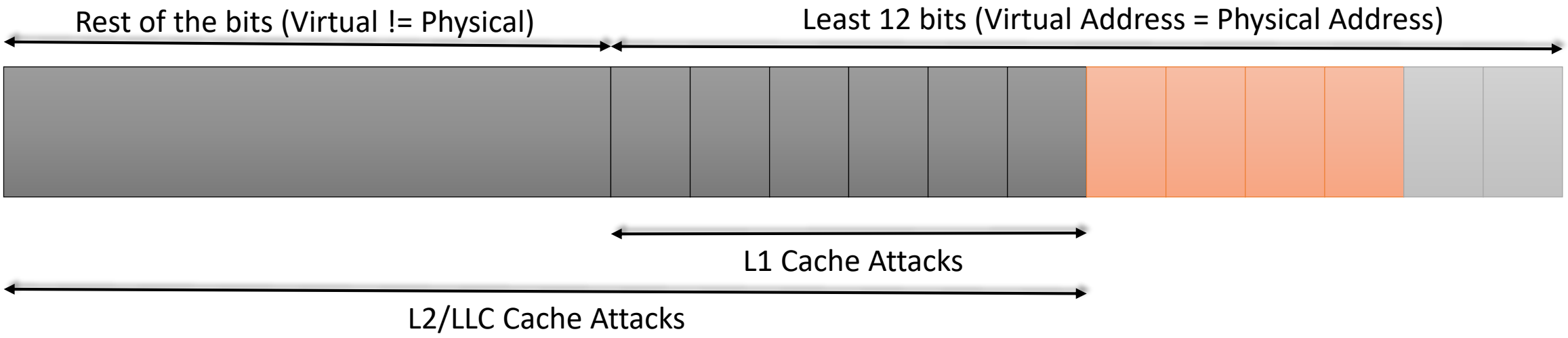
MemJam – Intra Cache Line Resolution



MemJam – Intra Cache Line Resolution

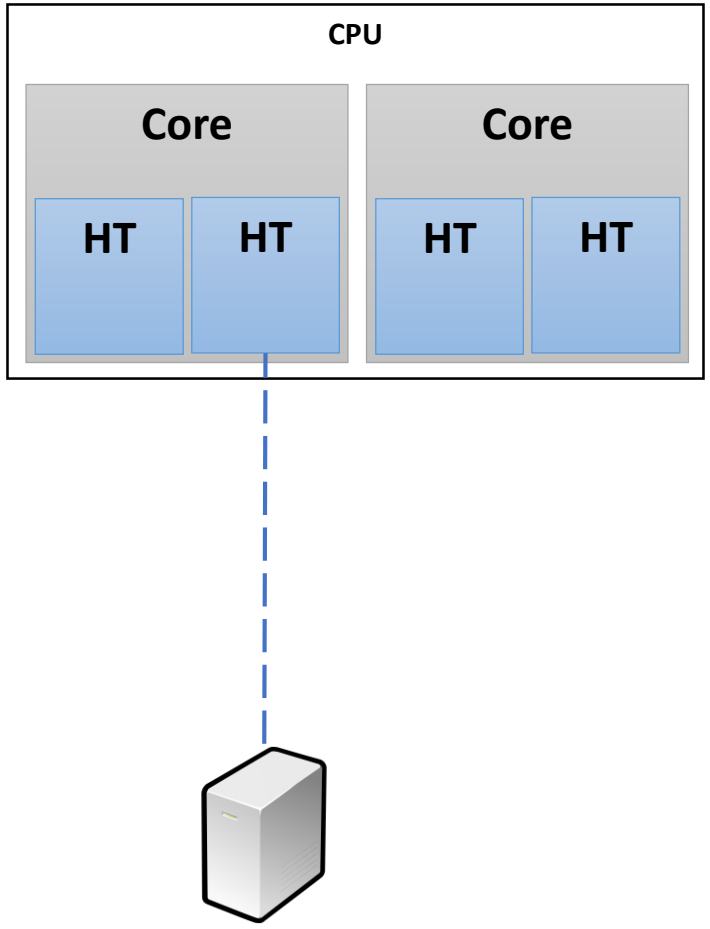


MemJam – Intra Cache Line Resolution



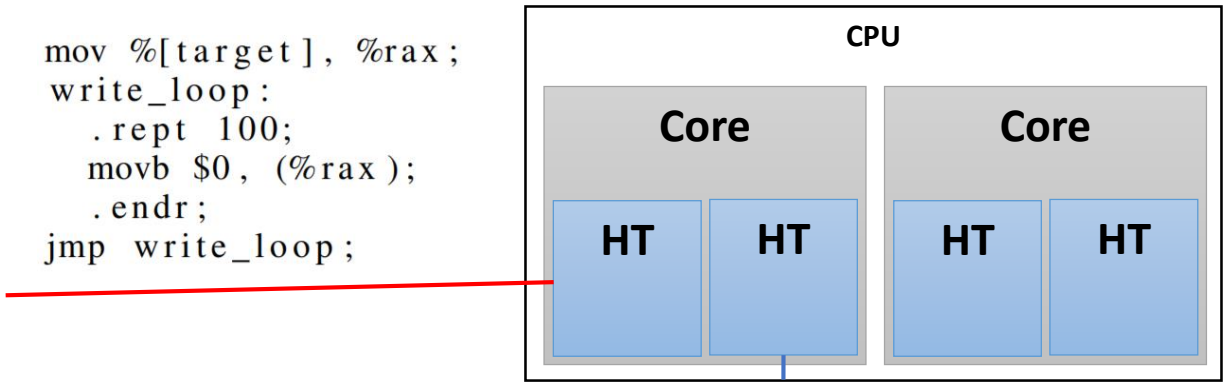
- Intra-cache line Leakage (4-byte granularity)
- Higher time **correlates**→ Memory accesses with the same bit 3 to 12
- 4 bits of intra-cache level leakage

MemJam Attack



MemJam Attack

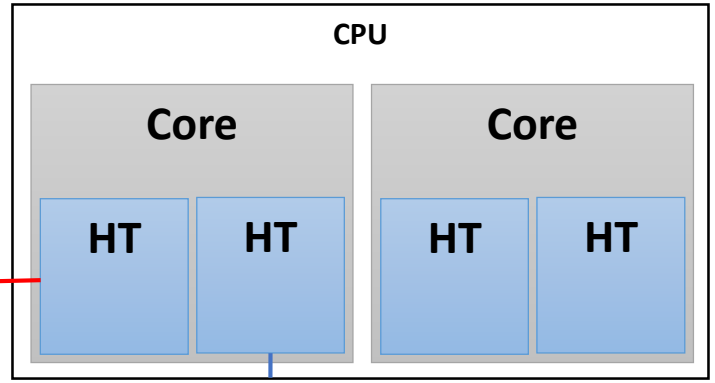
```
mov %[target], %rax;
write_loop:
   .rept 100;
   movb $0, (%rax);
   .endr;
jmp write_loop;
```

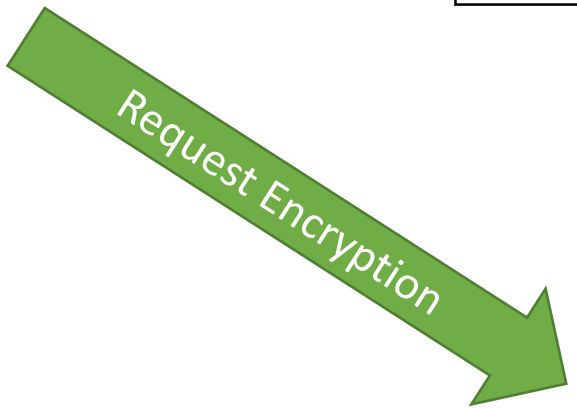


Encryption Service

MemJam Attack

```
mov %[target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
jmp write_loop;
```

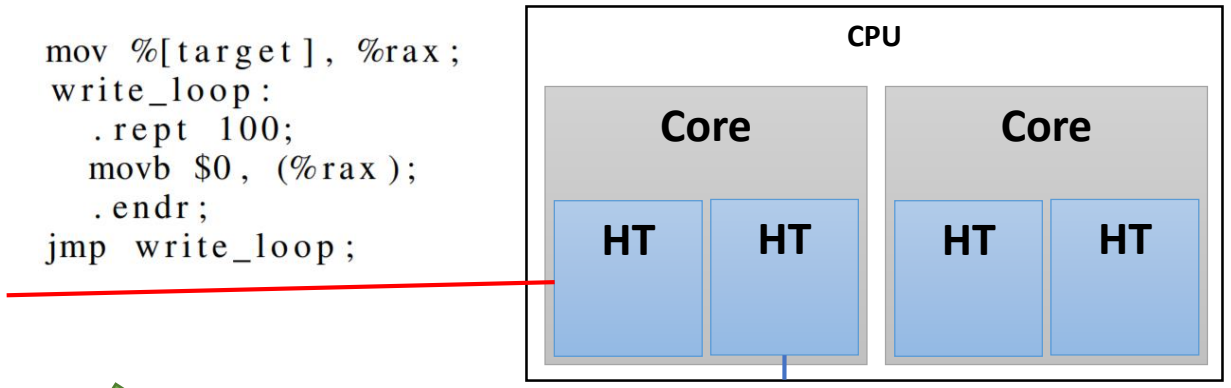


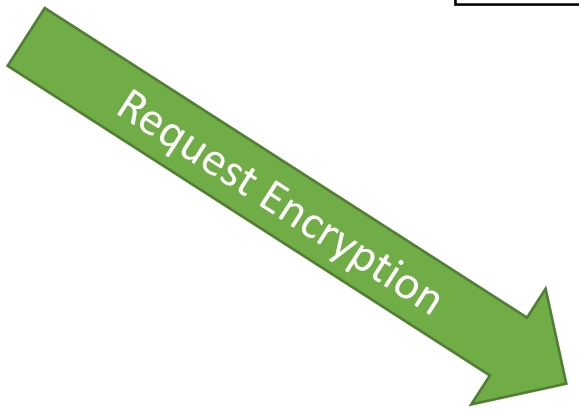


Encryption Service

MemJam Attack

```
mov %[target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
jmp write_loop;
```





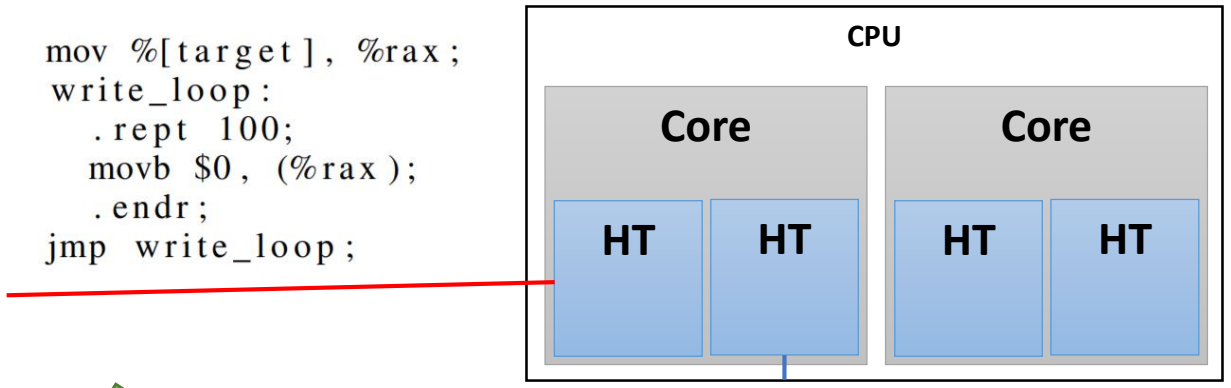
Encryption Service

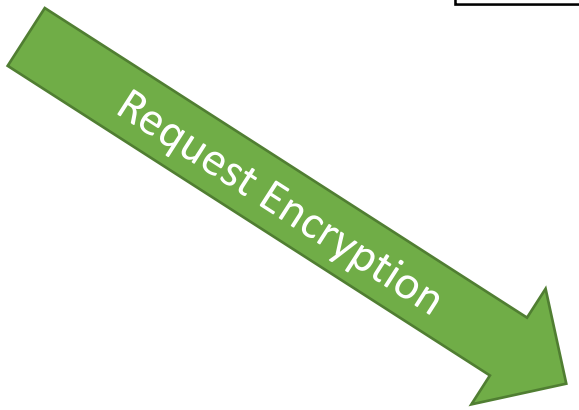
Execute

load
compute
load
load
compute
load
compute
load
load

MemJam Attack

```
mov %[target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
jmp write_loop;
```





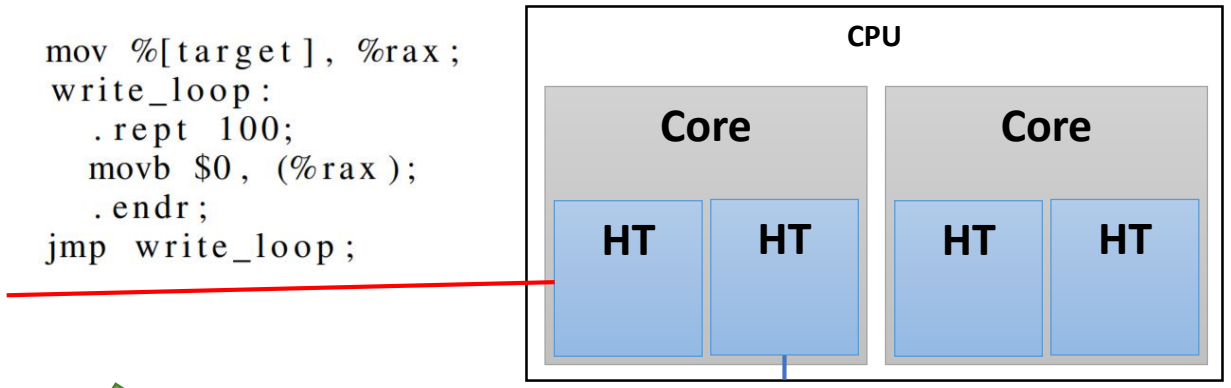
Encryption Service

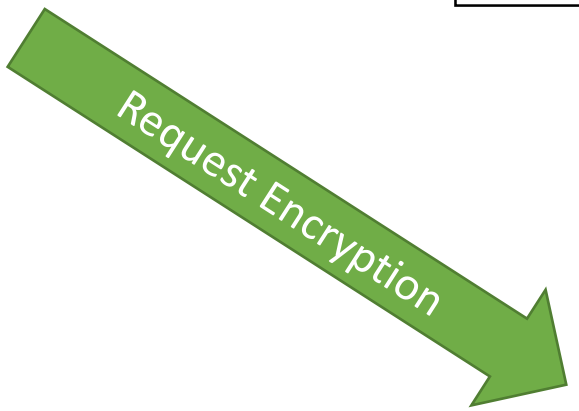
Execute

load
compute
~~load~~
load
~~compute~~
load
compute
~~load~~
load

MemJam Attack

```
mov %[target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
jmp write_loop;
```

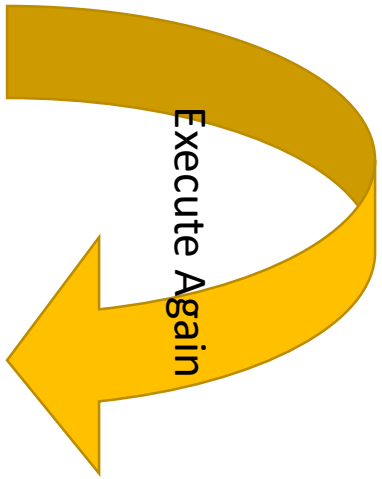




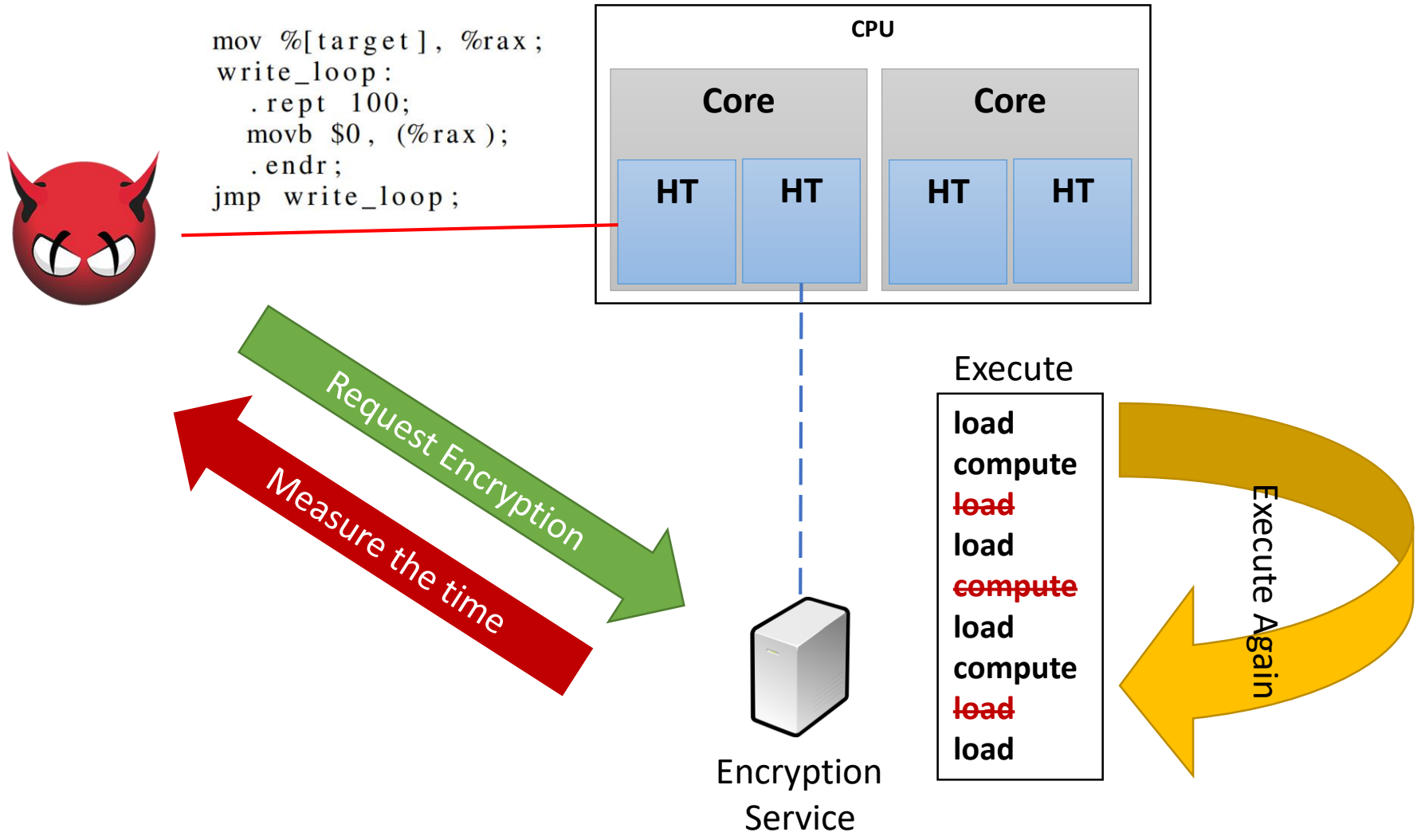
Encryption Service

Execute

load
compute
~~load~~
load
~~compute~~
load
compute
~~load~~
load

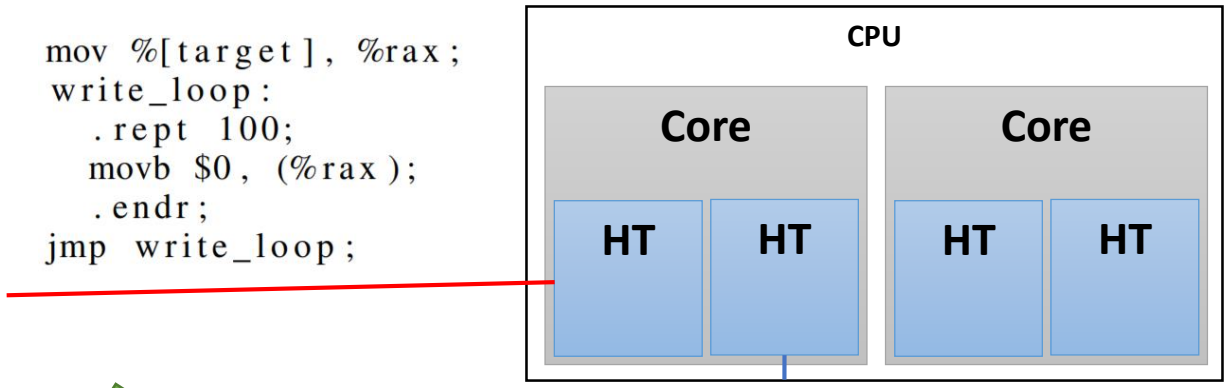


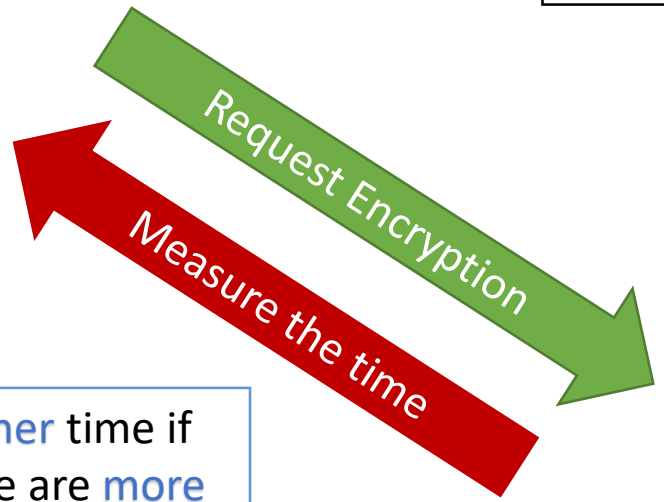
MemJam Attack



MemJam Attack

```
mov %[target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
jmp write_loop;
```



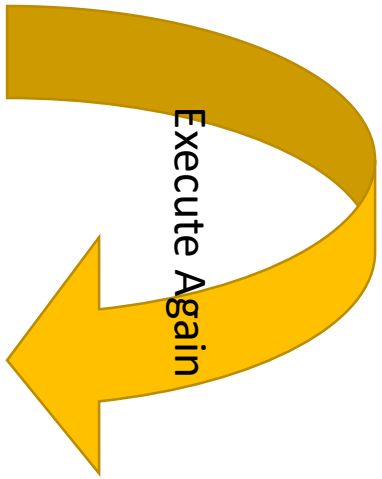


Higher time if there are more number of 4K conflicts

Encryption Service

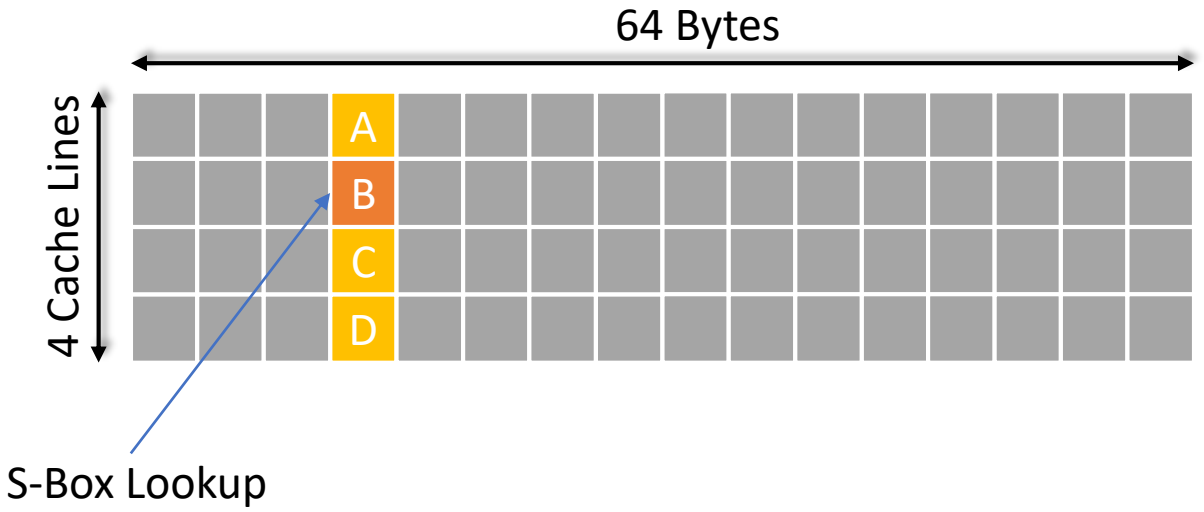
Execute

load
compute
~~load~~
load
~~compute~~
load
compute
~~load~~
load



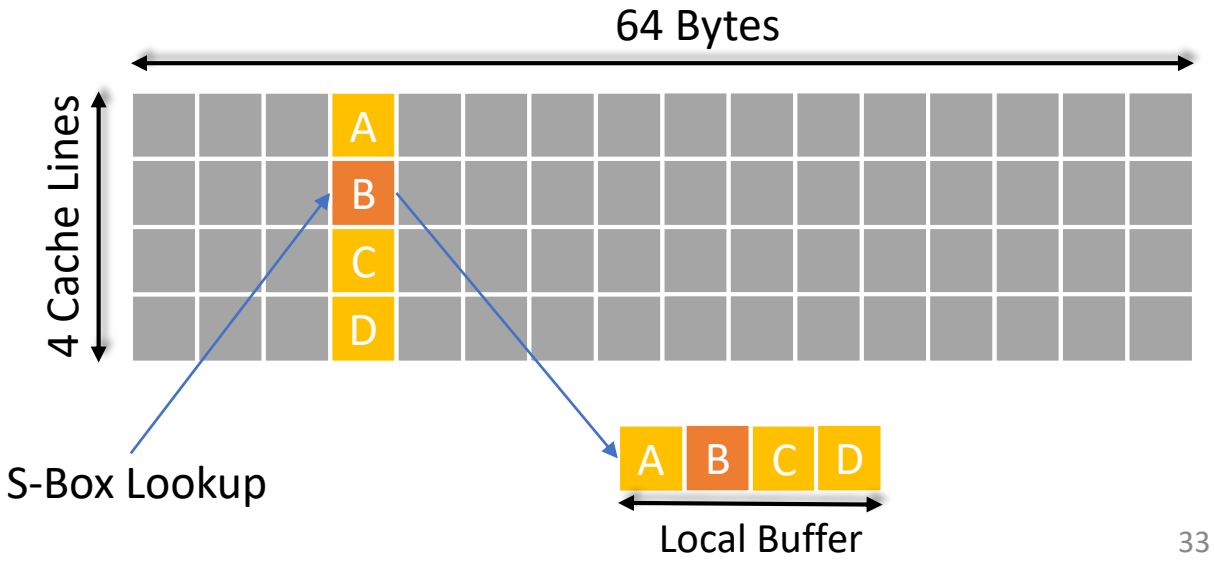
Constant time AES – Safe2Encrypt_RIJ128

- Scatter-gather implementation of AES
  - 256 S-Box – 4 Cache Line
  - Cache independent access pattern
- Implemented and distributed as part of Intel products
  - Intel SGX Linux Software Development Kit (SDK)
  - Intel IPP Cryptography Library



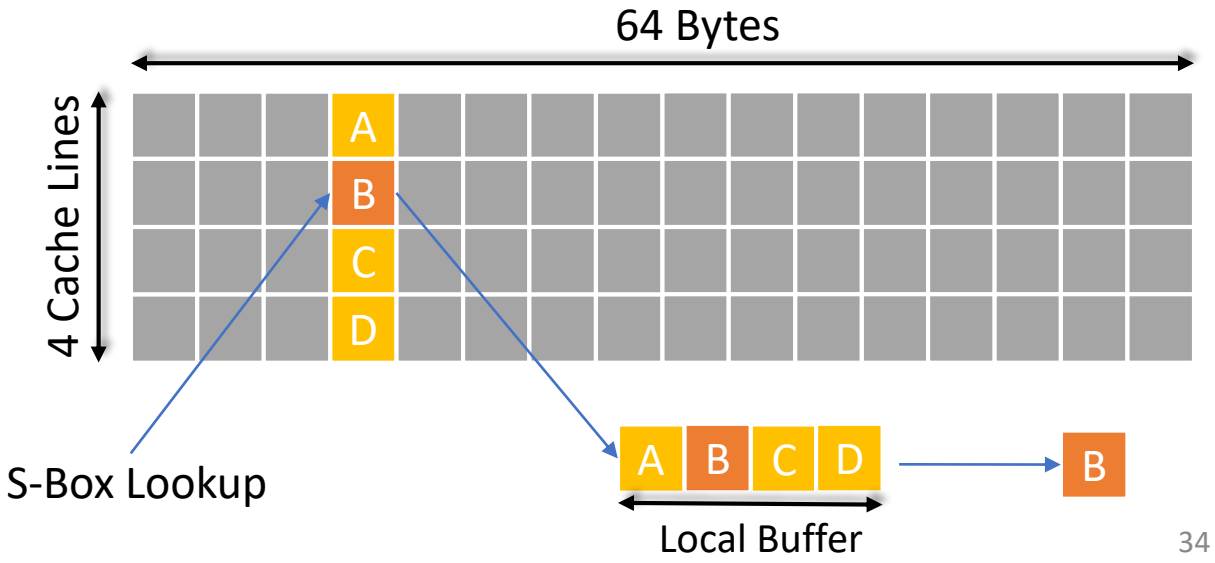
Constant time AES – Safe2Encrypt_RIJ128

- Scatter-gather implementation of AES
  - 256 S-Box – 4 Cache Line
  - Cache independent access pattern
- Implemented and distributed as part of Intel products
  - Intel SGX Linux Software Development Kit (SDK)
  - Intel IPP Cryptography Library

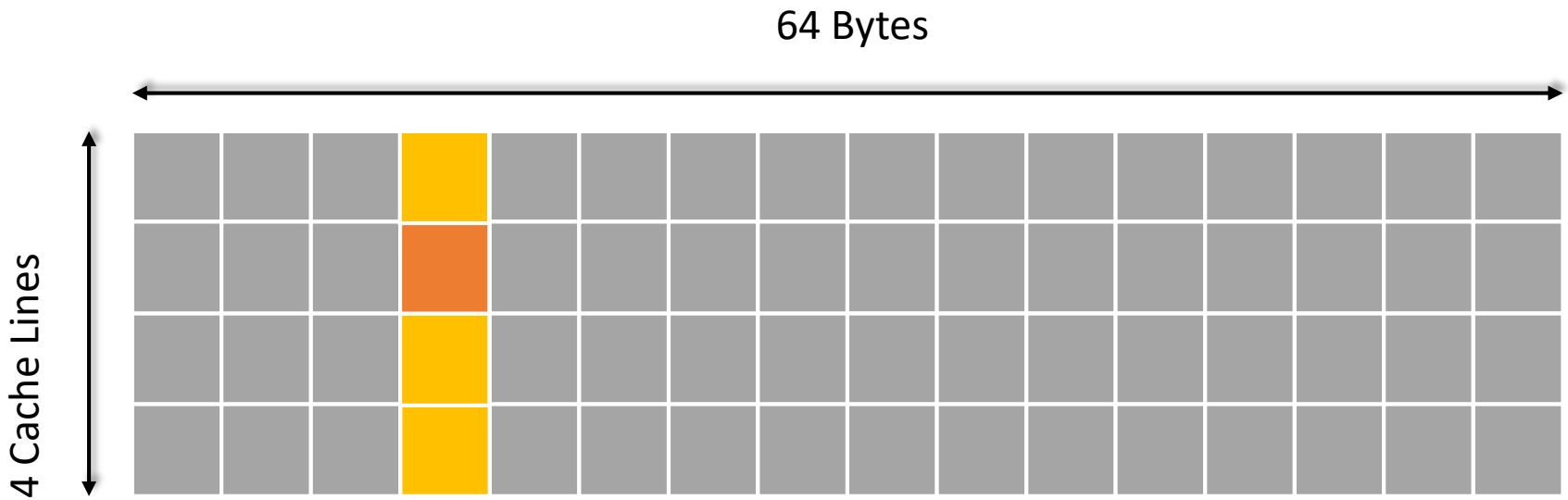


Constant time AES – Safe2Encrypt_RIJ128

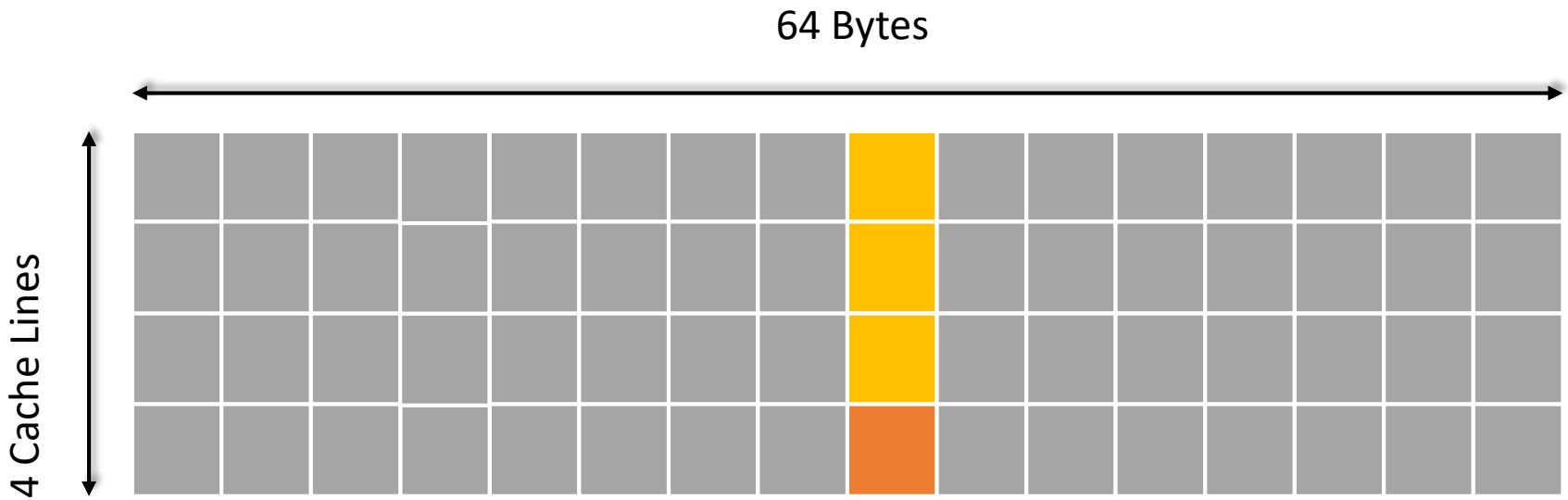
- Scatter-gather implementation of AES
  - 256 S-Box – 4 Cache Line
  - Cache independent access pattern
- Implemented and distributed as part of Intel products
  - Intel SGX Linux Software Development Kit (SDK)
  - Intel IPP Cryptography Library



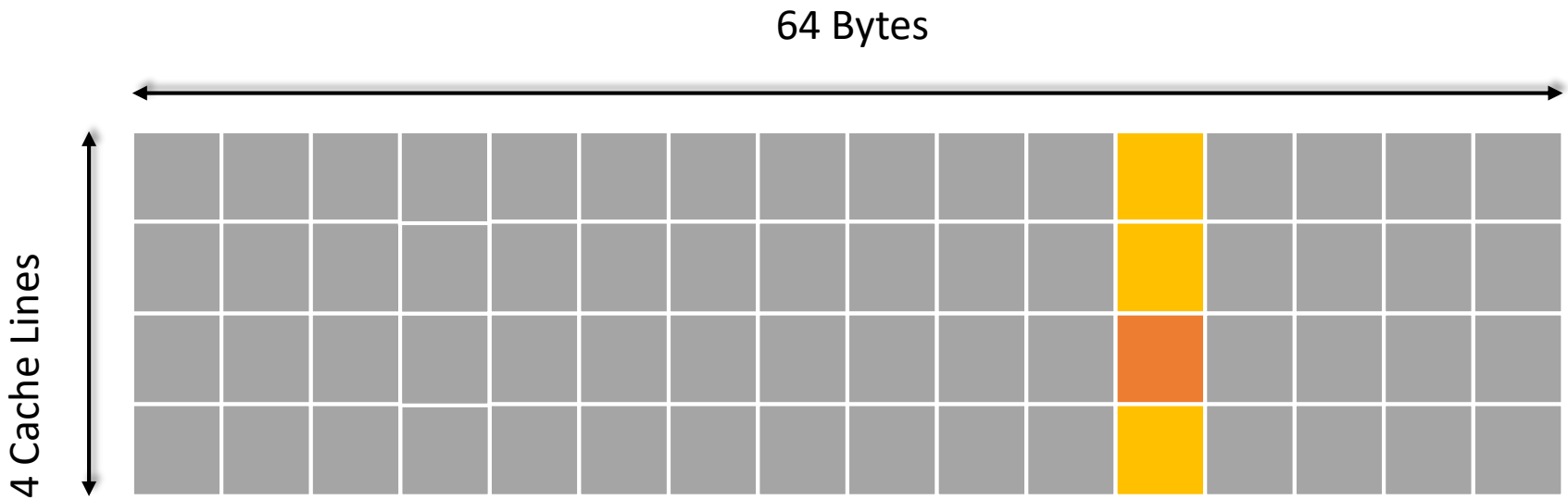
MemJam Attack on AES



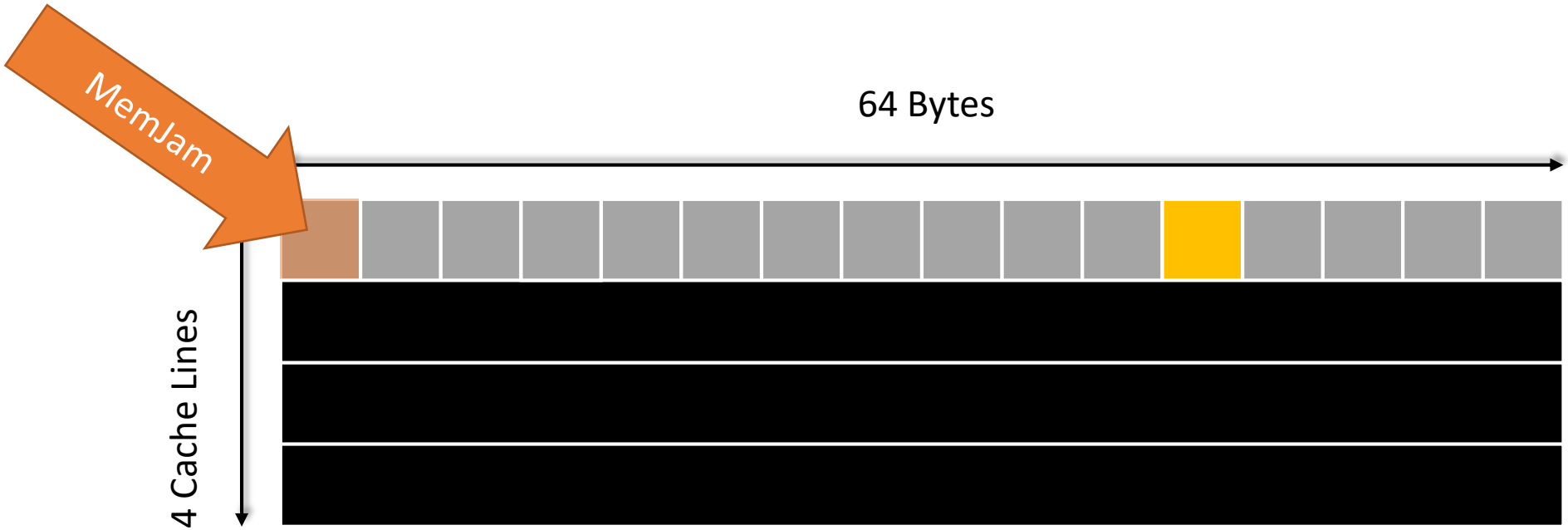
MemJam Attack on AES



MemJam Attack on AES

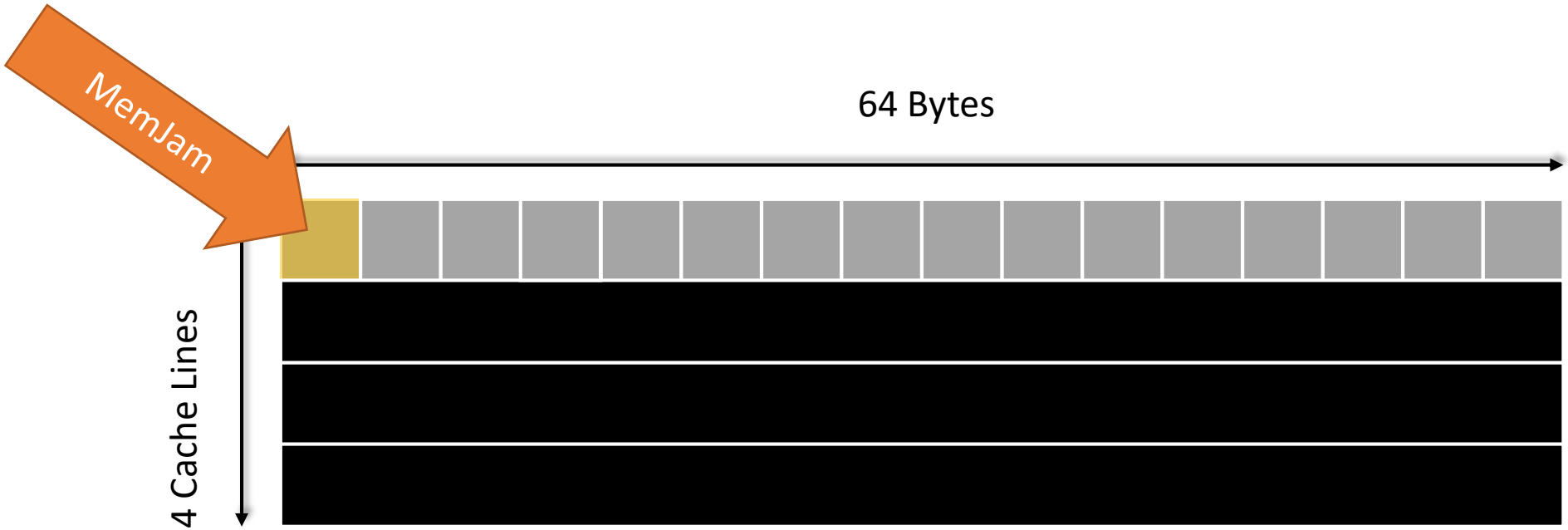


MemJam Attack on AES



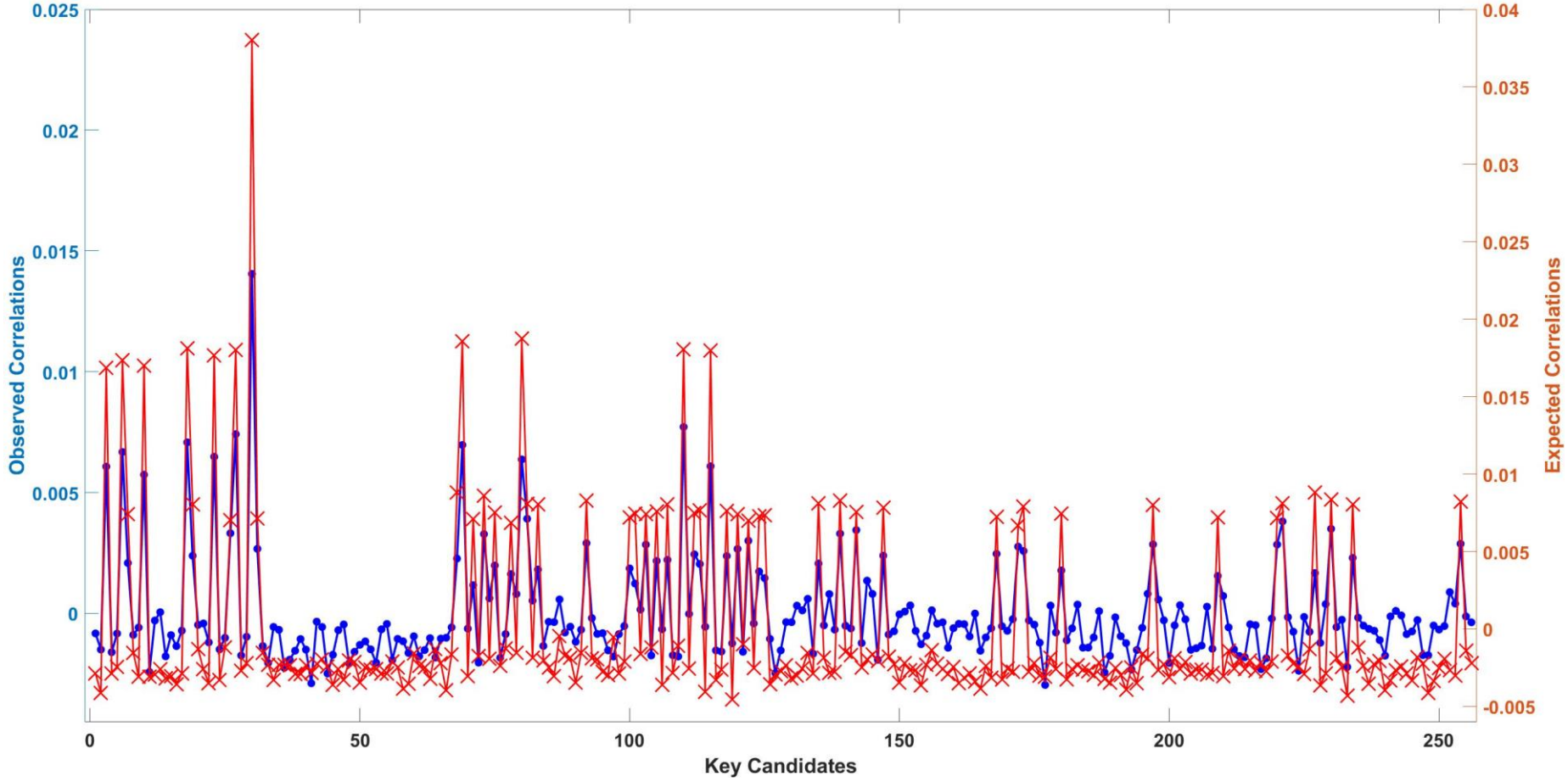
$$index = S^{-1}(c \oplus k)$$

MemJam Attack on AES

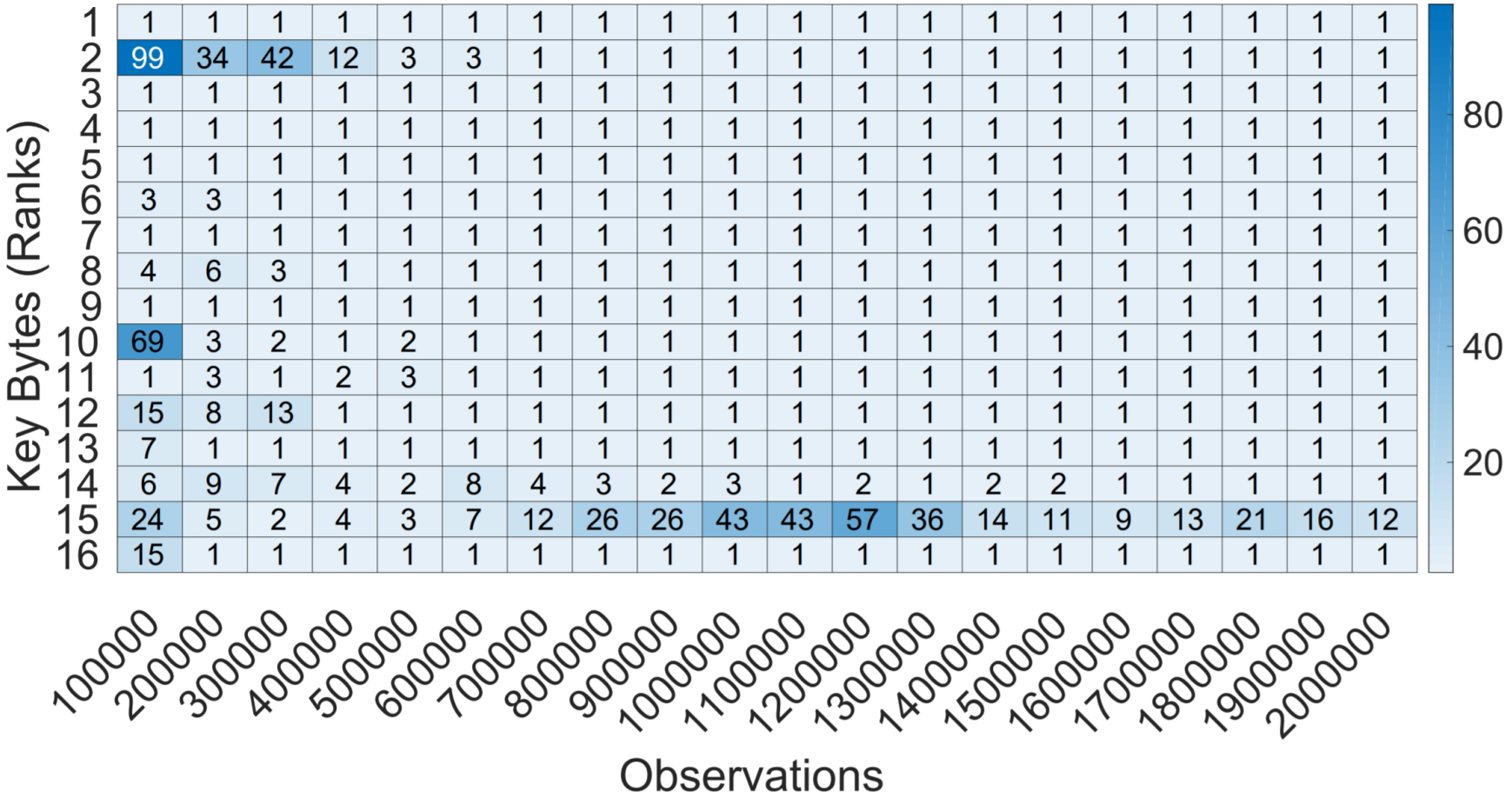


$$index = S^{-1}(c \oplus k) \longrightarrow index < 4$$

AES Key Recovery

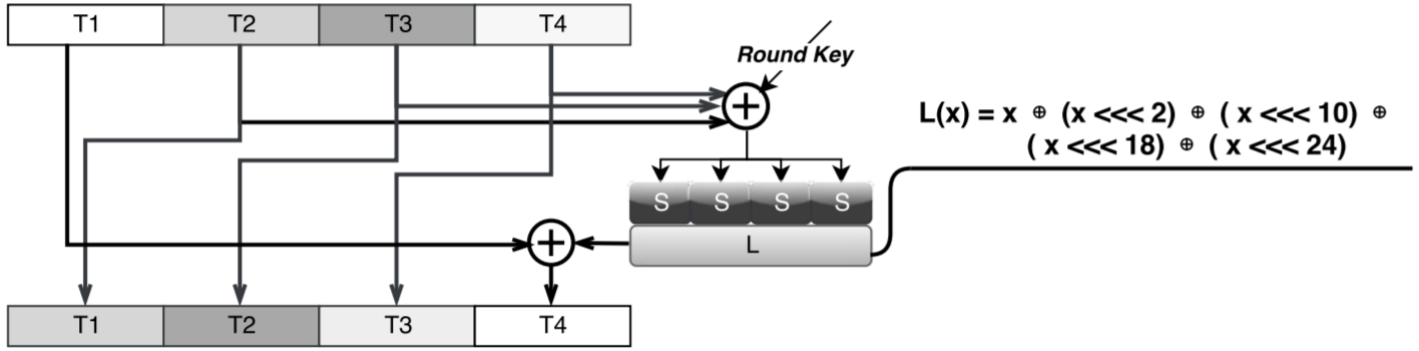


AES Key Recovery



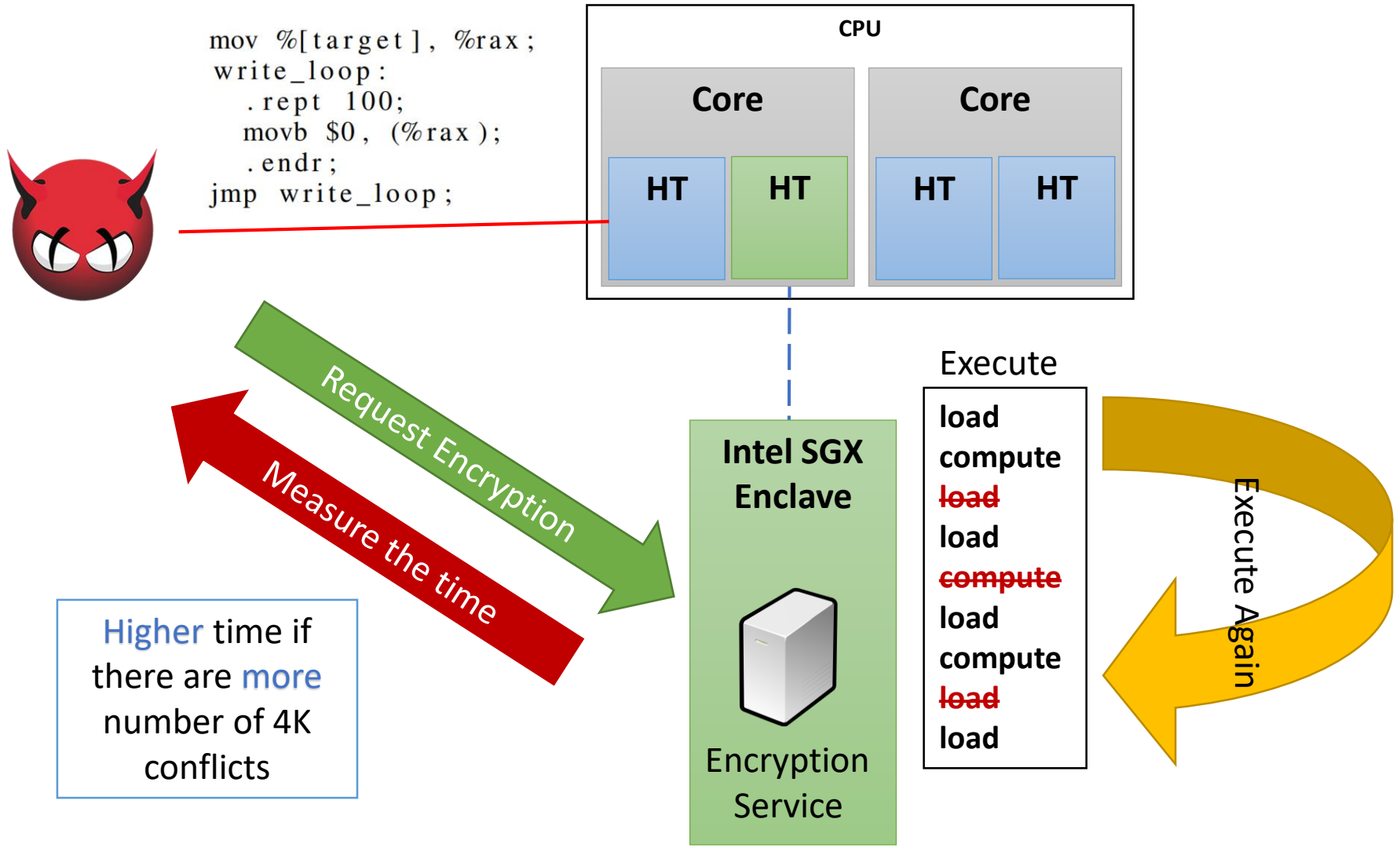
SM4 Block cipher – cpSMS4_Cipher

- Standard Cipher support by Intel
  - Chinese National Standard for Wireless LAN WAPI
- S-Box + Unbalanced Feistel Structure
- Protected by Cache State Normalization

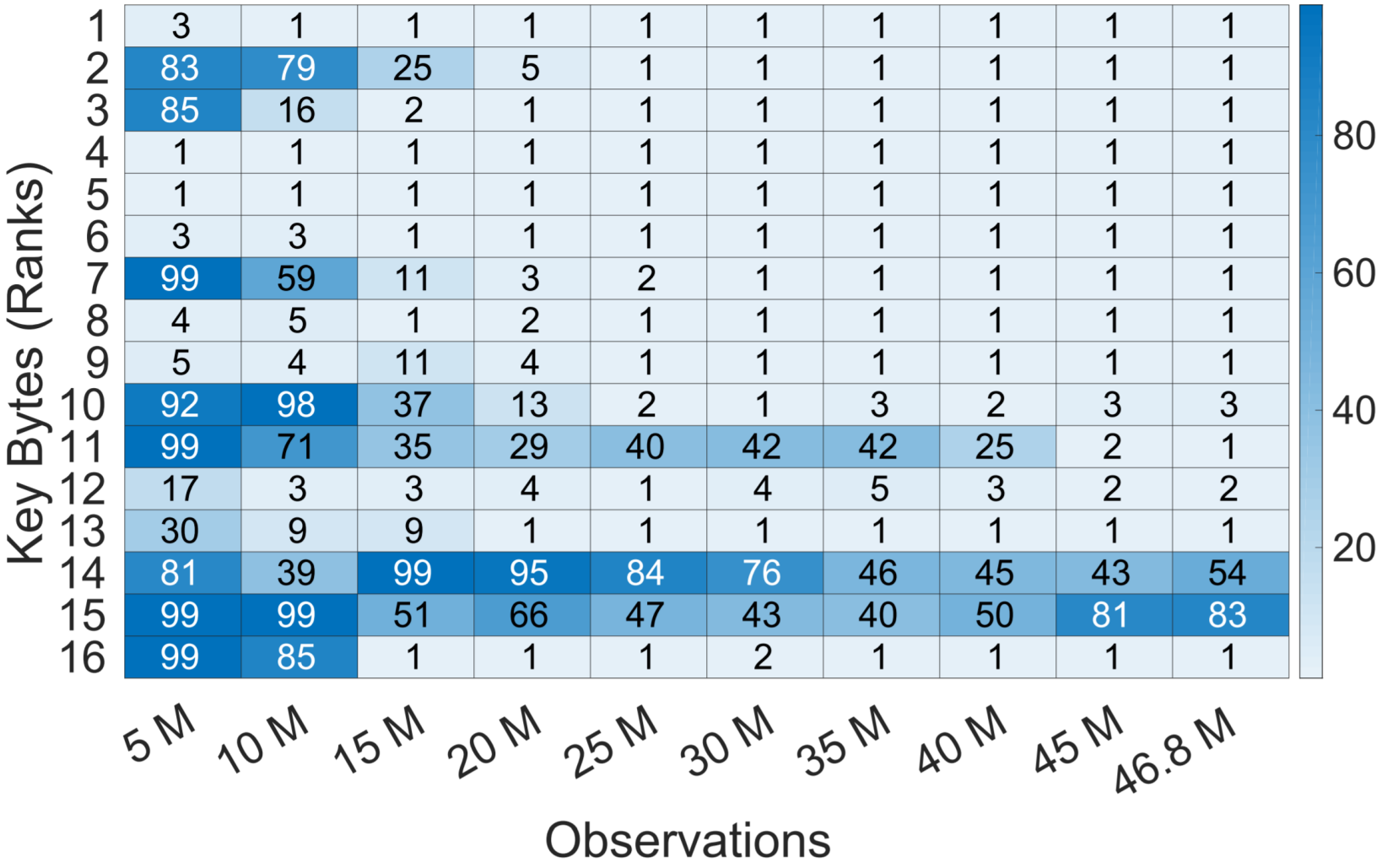


- Recursive attack → Full key recovery with 40K observations

MemJaming Intel SGX Secure Enclave



Intel SGX – AES Key Recovery



Conclusion

- New Side-Channel Attack Applicable to all Intel Processors
  - Intel SGX extensions
- Bypass of Constant-Time Implementations Techniques
  - Scatter-Gather
  - Cache State Normalization
- Agnostic to other Cache Attack Defense Mechanism

- Intel Trilogy
  - Intel Hardware
  - Intel Trusted Execution Environment
  - Intel Hardened Crypto Implementation

Responsible Disclosure

Date	Progress
08/02/2017	Reported
08/04/2017	Acknowledged
11/07/2017	Safe2Encrypt_RIJ128 got removed from SGX SDK.
11/17/2017	CVE-2017-5737 Assigned
work-in-progress	Patch

Questions?!

Vernam Group

v.wpi.edu

@VernamGroup

@danielmgmi

Implementation Technique	Function Name	l9 n0 y8 k0 e9	m7 mx	n8	Linux SGX SDK
AES-NI	Encrypt_RIJ128_AES_NI	✓	×	×	✓ (pre-built)
AES Bitsliced	SafeEncrypt_RIJ128	✓	×	✓	✓ (pre-built)
AES Constant-Time	Safe2Encrypt_RIJ128	×	✓	×	✓ (source)
SM4 Bitsliced using AES-NI	cpSMS4_ECB_aesni	✓	×	×	N/A
SM4 Cache Normalization	cpSMS4_Cipher	✓	✓	✓	N/A

Release	Family	Cache Bank Conflicts	4K Aliasing
2006	Core	✓	✓
2008	Nehalem	×	✓
2011	Sandy bridge	✓	✓
2013	Silvermont, Haswell, Broadwell	×	✓
2015	Skylake	×	✓
2016	KabyLake	×	✓