Safecracker: Leaking Secrets through Compressed Caches

Po-An Tsai, Andres Sanchez,
Christopher Fletcher, and Daniel Sanchez

ASPLOS 2020

Massachusetts Institute of Technology
CSAIL
ILLINOIS

Executive Summary

- First security analysis of cache compression

Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data

Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets

Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets

Attacker

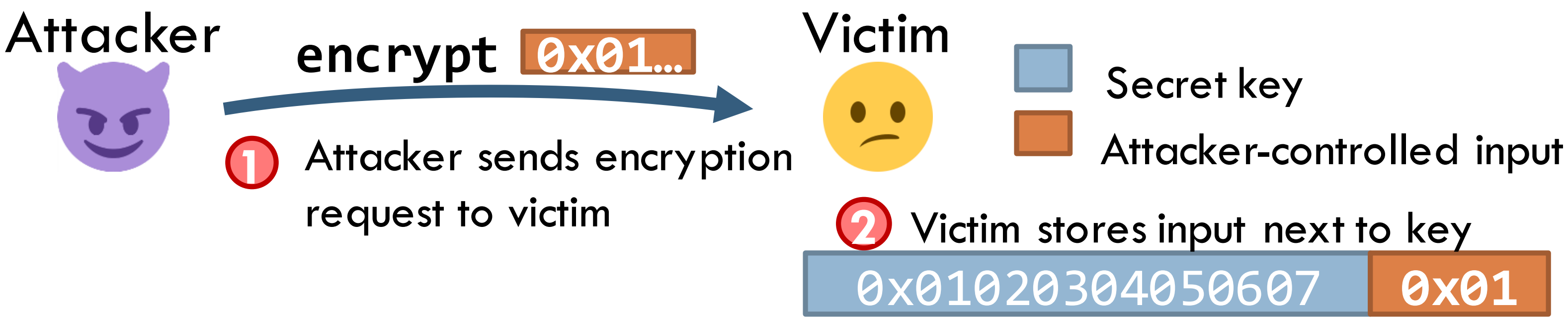
Victim

Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets

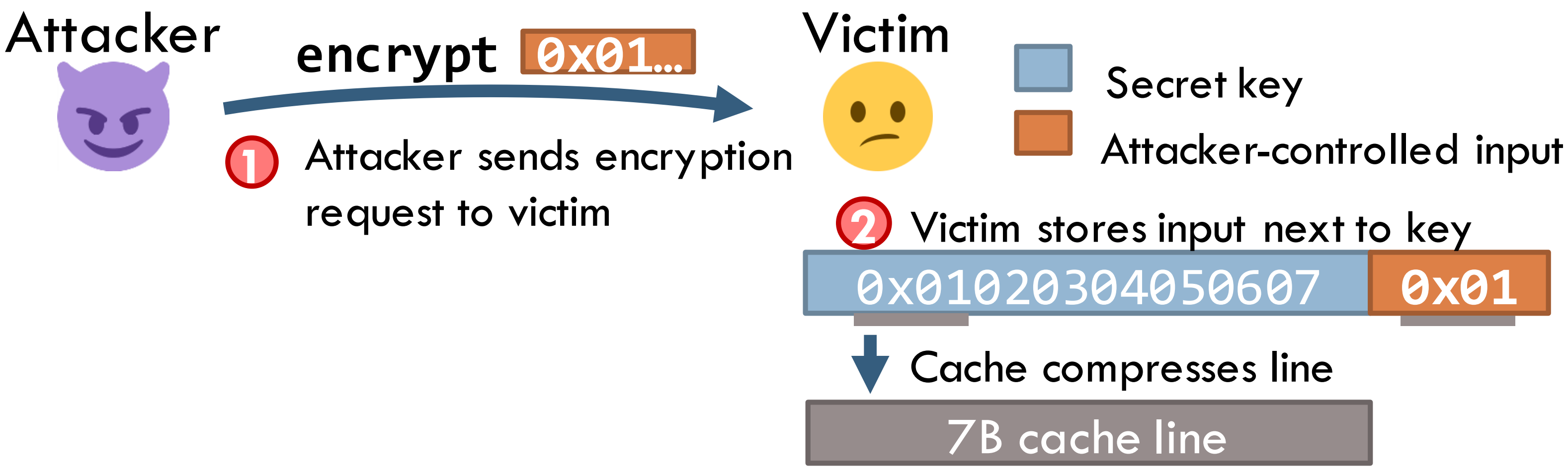
Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets



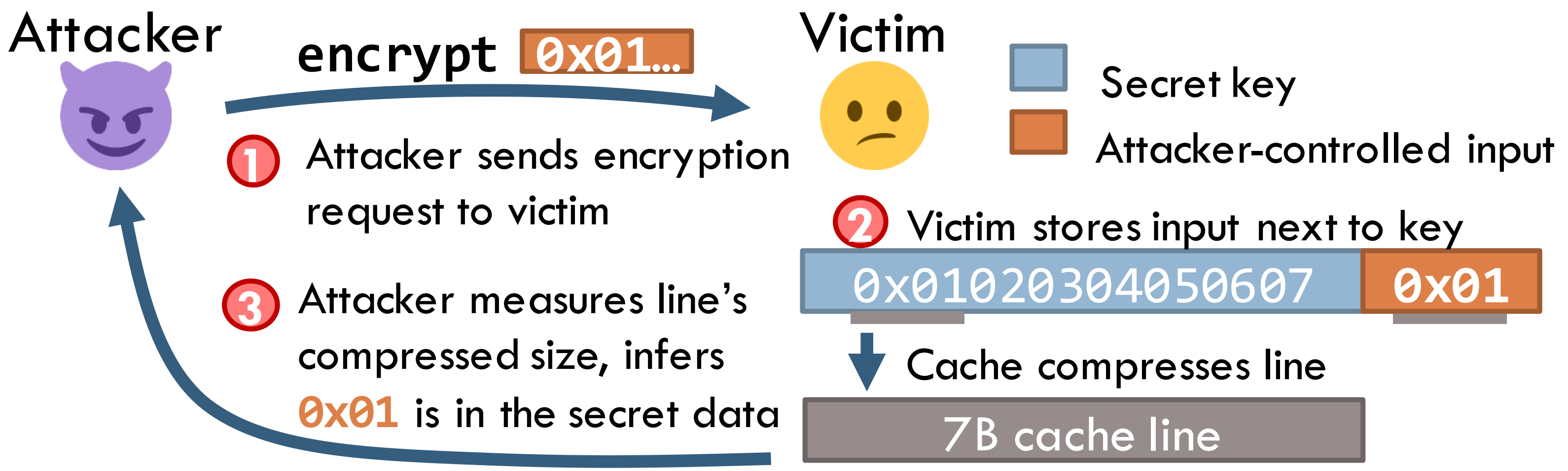
Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets



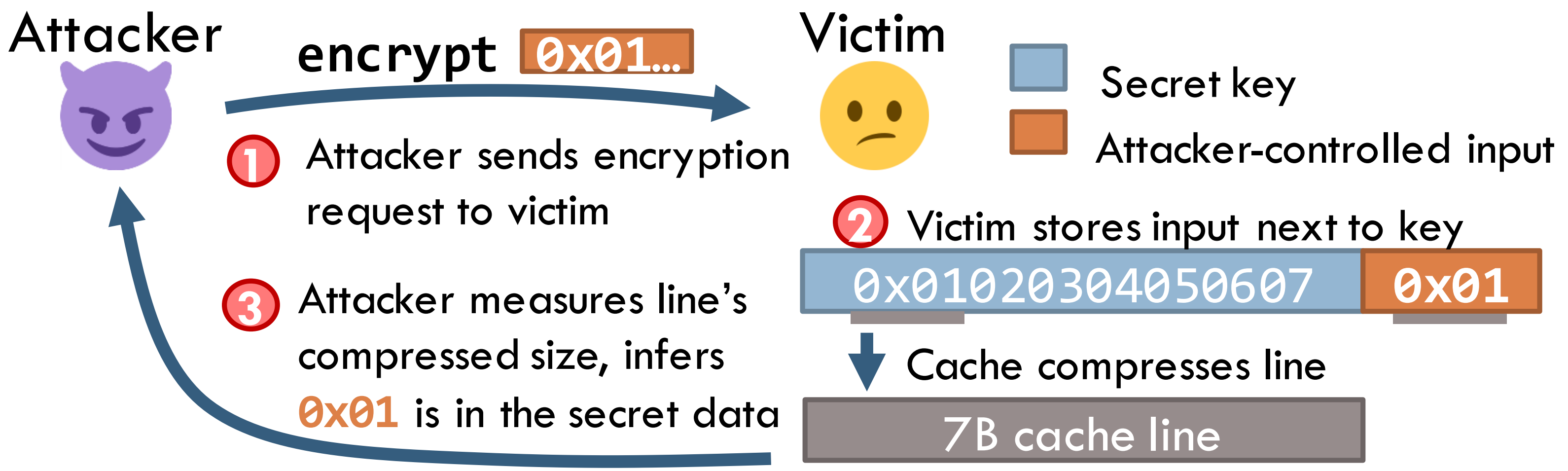
Executive Summary

- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets



Executive Summary

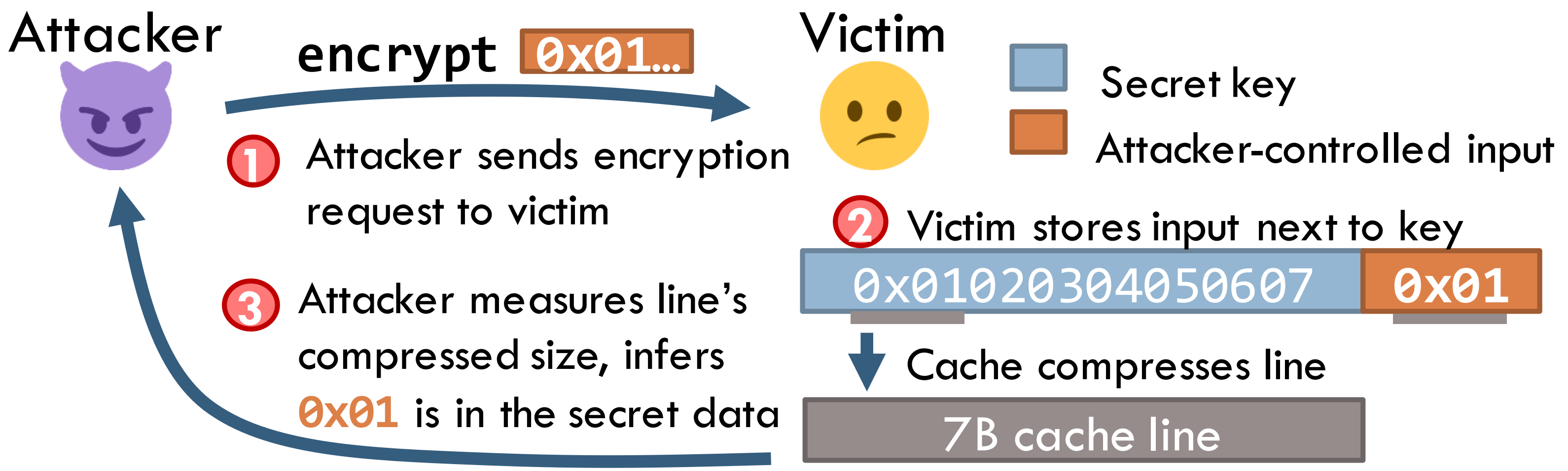
- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets



Compromises secret key in ~10ms

Executive Summary

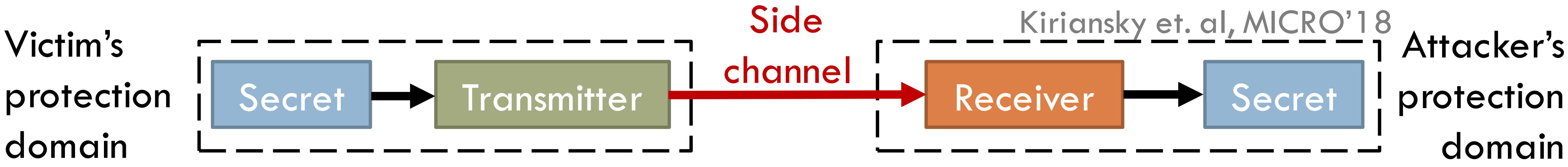
- First security analysis of cache compression
- Compressibility of a cache line reveals info about its data
- Attacker can exploit data colocation to leak secrets



Compromises secret key in ~10ms

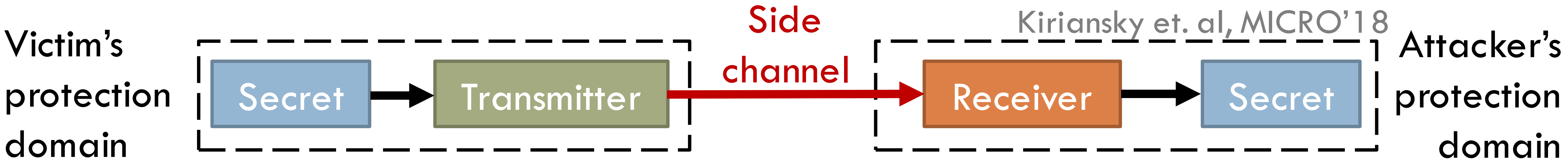
Leaks large fraction of victim memory when combined latent memory safety vulnerabilities

Speculation-Based vs. Compressed Cache Side-Channel Attacks



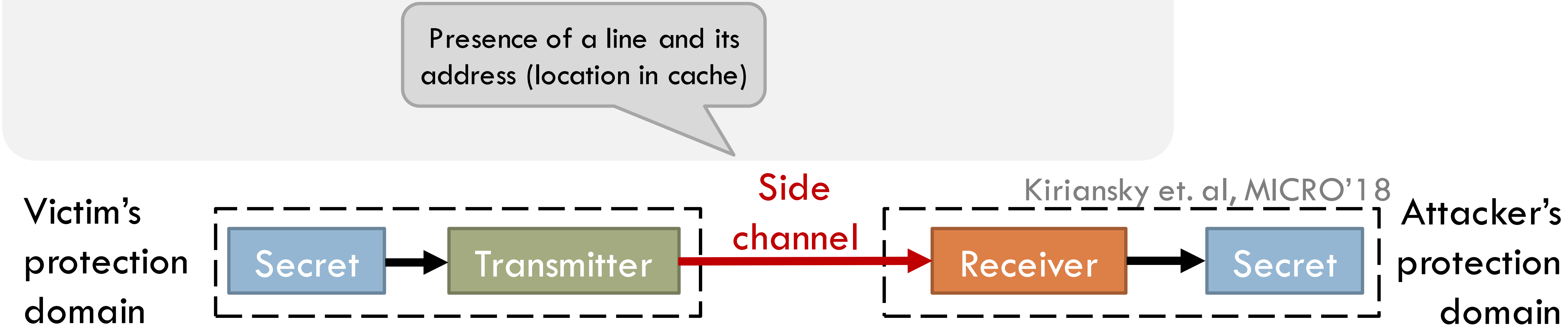
Speculation-Based vs. Compressed Cache Side-Channel Attacks

Speculation-based cache side channel attacks (e.g., Spectre)

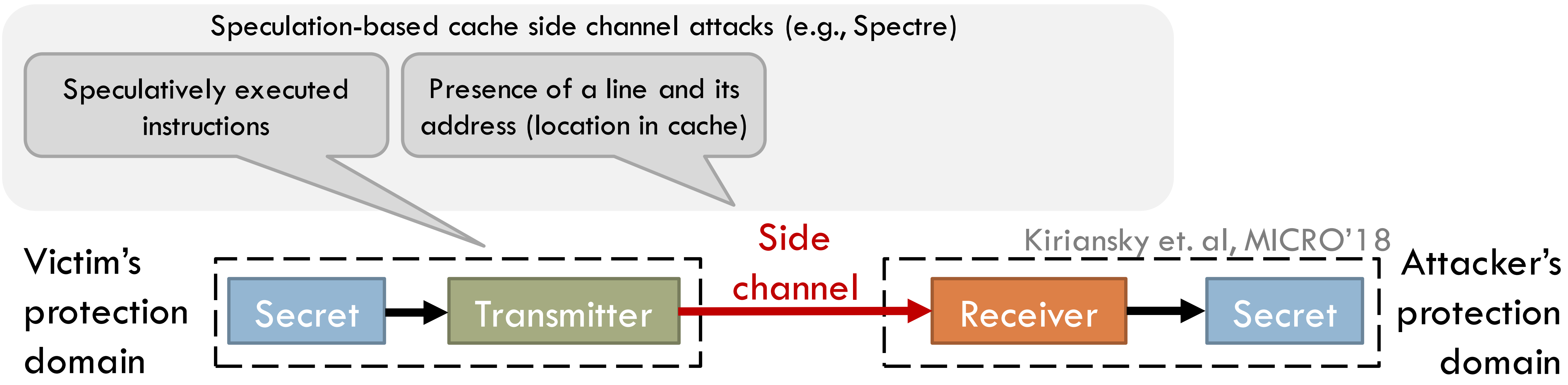


Speculation-Based vs. Compressed Cache Side-Channel Attacks

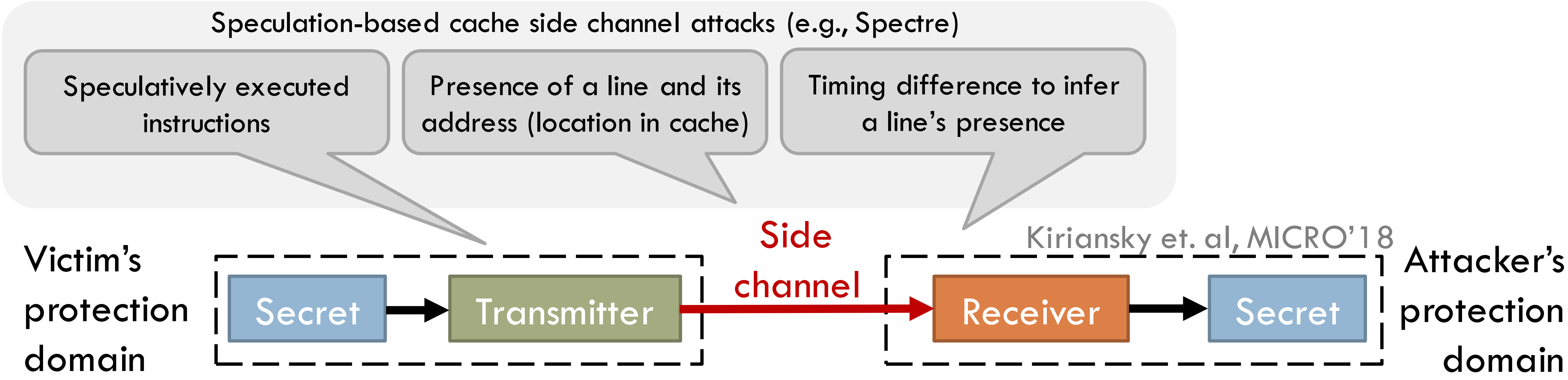
Speculation-based cache side channel attacks (e.g., Spectre)



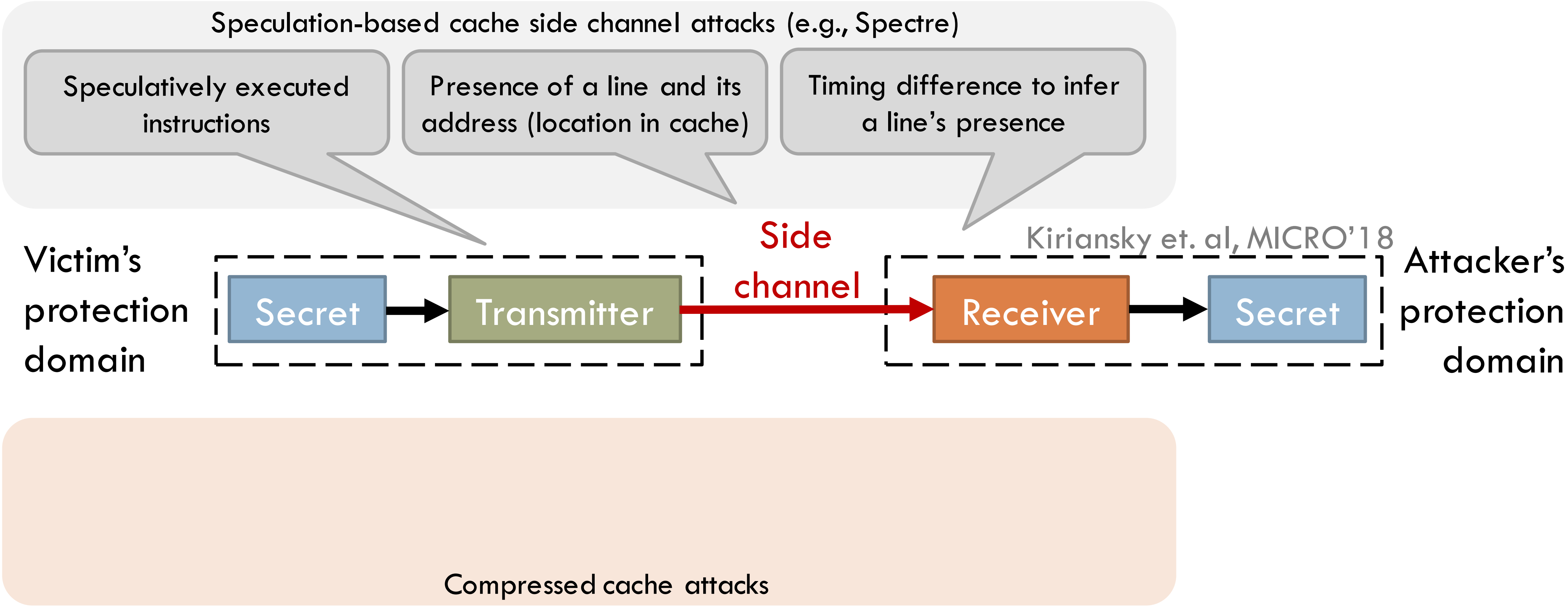
Speculation-Based vs. Compressed Cache Side-Channel Attacks



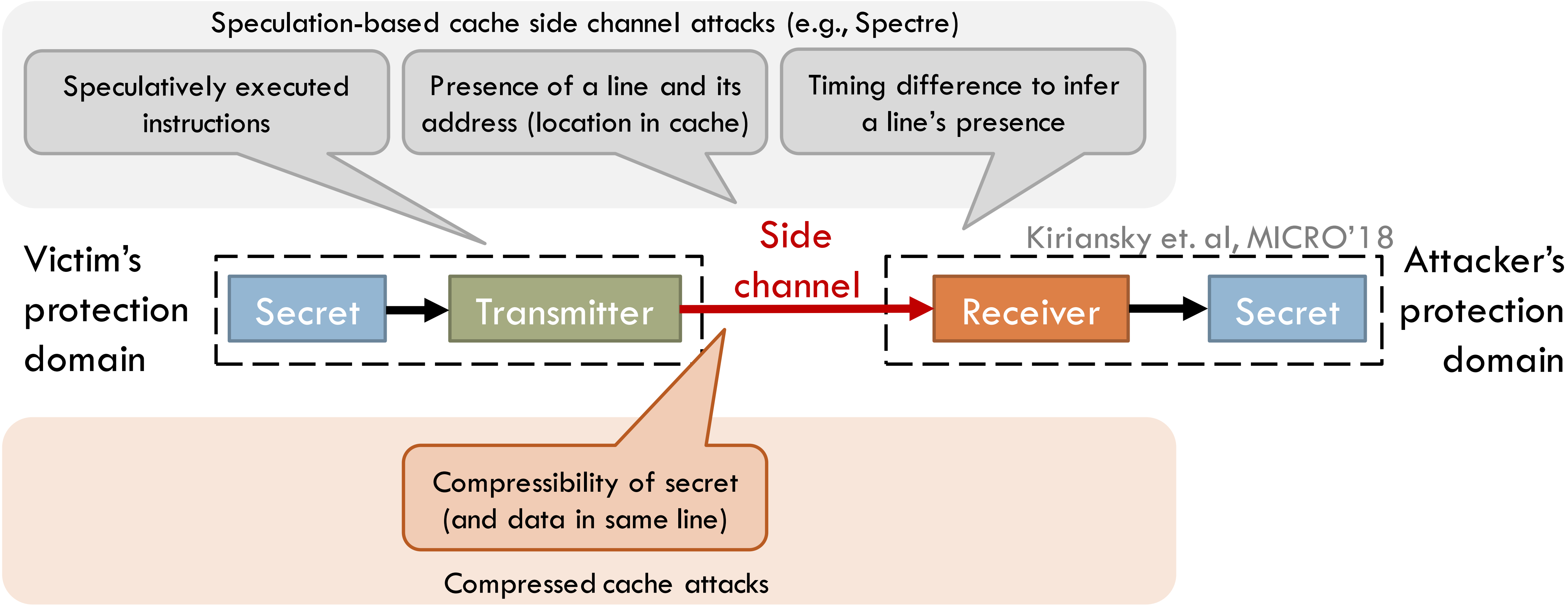
Speculation-Based vs. Compressed Cache Side-Channel Attacks



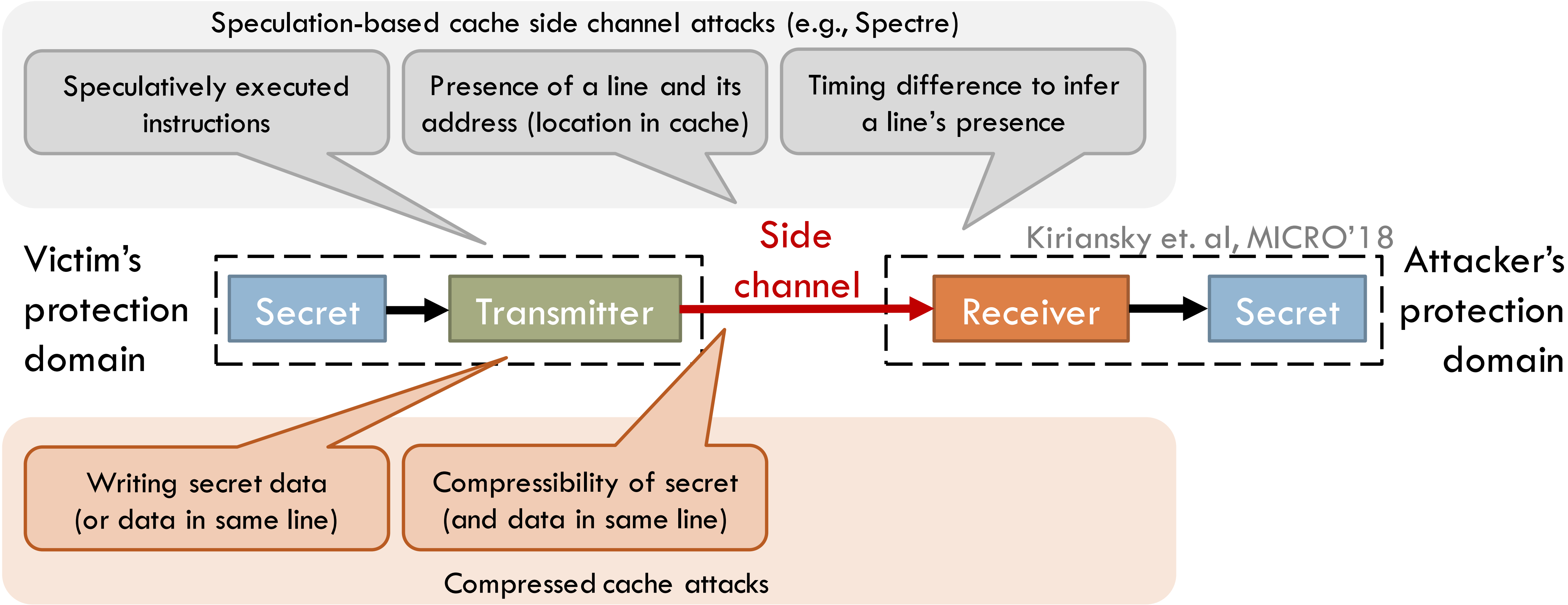
Speculation-Based vs. Compressed Cache Side-Channel Attacks



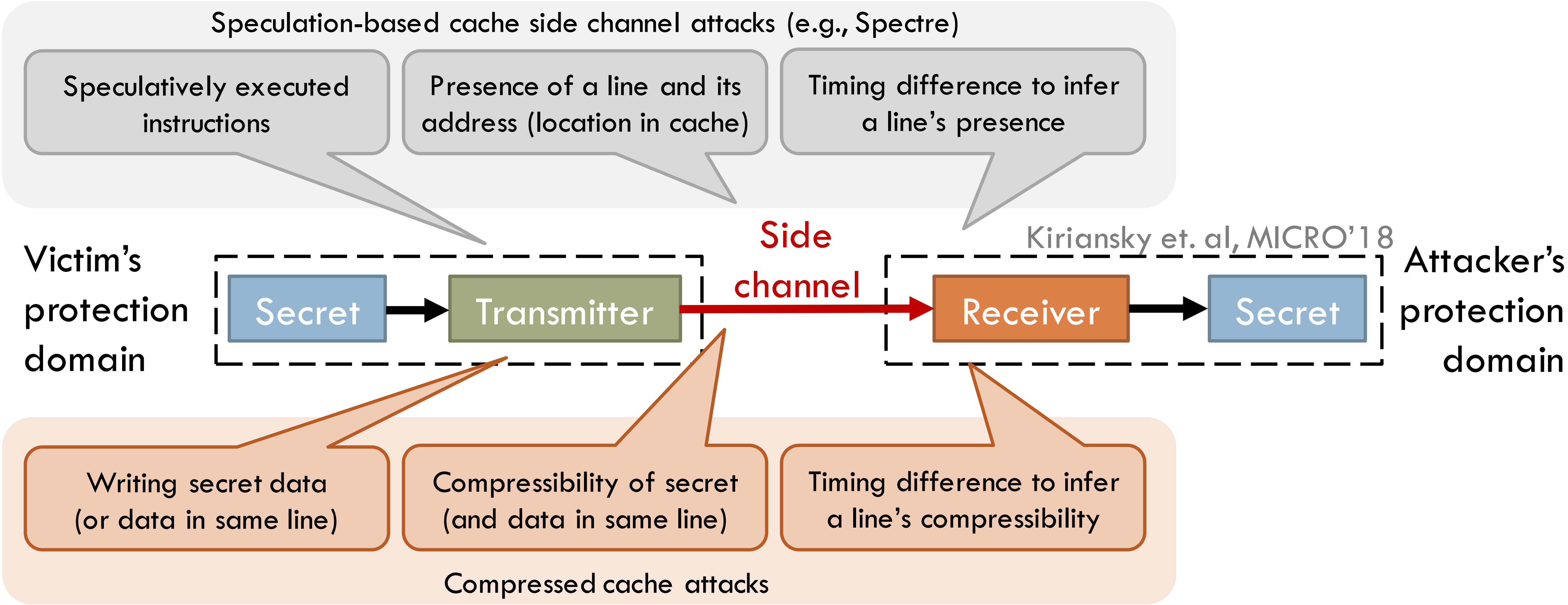
Speculation-Based vs. Compressed Cache Side-Channel Attacks



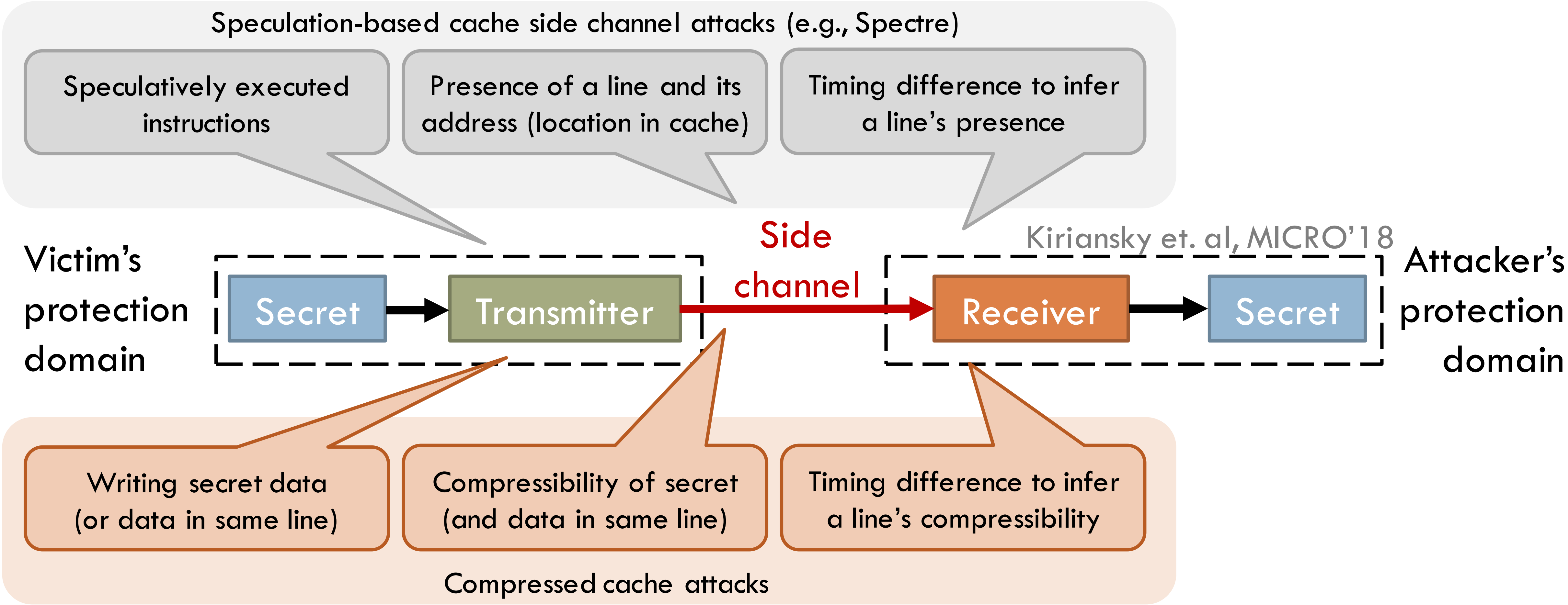
Speculation-Based vs. Compressed Cache Side-Channel Attacks



Speculation-Based vs. Compressed Cache Side-Channel Attacks



Speculation-Based vs. Compressed Cache Side-Channel Attacks



Compressed cache attacks leak data without relying on speculation

Outline

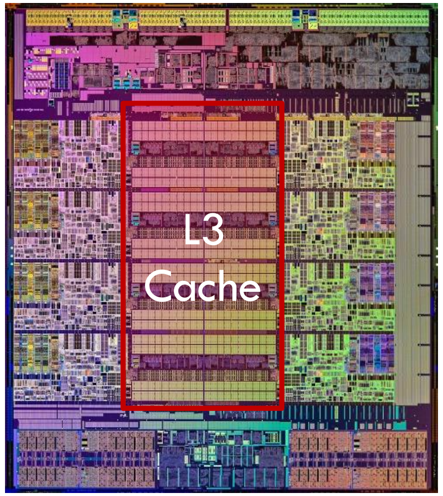
- Background on cache compression
- Pack+Probe: Measuring cache line compressibility
- Safecracker: Exploiting data colocation to leak secrets
- Potential defenses

Cache Compression Tradeoffs

- Higher effective capacity → Higher hit rate
- Somewhat higher hit latency

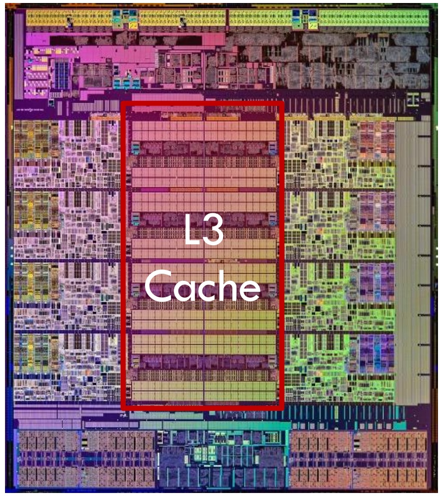
Cache Compression Tradeoffs

- Higher effective capacity → Higher hit rate
- Somewhat higher hit latency
- Highly beneficial for large caches (e.g., LLC)



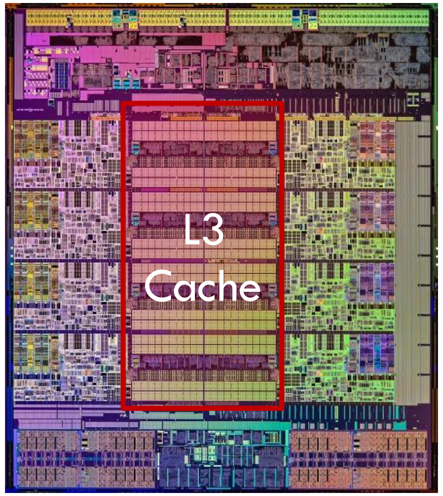
Cache Compression Tradeoffs

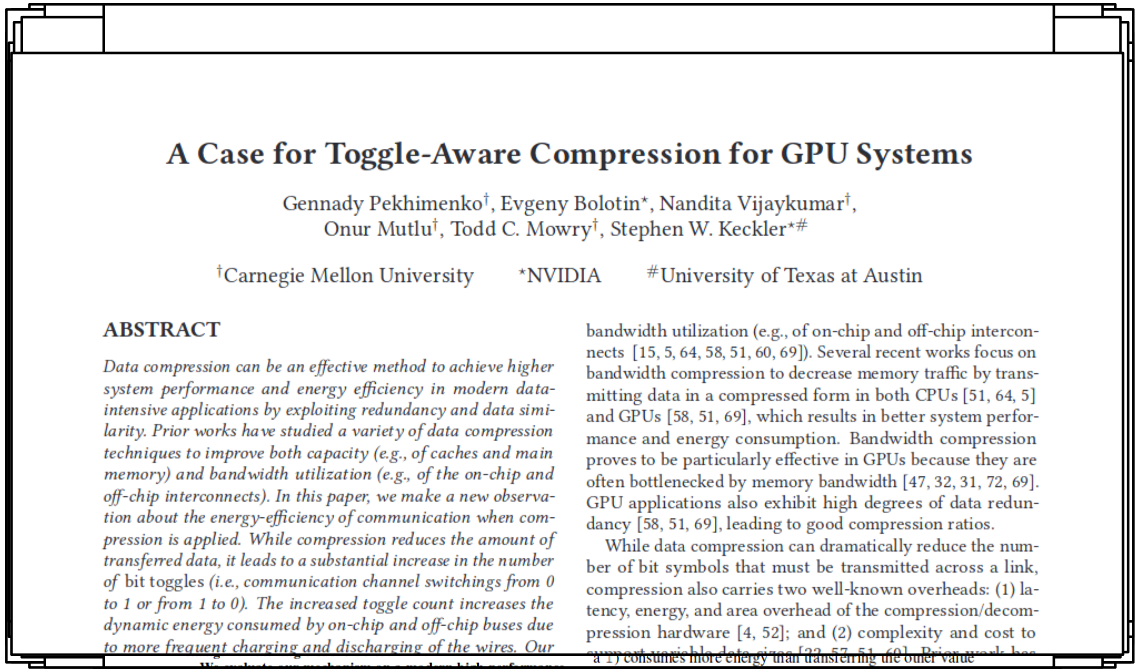
- Higher effective capacity → Higher hit rate
- Somewhat higher hit latency
- Highly beneficial for large caches (e.g., LLC)
- Intense research activity over past 15 years



Cache Compression Tradeoffs

- Higher effective capacity → Higher hit rate
- Somewhat higher hit latency
- Highly beneficial for large caches (e.g., LLC)
- Intense research activity over past 15 years





A Case for Toggle-Aware Compression for GPU Systems

Gennady Pekhimenko†, Evgeny Bolotin*, Nandita Vijaykumar†,
Onur Mutlu†, Todd C. Mowry†, Stephen W. Keckler*#

†Carnegie Mellon University *NVIDIA #University of Texas at Austin

ABSTRACT

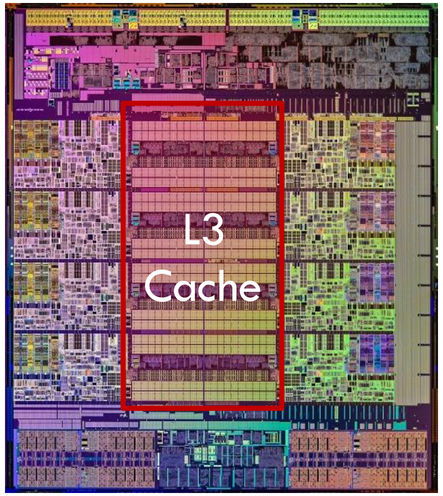
Data compression can be an effective method to achieve higher system performance and energy efficiency in modern data-intensive applications by exploiting redundancy and data similarity. Prior works have studied a variety of data compression techniques to improve both capacity (e.g., of caches and main memory) and bandwidth utilization (e.g., of the on-chip and off-chip interconnects). In this paper, we make a new observation about the energy-efficiency of communication when compression is applied. While compression reduces the amount of transferred data, it leads to a substantial increase in the number of bit toggles (i.e., communication channel switchings from 0 to 1 or from 1 to 0). The increased toggle count increases the dynamic energy consumed by on-chip and off-chip buses due to more frequent charging and discharging of the wires. Our

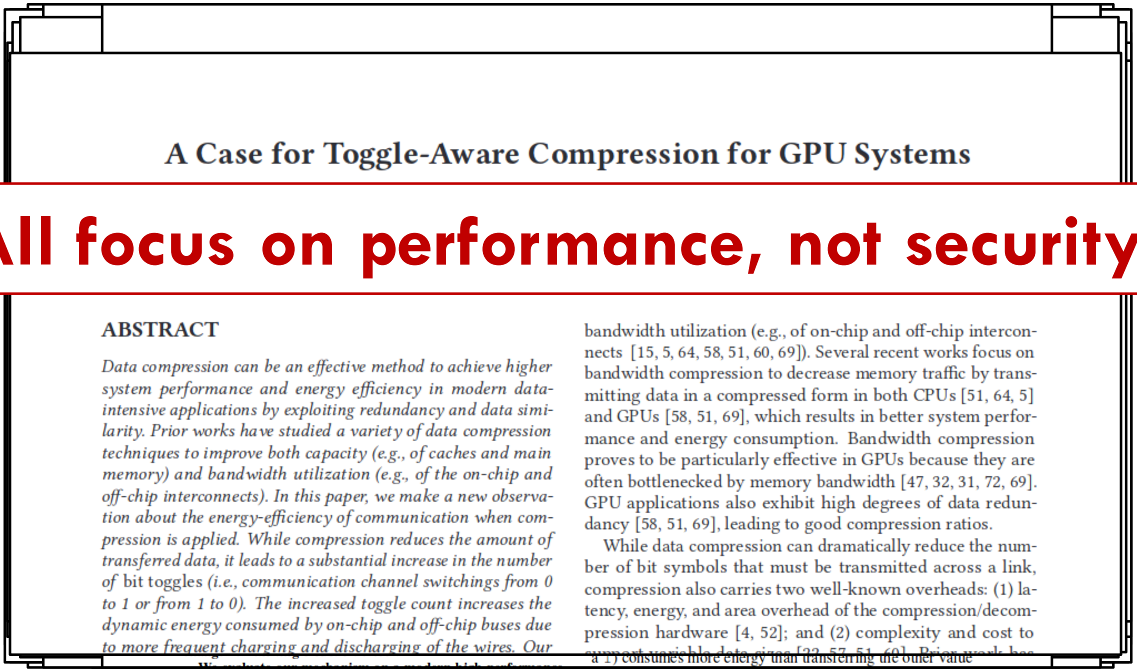
bandwidth utilization (e.g., of on-chip and off-chip interconnects [15, 5, 64, 58, 51, 60, 69]). Several recent works focus on bandwidth compression to decrease memory traffic by transmitting data in a compressed form in both CPUs [51, 64, 5] and GPUs [58, 51, 69], which results in better system performance and energy consumption. Bandwidth compression proves to be particularly effective in GPUs because they are often bottlenecked by memory bandwidth [47, 32, 31, 72, 69]. GPU applications also exhibit high degrees of data redundancy [58, 51, 69], leading to good compression ratios.

While data compression can dramatically reduce the number of bit symbols that must be transmitted across a link, compression also carries two well-known overheads: (1) latency, energy, and area overhead of the compression/decompression hardware [4, 52]; and (2) complexity and cost to

Cache Compression Tradeoffs

- Higher effective capacity → Higher hit rate
- Somewhat higher hit latency
- Highly beneficial for large caches (e.g., LLC)
- Intense research activity over past 15 years





A Case for Toggle-Aware Compression for GPU Systems

All focus on performance, not security

ABSTRACT

Data compression can be an effective method to achieve higher system performance and energy efficiency in modern data-intensive applications by exploiting redundancy and data similarity. Prior works have studied a variety of data compression techniques to improve both capacity (e.g., of caches and main memory) and bandwidth utilization (e.g., of the on-chip and off-chip interconnects). In this paper, we make a new observation about the energy-efficiency of communication when compression is applied. While compression reduces the amount of transferred data, it leads to a substantial increase in the number of bit toggles (i.e., communication channel switchings from 0 to 1 or from 1 to 0). The increased toggle count increases the dynamic energy consumed by on-chip and off-chip buses due to more frequent charging and discharging of the wires. Our

bandwidth utilization (e.g., of on-chip and off-chip interconnects [15, 5, 64, 58, 51, 60, 69]). Several recent works focus on bandwidth compression to decrease memory traffic by transmitting data in a compressed form in both CPUs [51, 64, 5] and GPUs [58, 51, 69], which results in better system performance and energy consumption. Bandwidth compression proves to be particularly effective in GPUs because they are often bottlenecked by memory bandwidth [47, 32, 31, 72, 69]. GPU applications also exhibit high degrees of data redundancy [58, 51, 69], leading to good compression ratios.

While data compression can dramatically reduce the number of bit symbols that must be transmitted across a link, compression also carries two well-known overheads: (1) latency, energy, and area overhead of the compression/decompression hardware [4, 52]; and (2) complexity and cost to

Cache Compression Ingredients

Cache Compression Ingredients

- Architecture: How to locate and manage variable-sized compressed blocks?

Cache Compression Ingredients

- Architecture: How to locate and manage variable-sized compressed blocks?

- Algorithm: How to compress each cache block?

Cache Compression Ingredients

- Architecture: How to locate and manage variable-sized compressed blocks?

- Algorithm: How to compress each cache block?

- We focus attacks on a commonly used baseline:
  - VSC compressed cache architecture
  - BDI compression algorithm

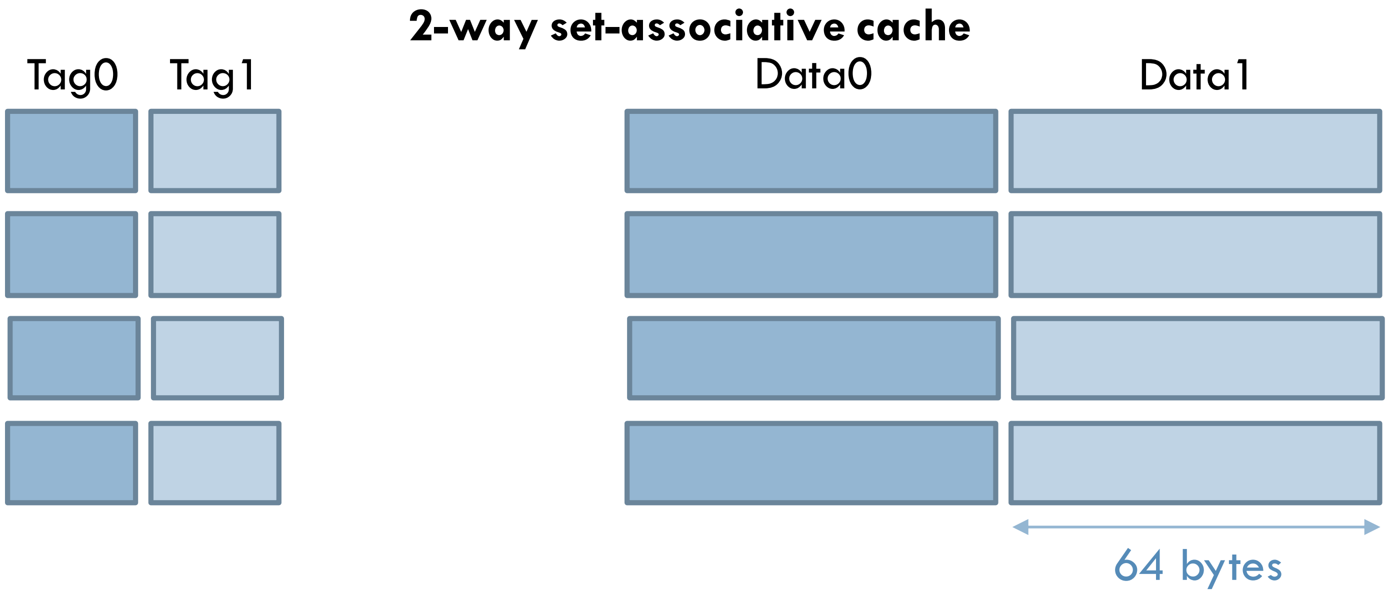
Cache Compression Ingredients

- Architecture: How to locate and manage variable-sized compressed blocks?

- Algorithm: How to compress each cache block?

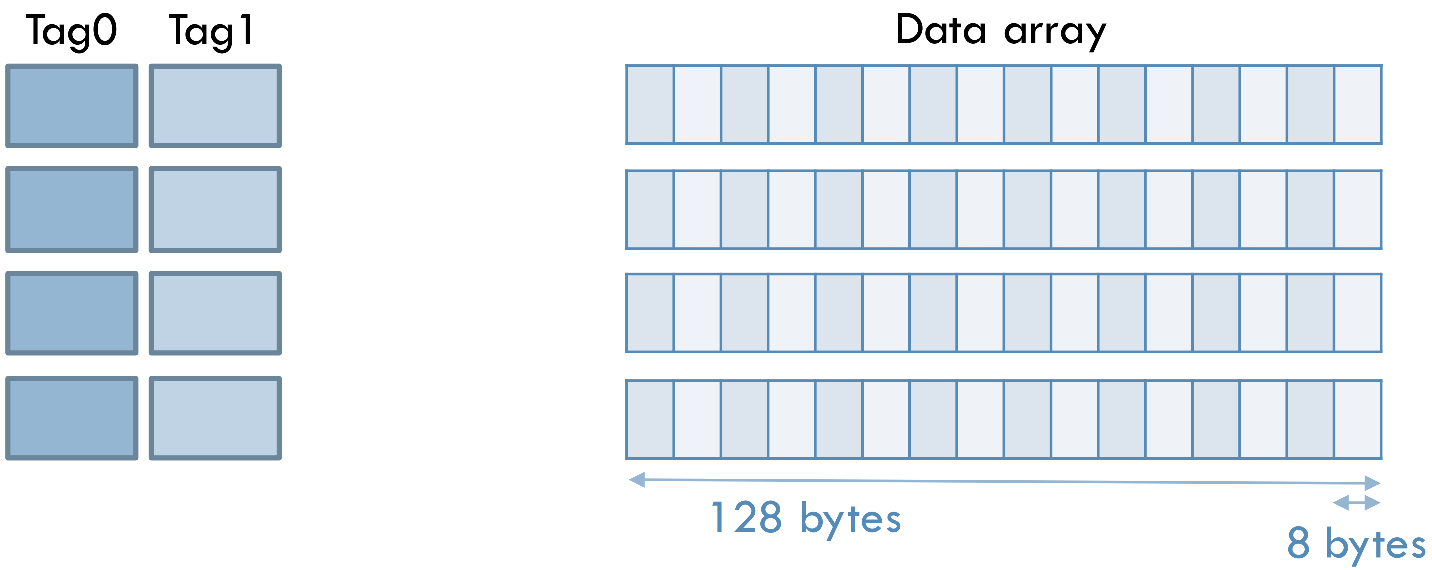
- We focus attacks on a commonly used baseline:
  - VSC compressed cache architecture
  - BDI compression algorithm
- Attacks apply to other architectures & algorithms
  - Leads to different characteristics about leaked data

VSC [Alameldeen and Wood ISCA'04]



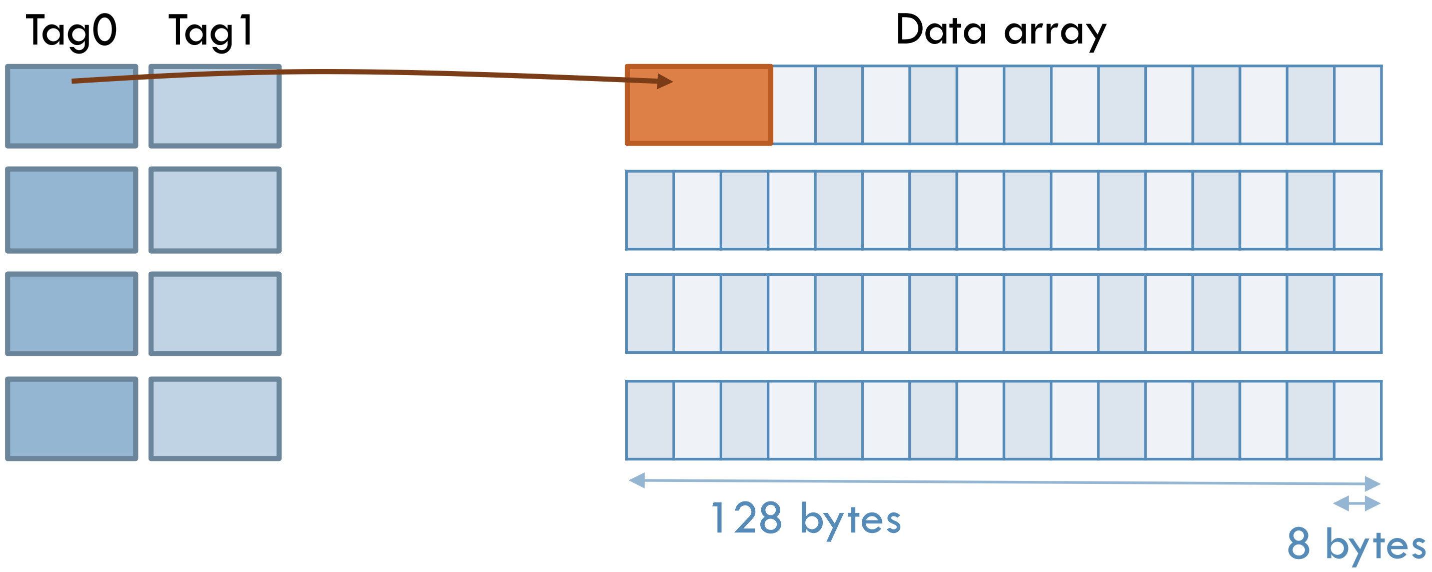
- Conventional caches can only manage fixed-size blocks

VSC [Alameldeen and Wood ISCA'04]



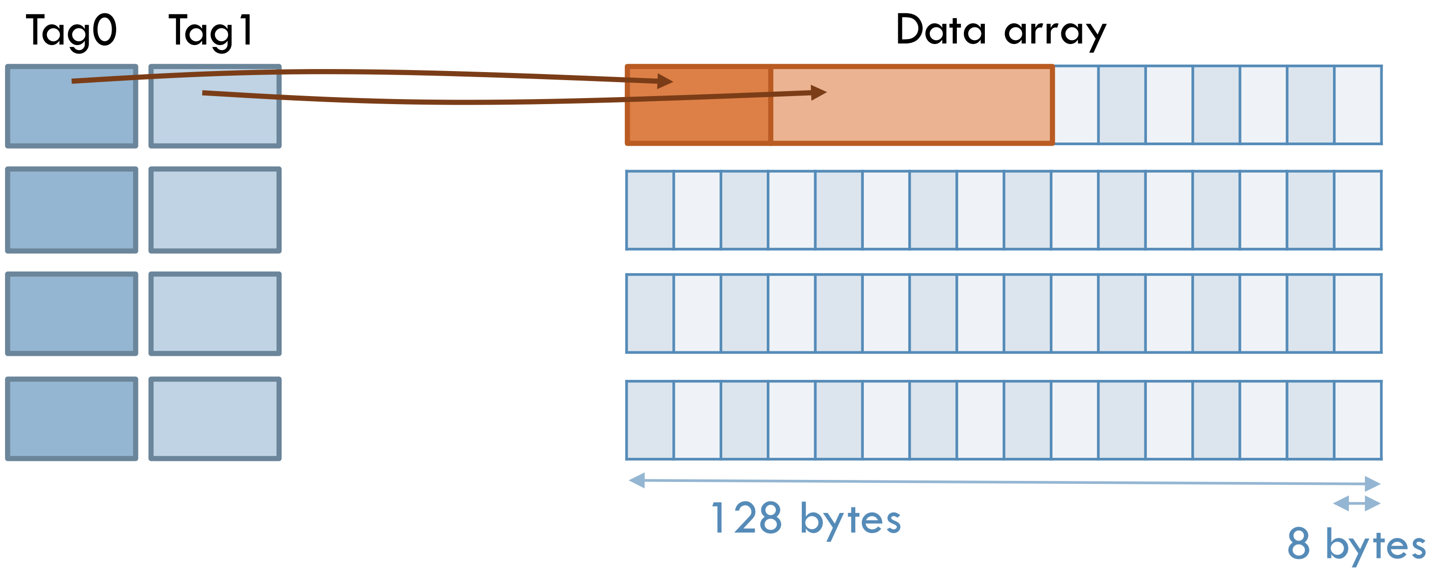
- VSC divides data array into small segments and lets compressed lines take a variable number of segments

VSC [Alameldeen and Wood ISCA'04]



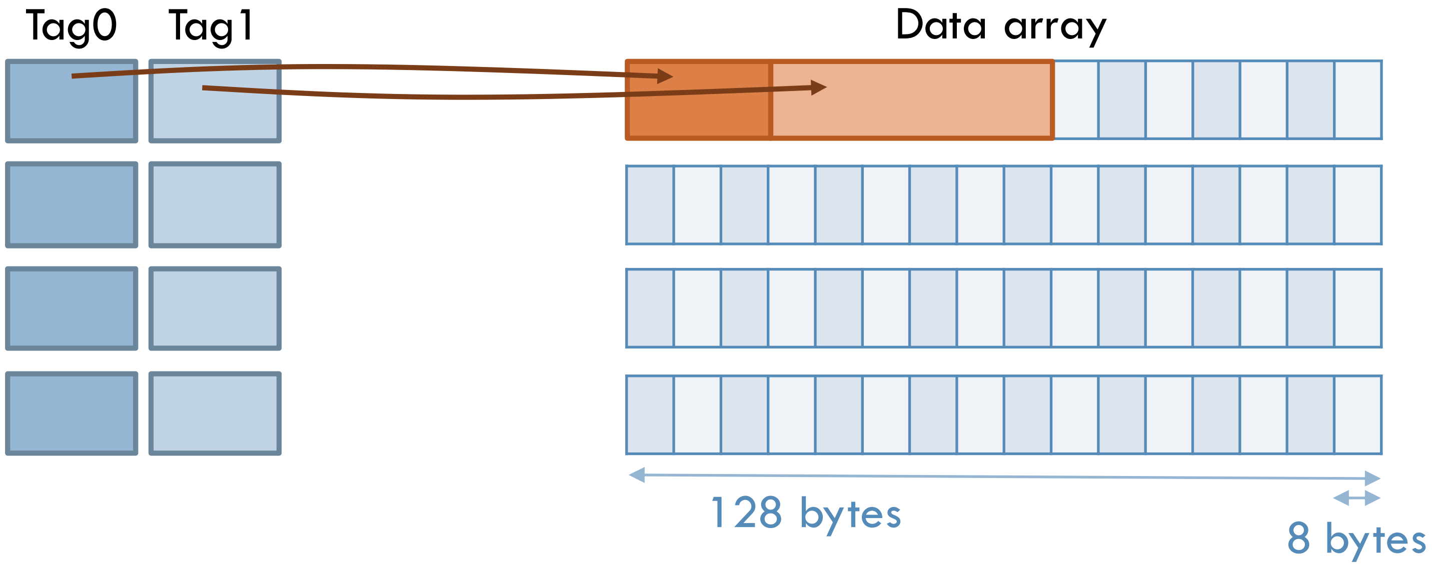
- VSC divides data array into small segments and lets compressed lines take a variable number of segments

VSC [Alameldeen and Wood ISCA'04]



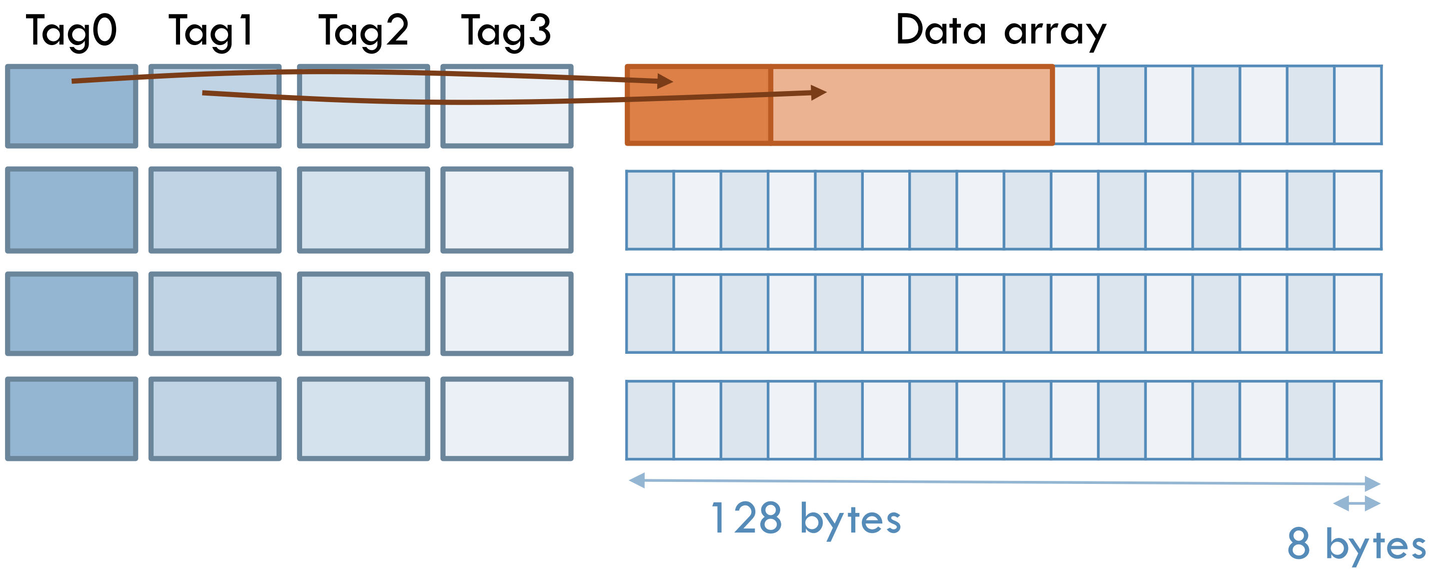
- VSC divides data array into small segments and lets compressed lines take a variable number of segments

VSC [Alameldeen and Wood ISCA'04]



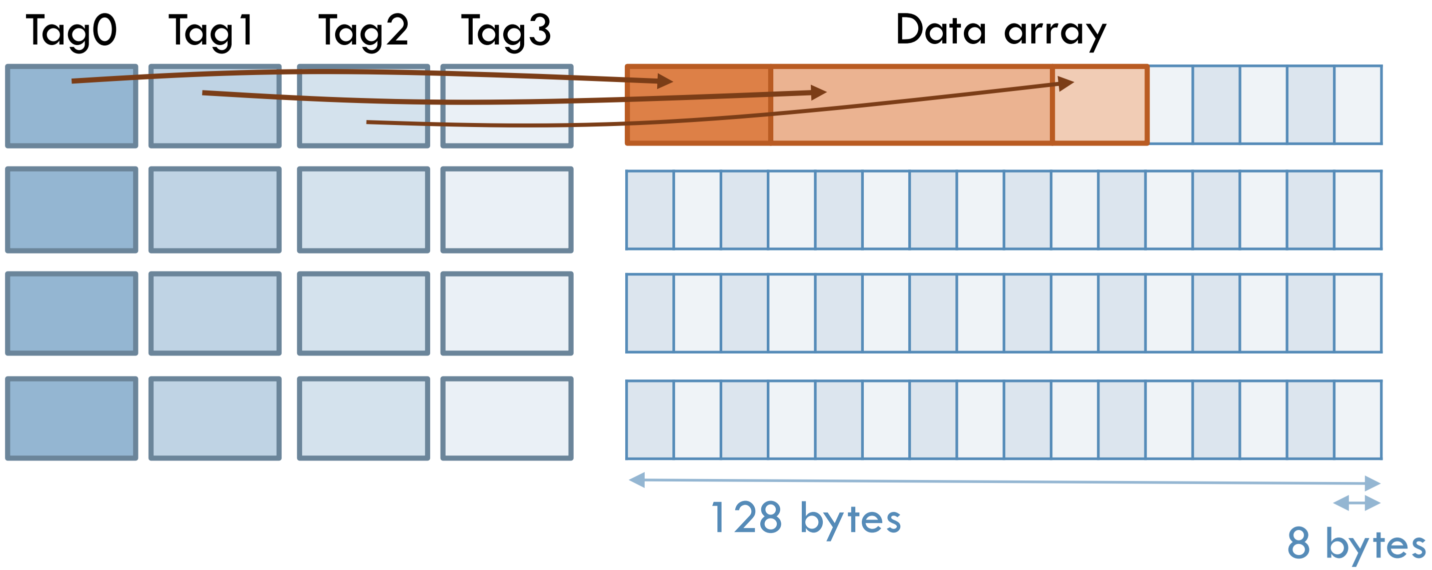
- VSC divides data array into small segments and lets compressed lines take a variable number of segments
- VSC increases tags relative to uncompressed caches to track more compressed lines per set

VSC [Alameldeen and Wood ISCA'04]



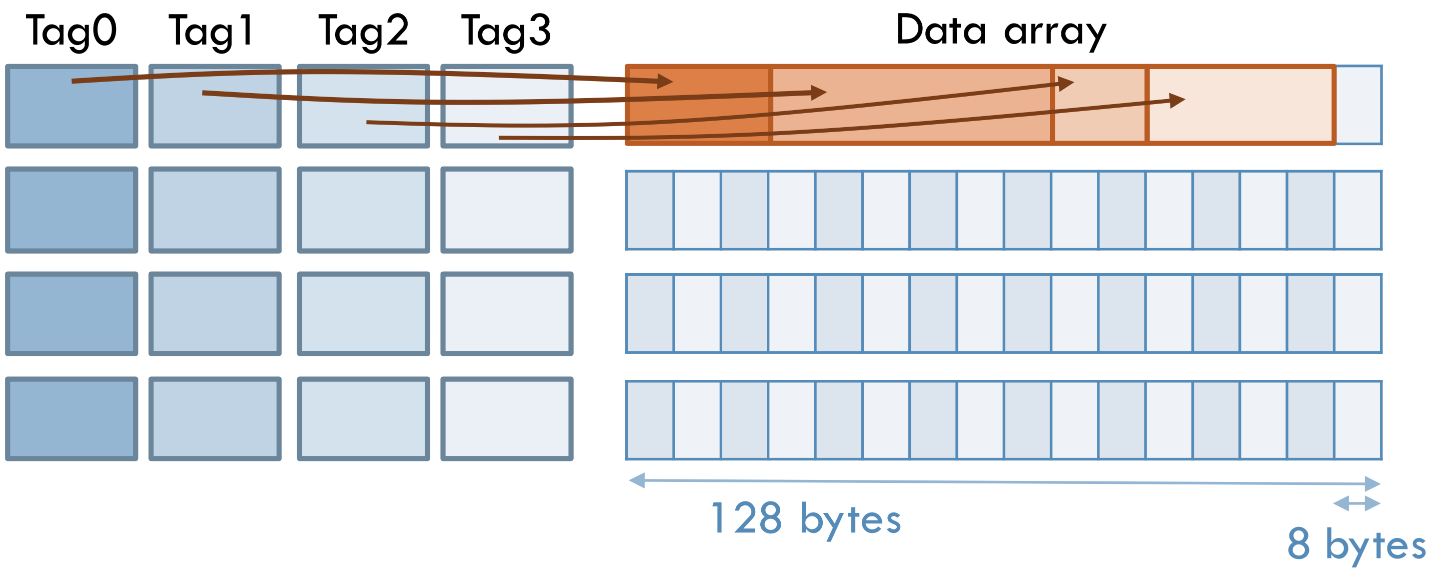
- VSC divides data array into small segments and lets compressed lines take a variable number of segments
- VSC increases tags relative to uncompressed caches to track more compressed lines per set

VSC [Alameldeen and Wood ISCA'04]



- VSC divides data array into small segments and lets compressed lines take a variable number of segments
- VSC increases tags relative to uncompressed caches to track more compressed lines per set

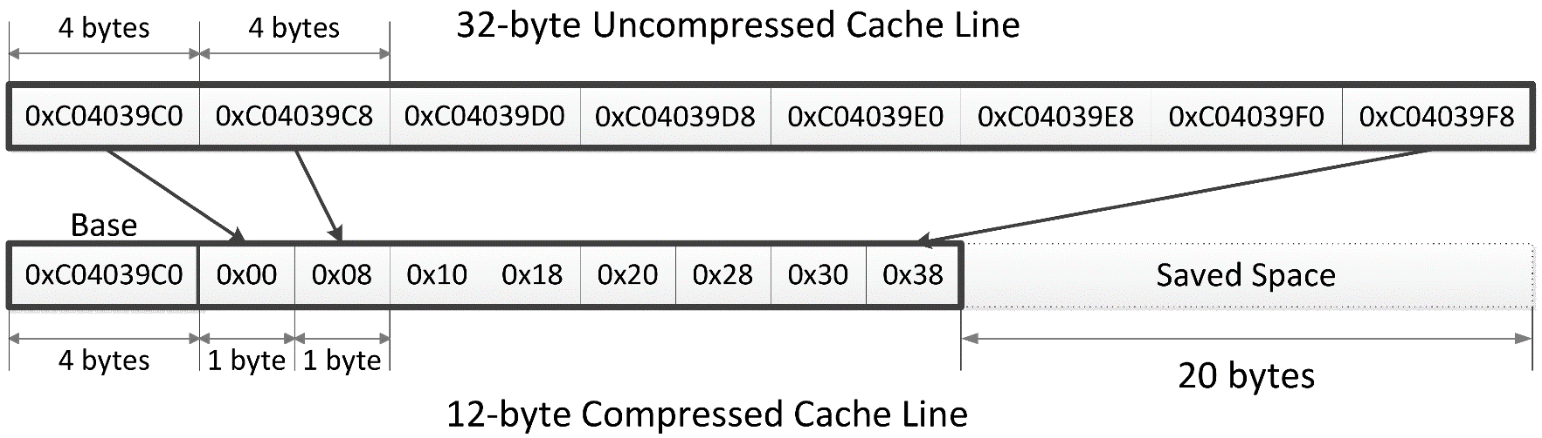
VSC [Alameldeen and Wood ISCA'04]



- VSC divides data array into small segments and lets compressed lines take a variable number of segments
- VSC increases tags relative to uncompressed caches to track more compressed lines per set

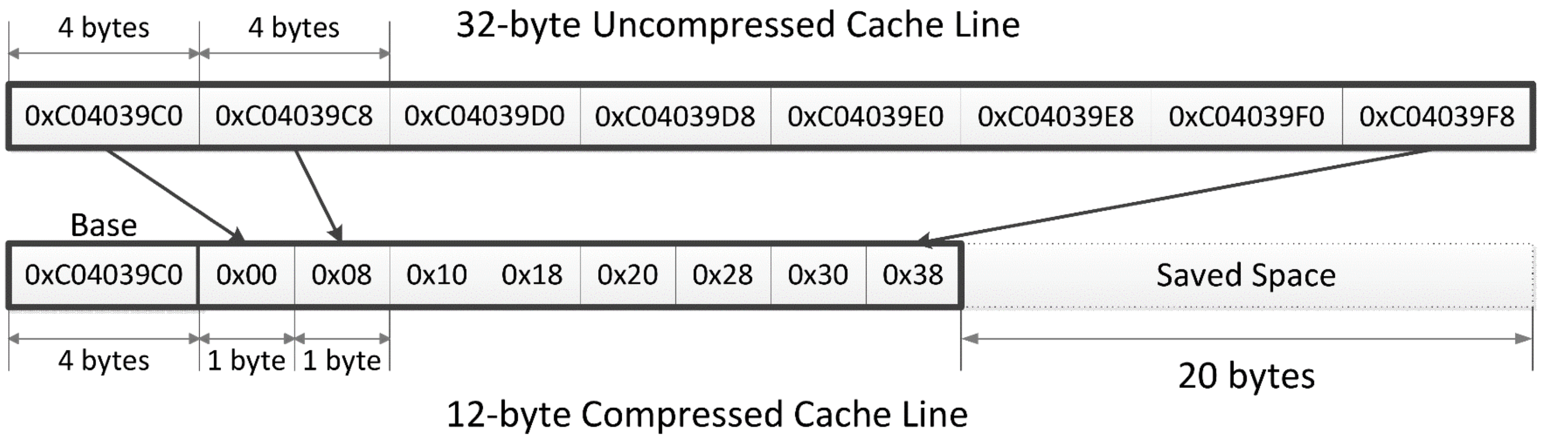
BDI [Pekhimenko et al. PACT'12]

- Base-Delta-Immediate (BDI) compresses lines with similar values by using a common base + small deltas



BDI [Pekhimenko et al. PACT'12]

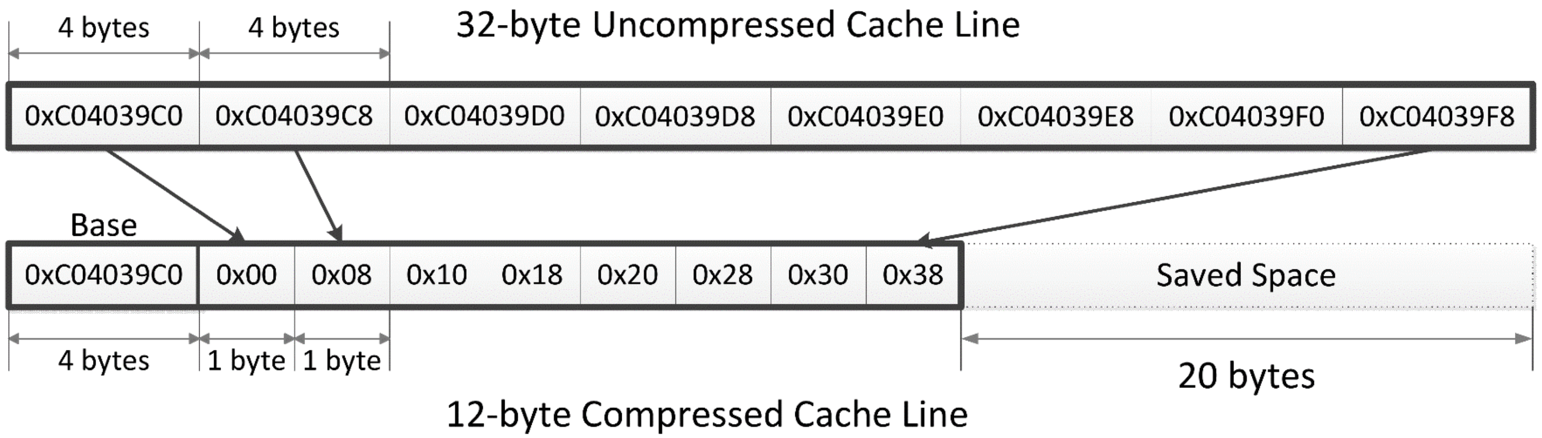
- Base-Delta-Immediate (BDI) compresses lines with similar values by using a common base + small deltas



- BDI supports multiple formats with different base sizes (2, 4, 8 bytes) and delta sizes (1, 2, 4 bytes)

BDI [Pekhimenko et al. PACT‘12]

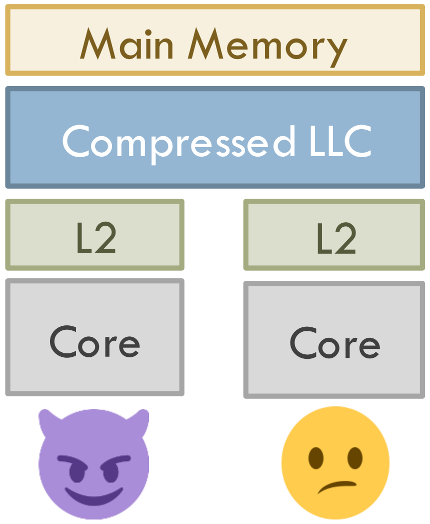
- Base-Delta-Immediate (BDI) compresses lines with similar values by using a common base + small deltas



- BDI supports multiple formats with different base sizes (2, 4, 8 bytes) and delta sizes (1, 2, 4 bytes)
- Reasonable compression ratio, simple implementation

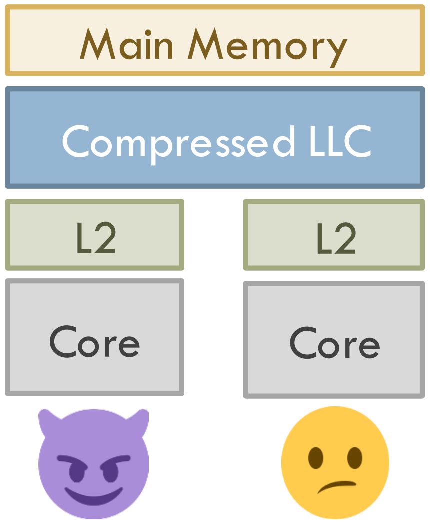
Pack+Probe: Measuring Compressibility

- Threat model:
  - Attacker and victim run in different protection domains (processes, VMs, etc.)
  - Attacker and victim share compressed cache
  - Attacker knows compressed cache architecture & algorithm used
  - Attacker knows set of victim's target line (can use standard techniques to find it)



Pack+Probe: Measuring Compressibility

- Threat model:
  - Attacker and victim run in different protection domains (processes, VMs, etc.)
  - Attacker and victim share compressed cache
  - Attacker knows compressed cache architecture & algorithm used
  - Attacker knows set of victim's target line (can use standard techniques to find it)



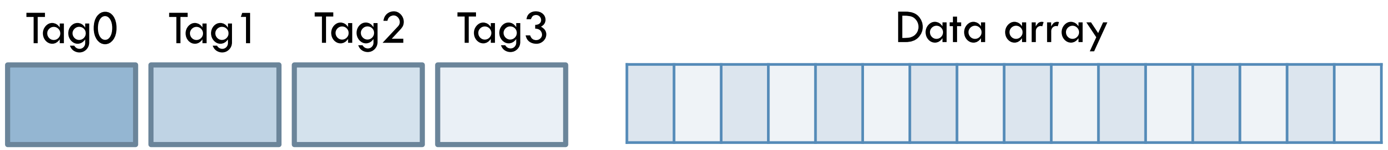
- Goal: Find compressed size of target line

Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag

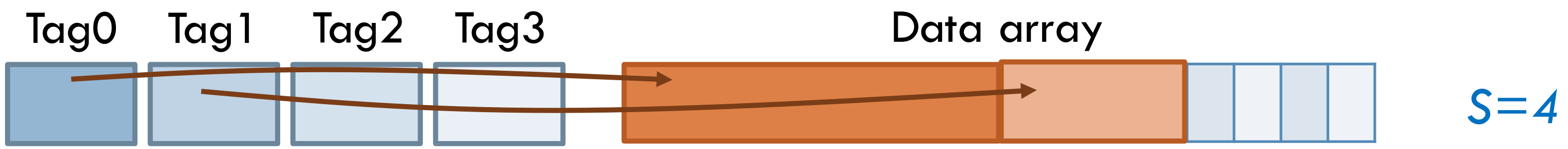
Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag



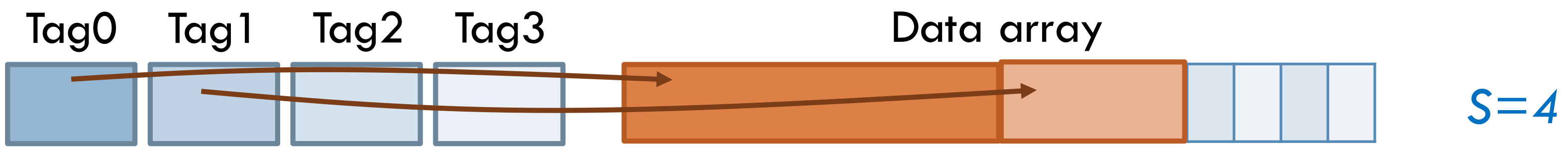
Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag



Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag

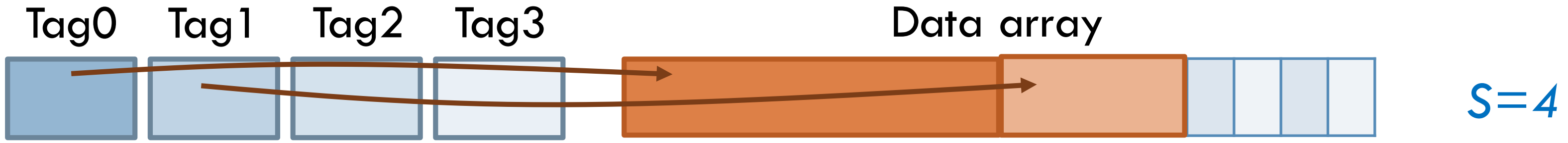


After victim accesses target set, attacker **probes** all lines used to pack target set

- All hits → Victim line ≤ S segments
- Any miss → Victim line > S segments

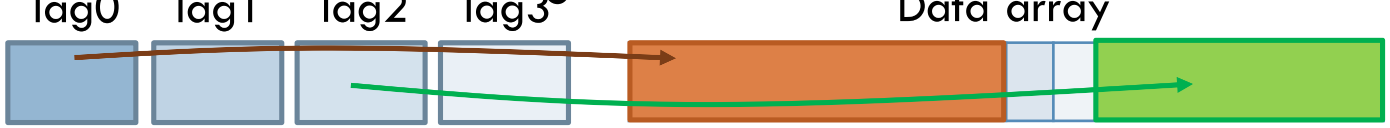
Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag



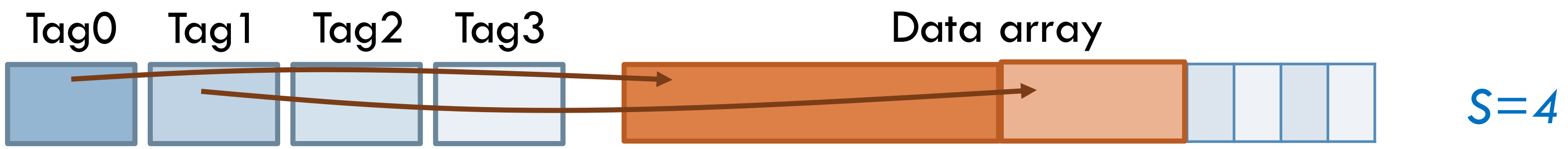
After victim accesses target set, attacker **probes** all lines used to pack target set

- All hits → Victim line ≤ S segments
- Any miss → Victim line > S segments



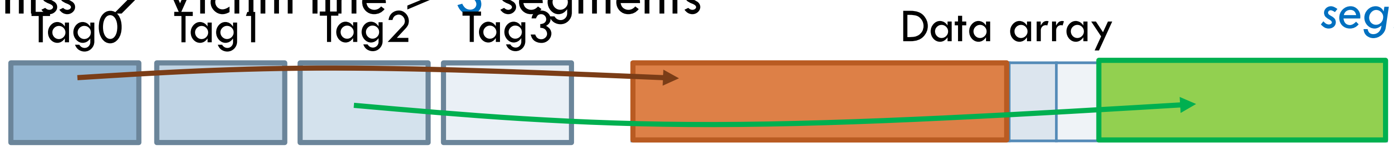
Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving *S* free segments and at least one free tag



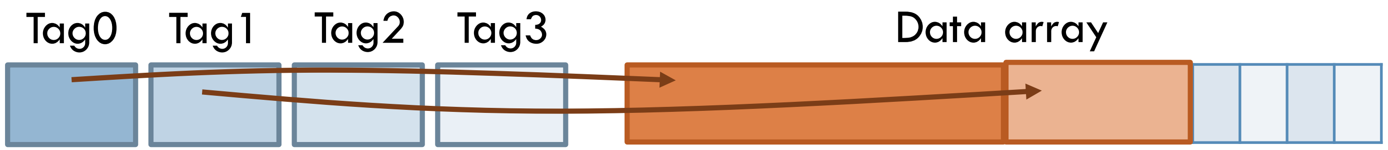
After victim accesses target set, attacker **probes** all lines used to pack target set

- All hits → Victim line ≤ S segments
- Any miss → Victim line > S segments



Pack+Probe: Measuring Compressibility

Attacker **packs** target set with lines of known sizes, leaving S free segments and at least one free tag

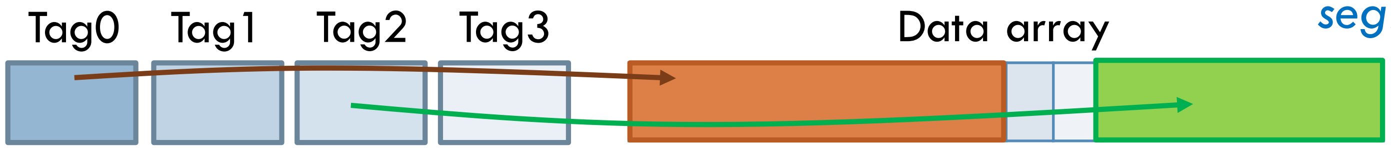


S=4

After victim accesses target set, attacker **probes** all lines used to pack target set

- All hits → Victim line ≤ S segments
- Any miss → Victim line > S segments

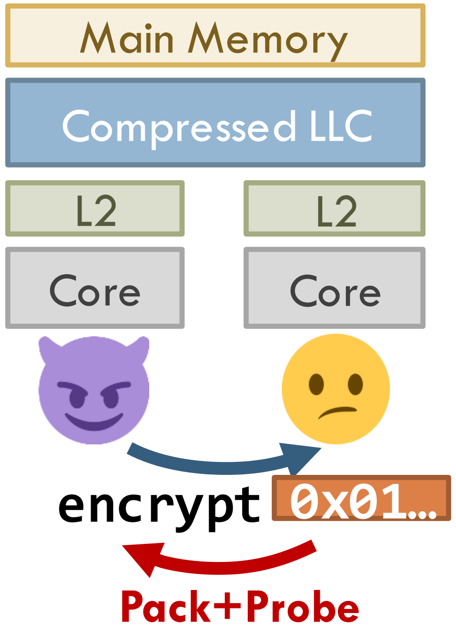
Miss → Victim > 4 segments



By doing a binary search over S, one can find exact size in log2(MaxSegmentsPerCacheLine) measurements

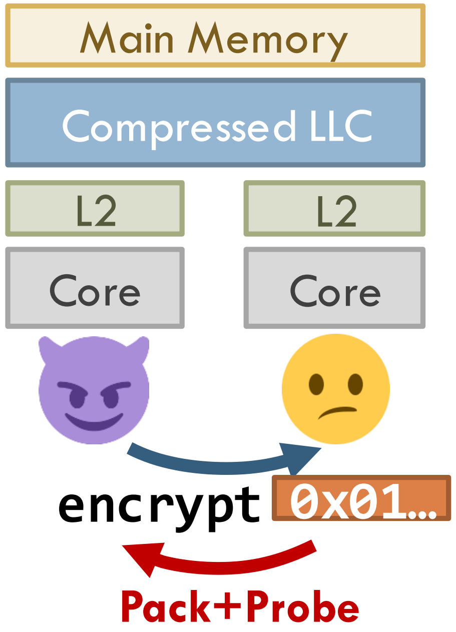
Safecracker: Exploiting Data Colocation to Leak Secrets

- Threat model:
  - Attacker and victim run in different domains, share compressed cache (as in Pack+Probe)
  - Attacker can get victim to collocate attacker-controlled data near victim's own secret data
- Goal: Leak victim's data



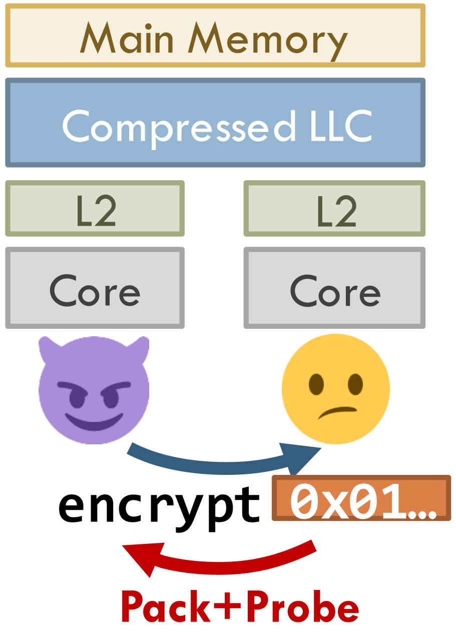
Safecracker: Exploiting Data Colocation to Leak Secrets

- Threat model:
  - Attacker and victim run in different domains, share compressed cache (as in Pack+Probe)
  - Attacker can get victim to collocate attacker-controlled data near victim's own secret data
- Goal: Leak victim's data
- Multiple colocation vectors:
  - Victim itself colocates (contiguous allocation, stack spills, etc.)
  - Memory safety violations (buffer overflows, heap spraying, etc.)



Safecracker: Exploiting Data Colocation to Leak Secrets

- Threat model:
  - Attacker and victim run in different domains, share compressed cache (as in Pack+Probe)
  - Attacker can get victim to collocate attacker-controlled data near victim's own secret data
- Goal: Leak victim's data
- Multiple colocation vectors:
  - Victim itself colocates (contiguous allocation, stack spills, etc.)
  - Memory safety violations (buffer overflows, heap spraying, etc.)
- Safecracker changes attacker-controlled data to reveal nearby secret data through changes in compressibility
  - Search strategy depends on compression algorithm



Safecracker on BDI

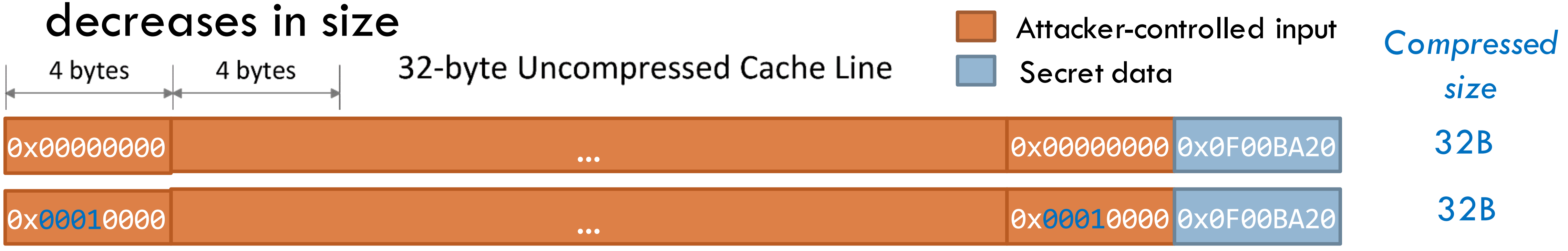
- Starting from largest delta, sweep high-order bytes until target line decreases in size

Safecracker on BDI

- Starting from largest delta, sweep high-order bytes until target line decreases in size

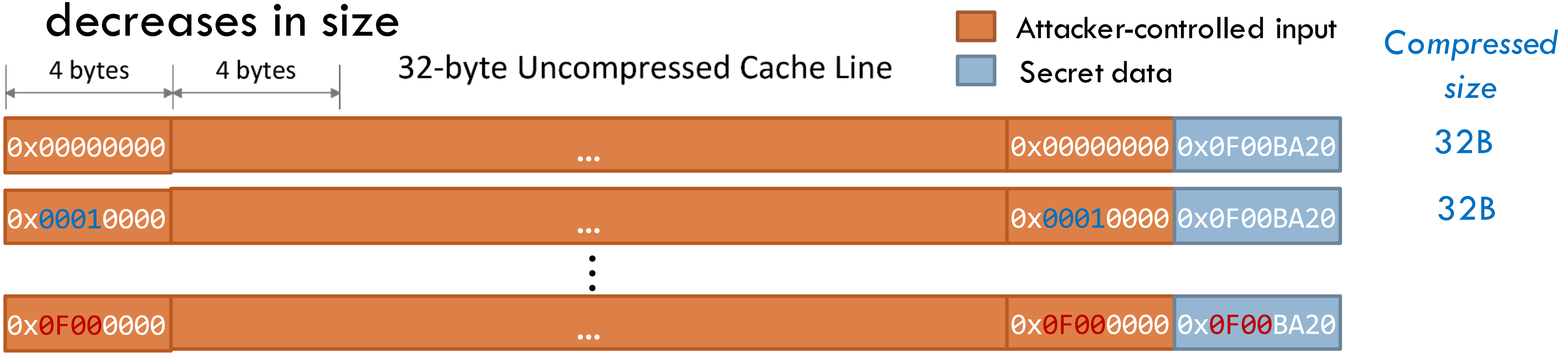
Safecracker on BDI

- Starting from largest delta, sweep high-order bytes until target line decreases in size



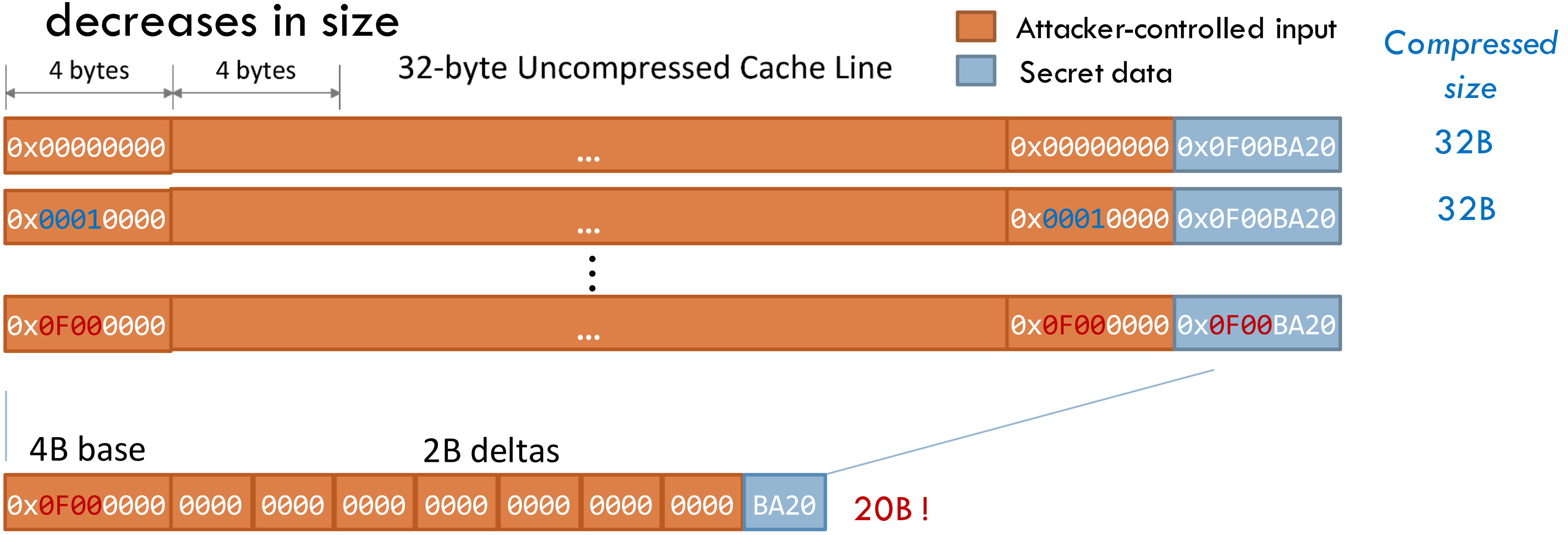
Safecracker on BDI

- Starting from largest delta, sweep high-order bytes until target line decreases in size



Safecracker on BDI

- Starting from largest delta, sweep high-order bytes until target line decreases in size

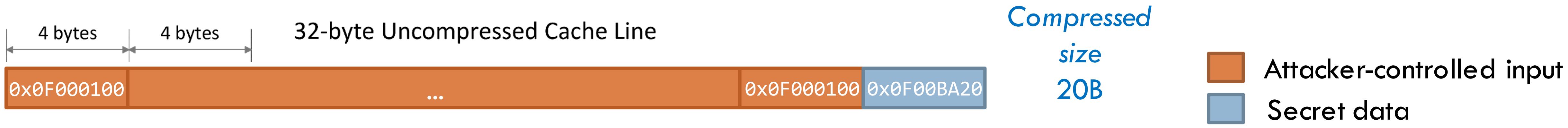


Safecracker on BDI

- Continue sweeping lower-order bytes until recovering all bytes

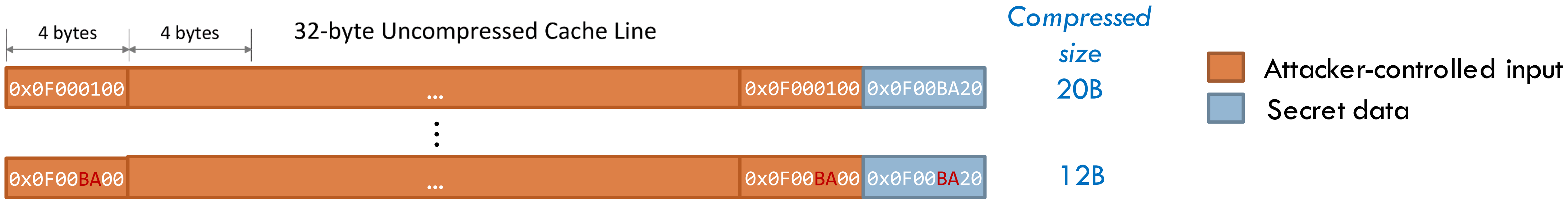
Safecracker on BDI

- Continue sweeping lower-order bytes until recovering all bytes



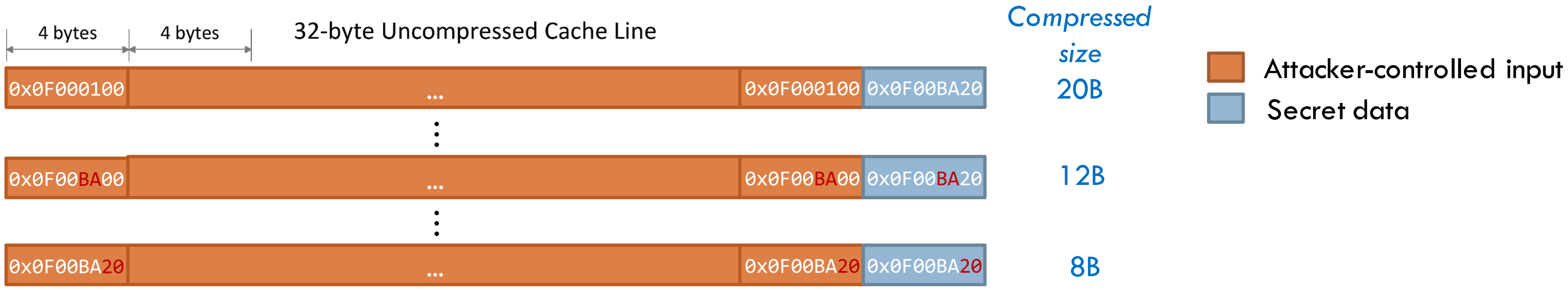
Safecracker on BDI

- Continue sweeping lower-order bytes until recovering all bytes



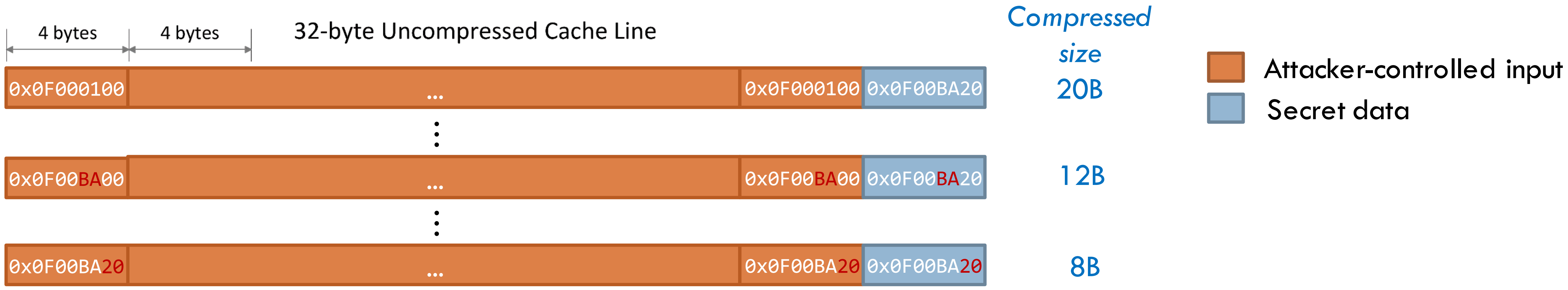
Safecracker on BDI

- Continue sweeping lower-order bytes until recovering all bytes



Safecracker on BDI

- Continue sweeping lower-order bytes until recovering all bytes

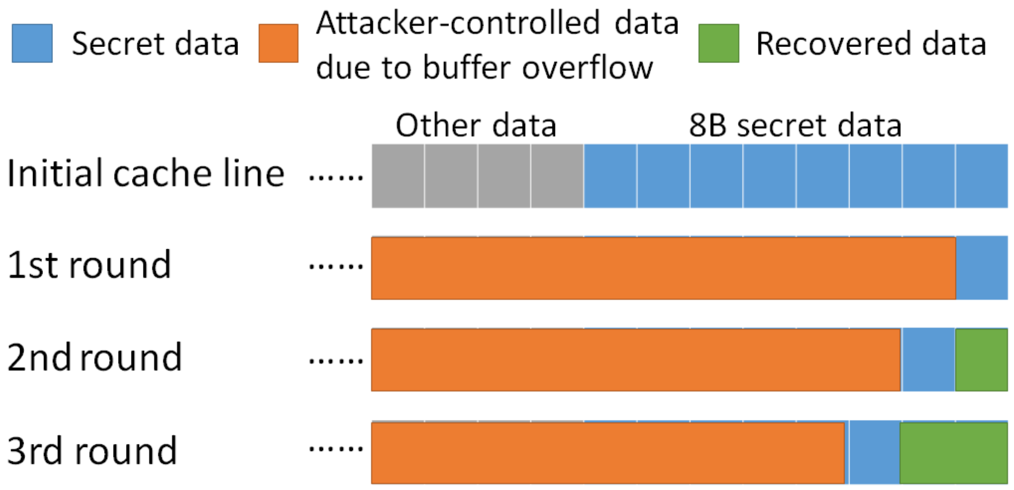


- BDI allows recovering up to 8 bytes this way

Secret Size	Compression Format Sequence	Attempts
2B	NoComp→B2D1→B8D0	$O(2^8)$
4B	NoComp→B4D2→B4D1→B8D0	$O(2^{16})$
8B	NoComp→B8D4→B8D2→B8D1→B8D0	$O(2^{32})$

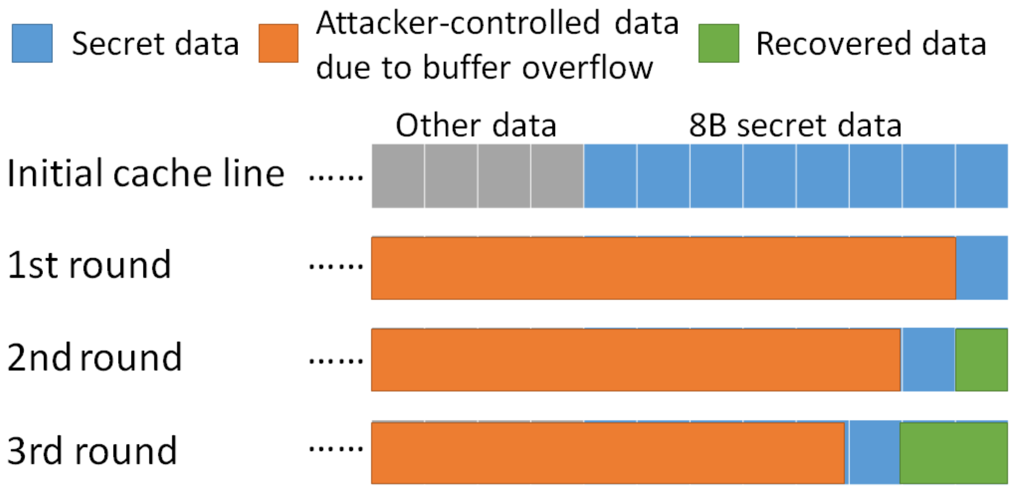
Enhancing Safecracker w/ buffer overflows

- Buffer overflows let Safecracker control where attacker-controlled data is located
  - Makes search more efficient
  - Can leak data far away from buffer



Enhancing Safecracker w/ buffer overflows

- Buffer overflows let Safecracker control where attacker-controlled data is located
  - Makes search more efficient
  - Can leak data far away from buffer

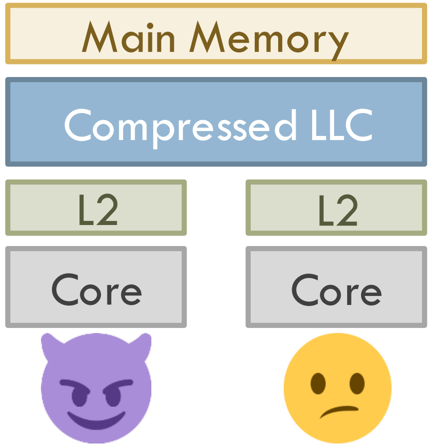


- With BDI, can leak $1/8^{th}$ of victim's memory!
  - Other compression algorithms (e.g., RLE) allow more leakage

Safecracker Evaluation

- Microarchitectural simulation using zsim

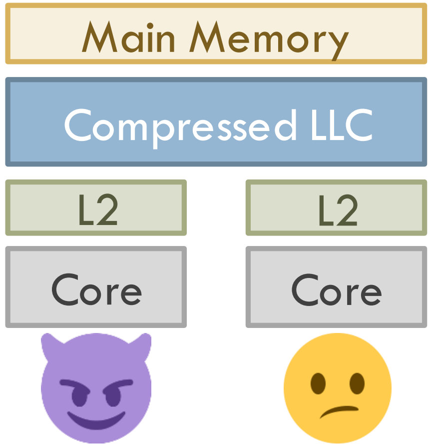
- Multicore system modeled after Skylake



8MB VSC with 64-byte lines,
2x tag array, 32 tags/set
BDI compression

Safecracker Evaluation

- Microarchitectural simulation using zsim

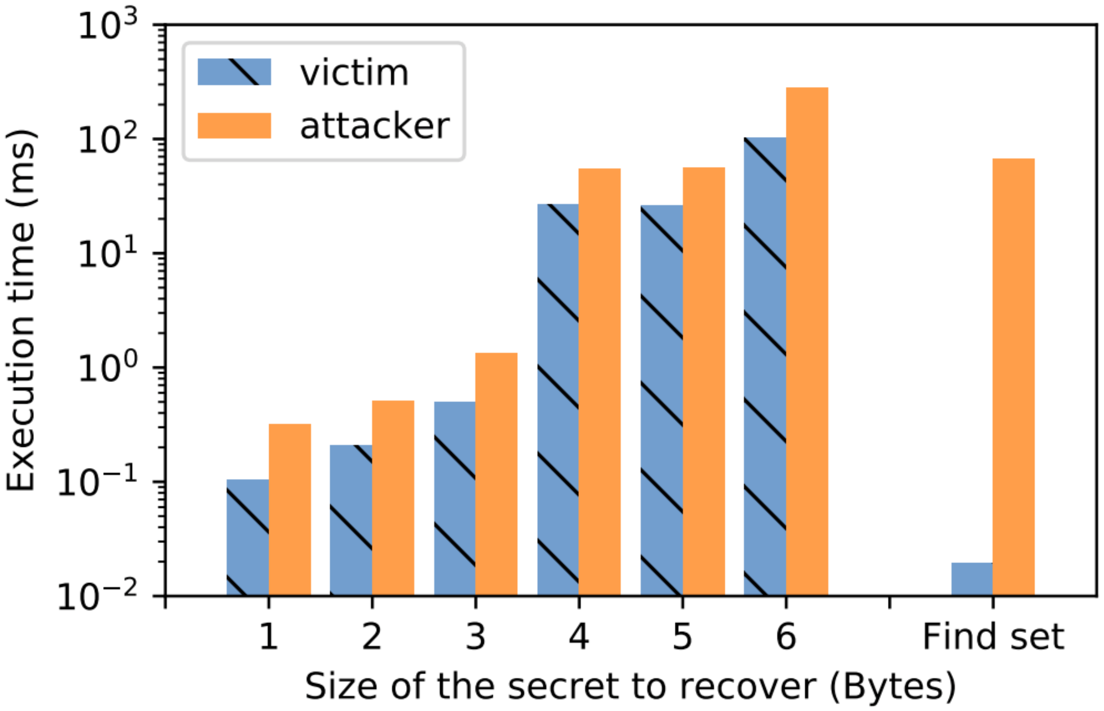


8MB VSC with 64-byte lines,
2x tag array, 32 tags/set
BDI compression

- Multicore system modeled after Skylake
- Two Proof-of-Concept (PoC) workloads:
  - Login server that colocates key and attacker data
  - Server with buffer overflow + key elsewhere in stack

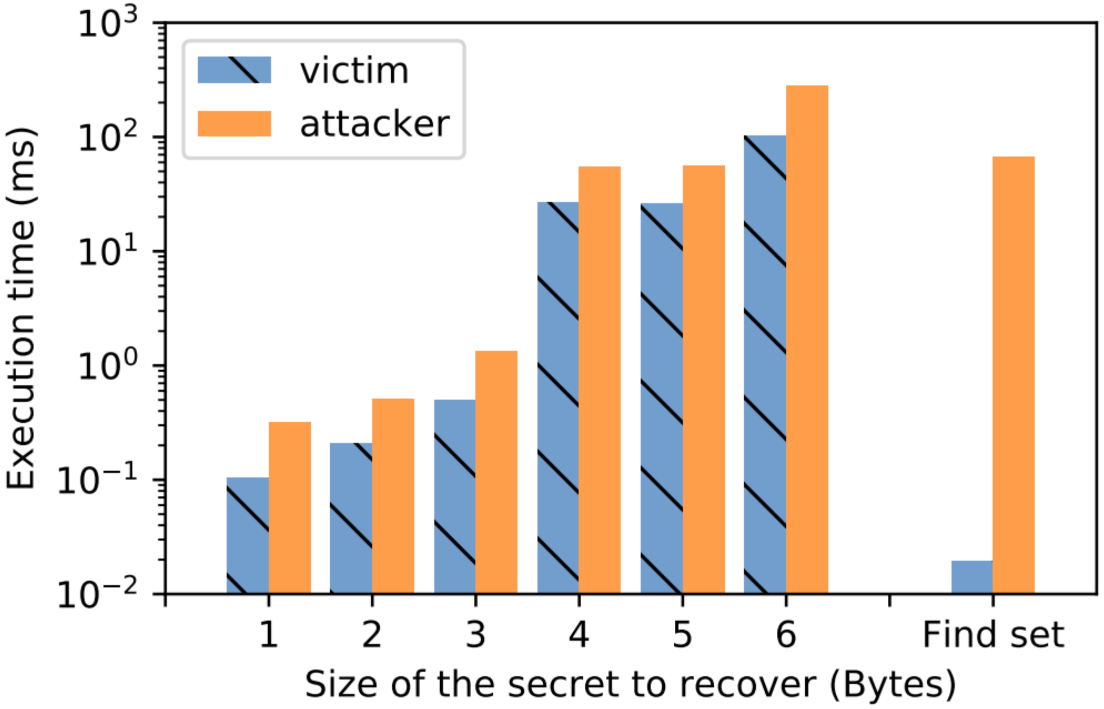
Safecracker steals secrets quickly

PoC 1: Fixed colocation



Safecracker steals secrets quickly

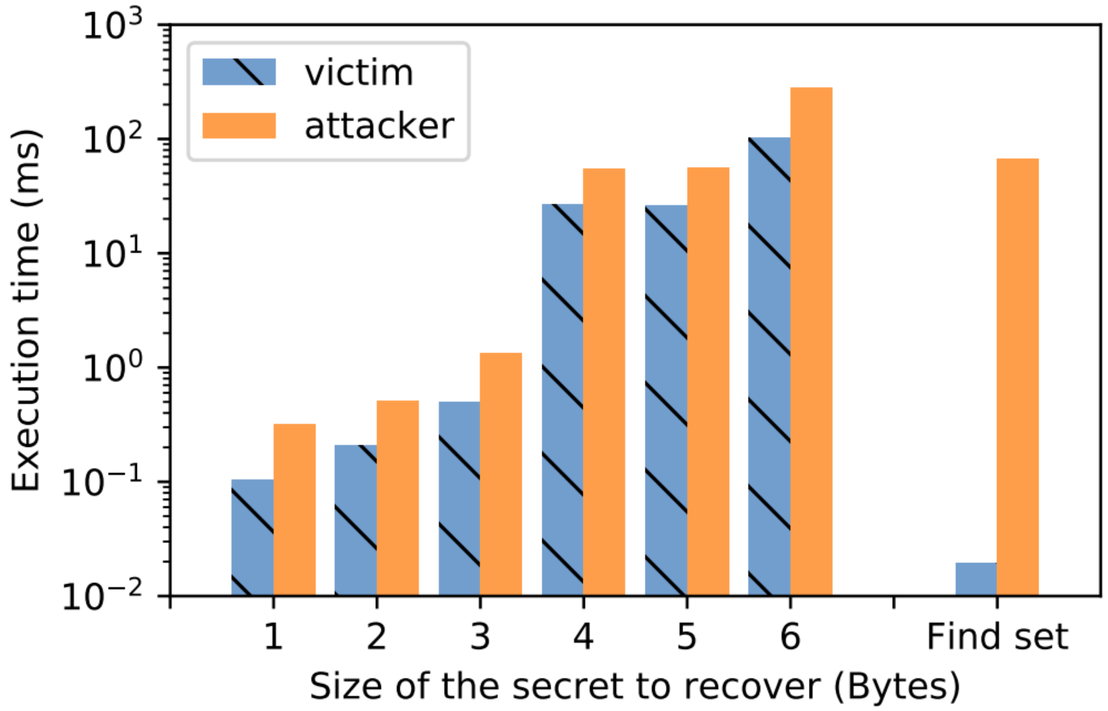
PoC 1: Fixed colocation



Leaks 4B in under 100ms, 6B in 200ms
(comparable to time spent finding target set)

Safecracker steals secrets quickly

PoC 1: Fixed colocation

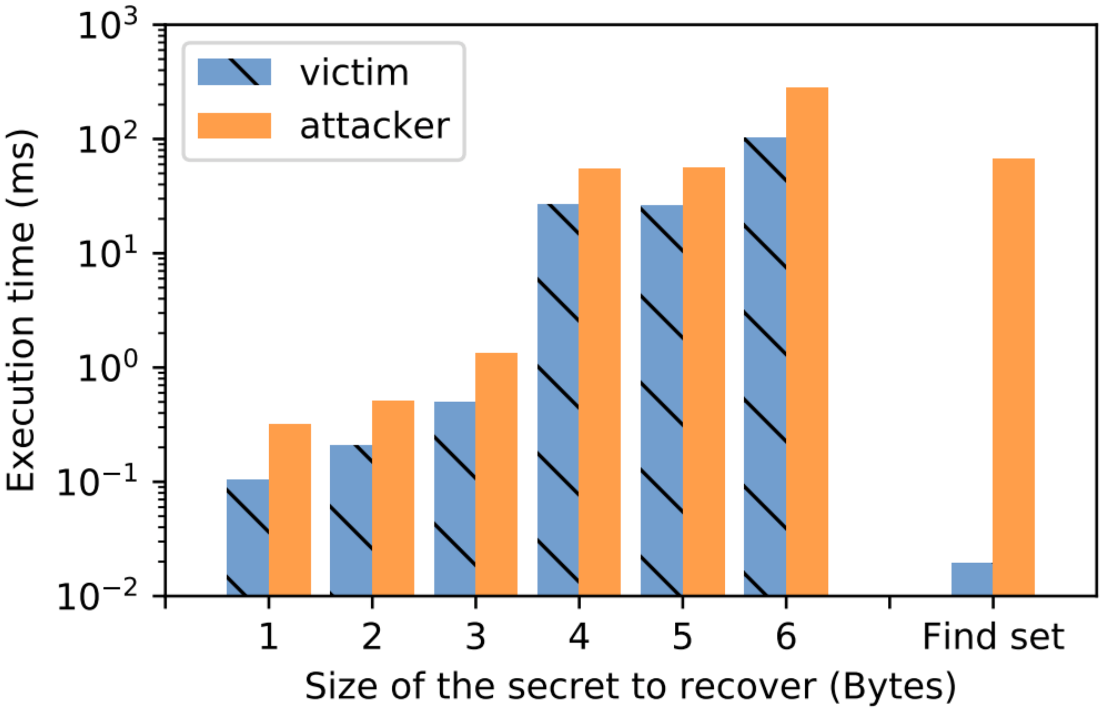


Leaks 4B in under 100ms, 6B in 200ms (comparable to time spent finding target set)

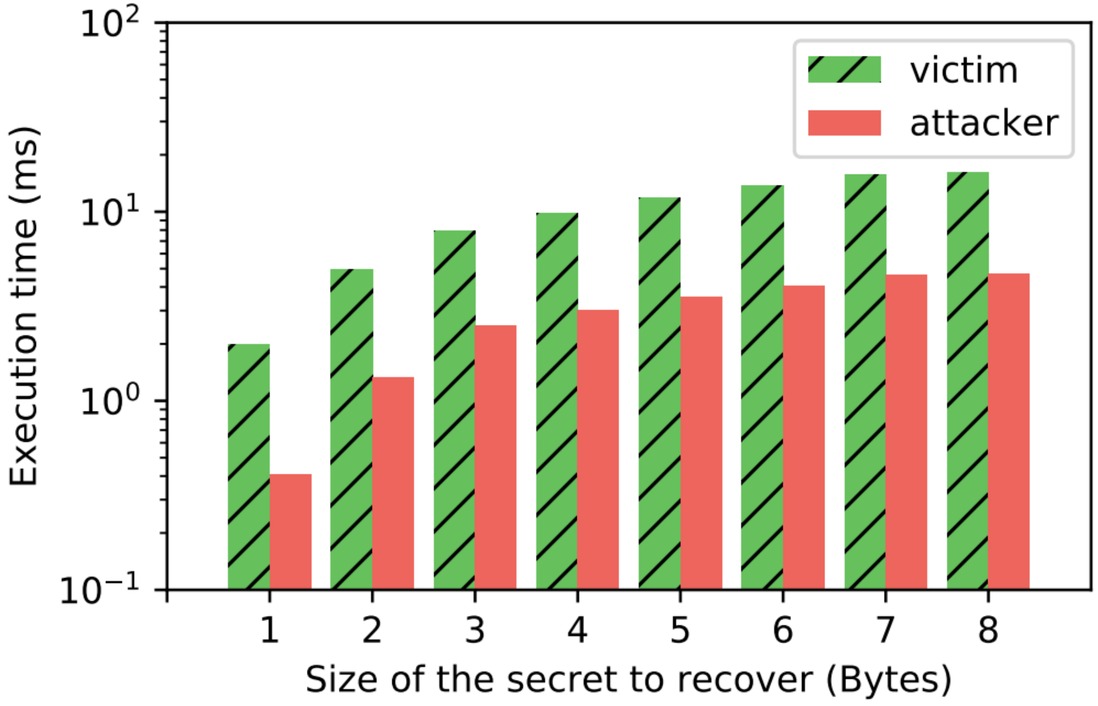
8B would take much longer (~90 hours)

Safecracker steals secrets quickly

PoC 1: Fixed colocation



PoC 2: Buffer overflow

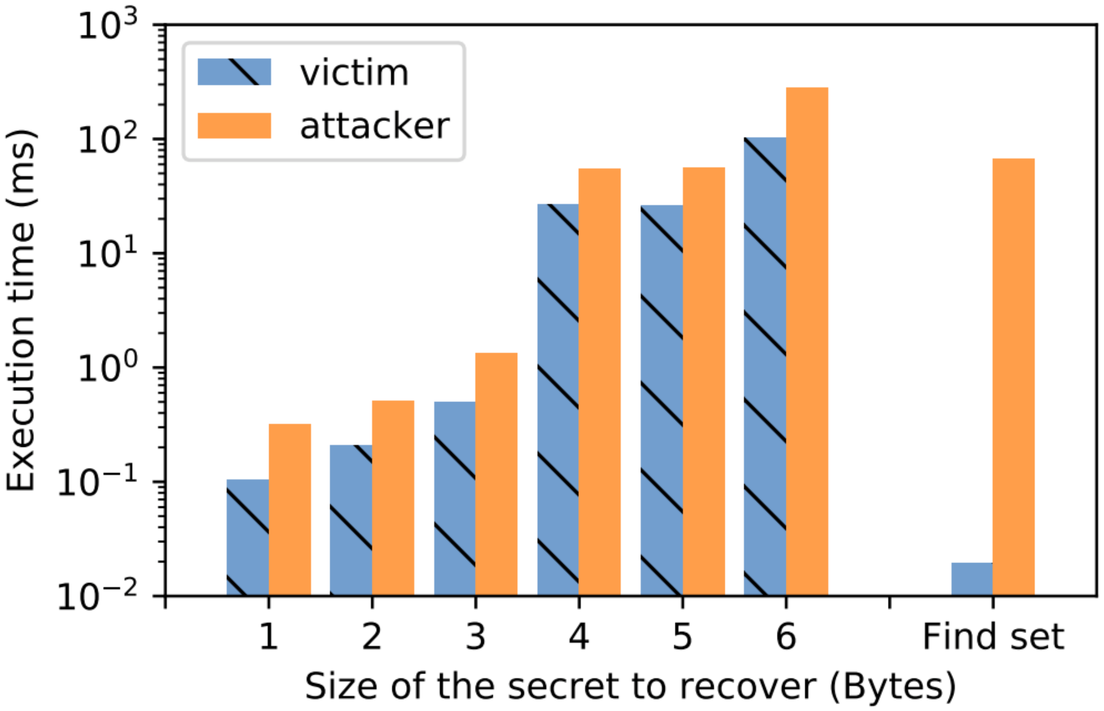


Leaks 4B in under 100ms, 6B in 200ms
(comparable to time spent finding target set)

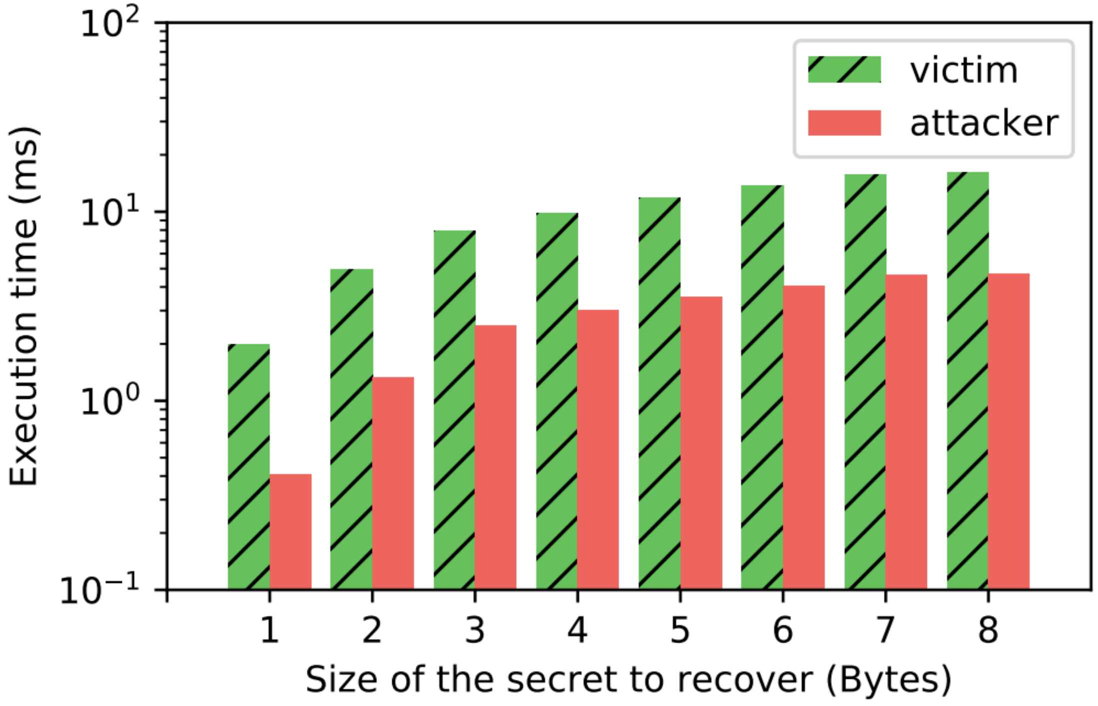
8B would take much longer (~90 hours)

Safecracker steals secrets quickly

PoC 1: Fixed colocation



PoC 2: Buffer overflow



Leaks 4B in under 100ms, 6B in 200ms
(comparable to time spent finding target set)

8B would take much longer (~90 hours)

Leaks 8B in ~10ms

Attack time grows linearly with leaked bytes

Generalizing attacks to other compressed caches

- Most compressed cache architectures allow conflicts among a small set of lines → Pack+Probe still applies

Generalizing attacks to other compressed caches

- Most compressed cache architectures allow conflicts among a small set of lines → Pack+Probe still applies
  - See paper for more discussions

Generalizing attacks to other compressed caches

- Most compressed cache architectures allow conflicts among a small set of lines → Pack+Probe still applies
  - See paper for more discussions

- Compressibility *always* leaks information about data
  - More info the better the compression algorithm is

Generalizing attacks to other compressed caches

- Most compressed cache architectures allow conflicts among a small set of lines → Pack+Probe still applies
  - See paper for more discussions

- Compressibility *always* leaks information about data
  - More info the better the compression algorithm is
  - Adaptive compression algorithms use shared state

Generalizing attacks to other compressed caches

- Most compressed cache architectures allow conflicts among a small set of lines → Pack+Probe still applies
  - See paper for more discussions

- Compressibility *always* leaks information about data
  - More info the better the compression algorithm is
  - Adaptive compression algorithms use shared state
    → additional attack vector

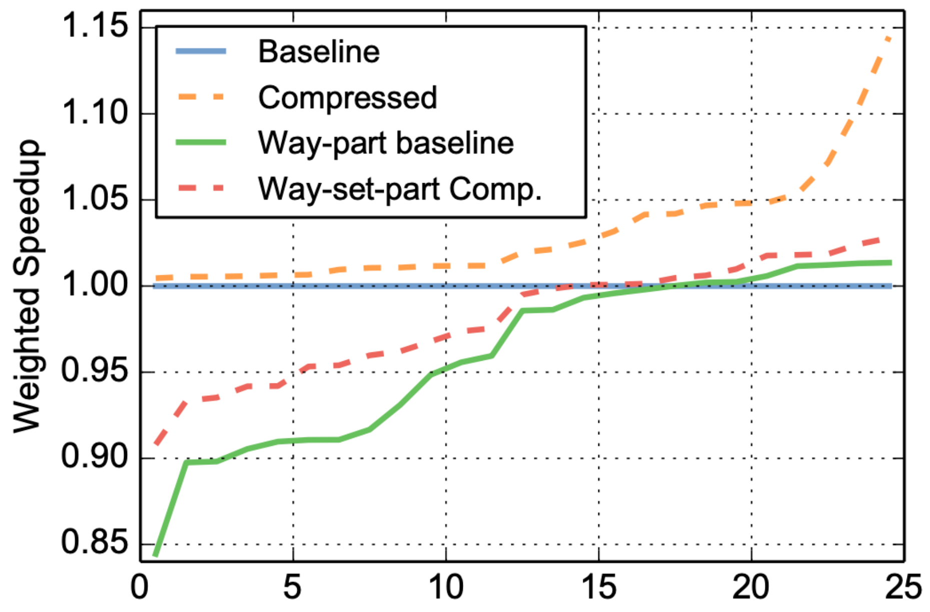
Defense against cache compression attacks

Defense against cache compression attacks

- Cache partitioning for isolation
  - Prevents attacks without software changes
  - Invasive: must partition both tag and data arrays

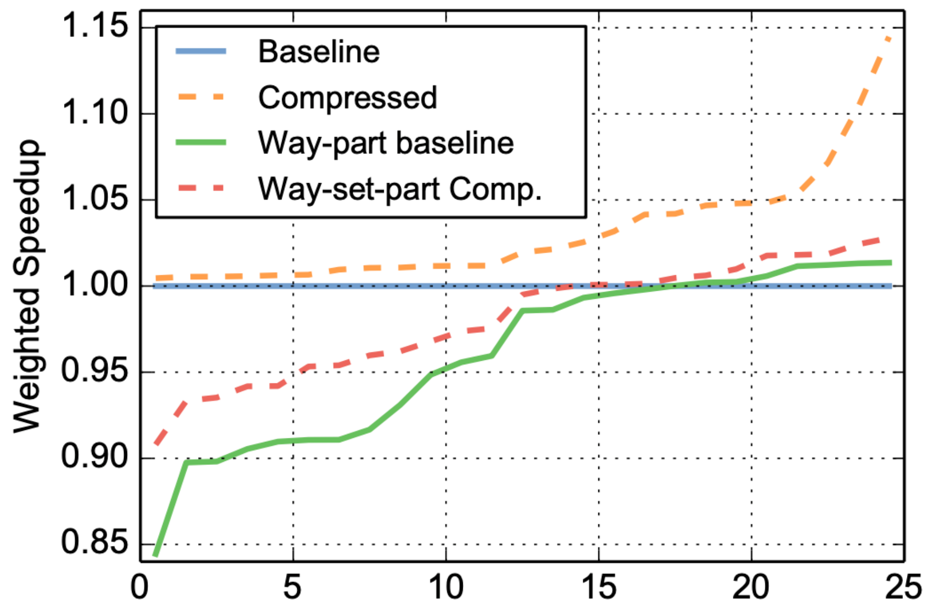
Defense against cache compression attacks

- Cache partitioning for isolation
  - Prevents attacks without software changes
  - Invasive: must partition both tag and data arrays
- Performance distribution of 25 mixes of 4 SPEC CPU2006 apps, using no and static partitioning:



Defense against cache compression attacks

- Cache partitioning for isolation
  - Prevents attacks without software changes
  - Invasive: must partition both tag and data arrays
- Performance distribution of 25 mixes of 4 SPEC CPU2006 apps, using no and static partitioning:



Partitioning increases fragmentation in VSC, reduces effective compression ratio

See paper for more!

- Other possible defenses for compressed cache attacks
- Examples of vulnerable apps due to colocation with attacker-controlled data
- Discussion on generalizing attacks to other compressed caches
- Artifact description

Conclusions

- Compressed caches introduce new side channel & attacks

Conclusions

- Compressed caches introduce new side channel & attacks

- Pack+Probe exploits compressed cache architectures to observe compressibility of victim's lines

Conclusions

- Compressed caches introduce new side channel & attacks
- Pack+Probe exploits compressed cache architectures to observe compressibility of victim's lines
- Safecracker exploits compression algorithms + colocation of attacker-controlled & secret data to leak data quickly
  - Can leak a large fraction of program data
  - Potentially as damaging as speculation-based attacks

Conclusions

- Compressed caches introduce new side channel & attacks

- Pack+Probe exploits compressed cache architectures to observe compressibility of victim's lines

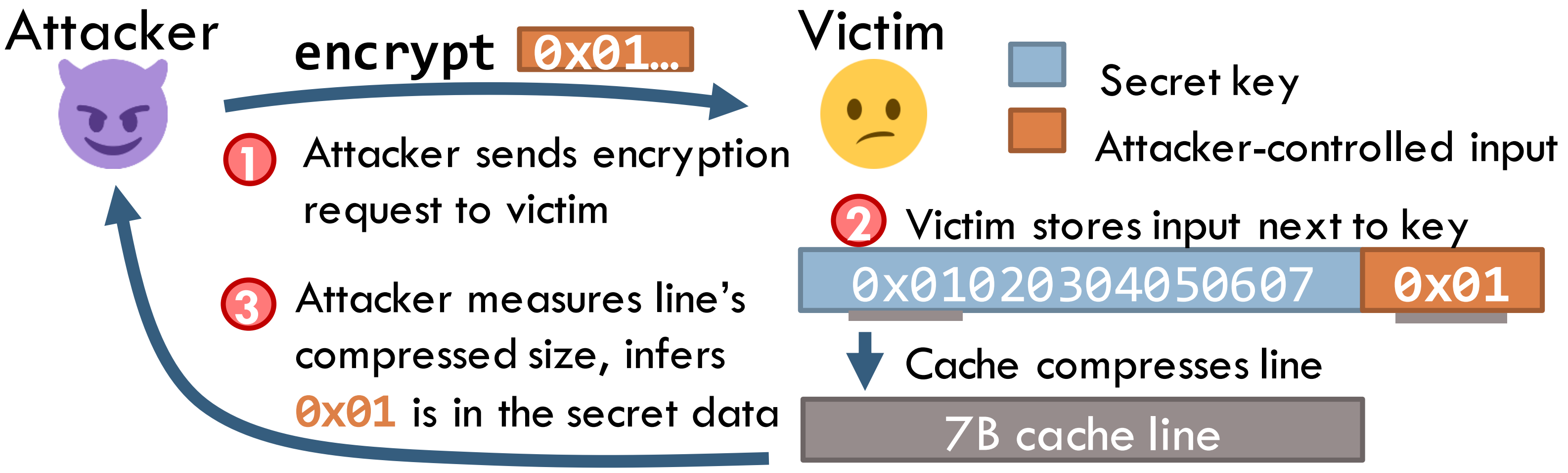
- Safecracker exploits compression algorithms + colocation of attacker-controlled & secret data to leak data quickly
  - Can leak a large fraction of program data
  - Potentially as damaging as speculation-based attacks

- Defenses have drawbacks
  - Motivates future work on efficient defenses

Thank you for watching!

Share your questions/comments with us!

Safecracker: Leaking Secrets through Compressed Caches



Compromises secret key in ~10ms