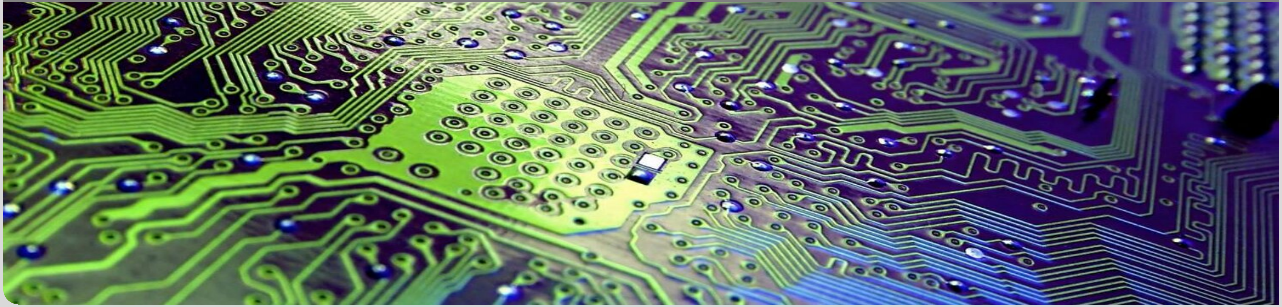
Karlsruhe Institute of Technology

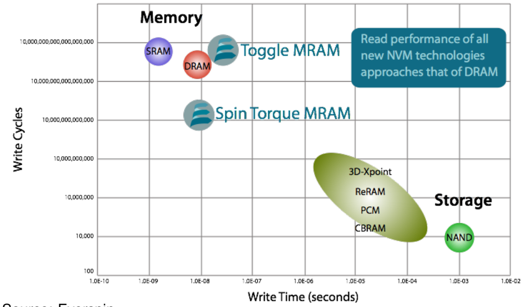
Data Leakage through Self-Terminated Write Schemes in Memristive Caches

Jonas Krautter, Mahta Mayahinia, Dennis R. E. Gnad, Mehdi B. Tahoori | 2022-01-20

INSTITUTE OF COMPUTER ENGINEERING – CHAIR OF DEPENDABLE NANO COMPUTING



Motivation

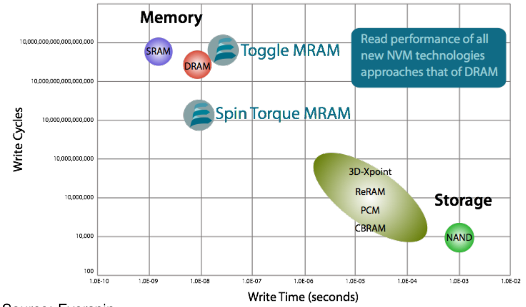


Source: Everspin

- Memristive memory close to reaching SRAM/DRAM performance

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

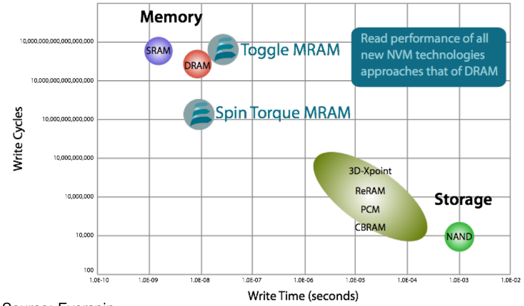
Motivation



Source: Everspin

- Memristive memory close to reaching SRAM/DRAM performance
- Major benefits: Power, density, efficiency, non-volatility

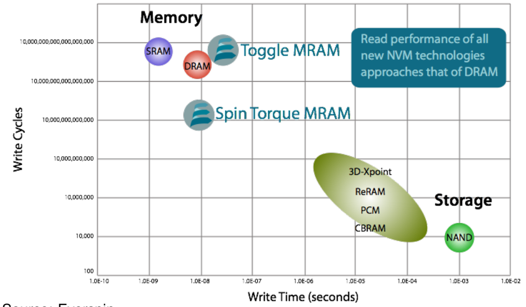
Motivation



Source: Everspin

- Memristive memory close to reaching SRAM/DRAM performance
- Major benefits: Power, density, efficiency, non-volatility
- Different emerging technologies: STT-MRAM, ReRAM, PCM, ...

Motivation

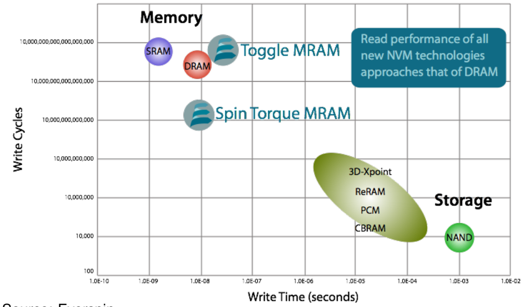


Source: Everspin

- Memristive memory close to reaching SRAM/DRAM performance
- Major benefits: Power, density, efficiency, non-volatility
- Different emerging technologies: STT-MRAM, ReRAM, PCM, ...
- Aside from challenges for manufacturability: Security a major concern!

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Motivation



Source: Everspin

- Memristive memory close to reaching SRAM/DRAM performance
- Major benefits: Power, density, efficiency, non-volatility
- Different emerging technologies: STT-MRAM, ReRAM, PCM, ...
- Aside from challenges for manufacturability: Security a major concern!
- Rowhammer is still a problem in DRAM...[1]

[1] Frigo et al., "TRRespass: Exploiting the Many Sides of Target Row Refresh", S&P 2021

Motivation

- Reliable write is **asymmetric**

Motivation

- Reliable write is **asymmetric**
- 0→1 vs. 1→0 have different delay/power

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Motivation

- Reliable write is **asymmetric**
- 0→1 vs. 1→0 have different delay/power
- This is the case for almost all memristive memory technologies

Motivation

- Reliable write is **asymmetric**
- $0 \rightarrow 1$ vs. $1 \rightarrow 0$ have different delay/power
- This is the case for almost all memristive memory technologies
- $\Rightarrow$ Self-terminated write[1] proposed for performance benefits

[1] Suzuki et al., "Cost-Efficient Self-Terminated Write Driver for Spin-Transfer-Torque RAM and Logic", IEEE Trans. Magn. 2014

Motivation

- Reliable write is **asymmetric**
- 0→1 vs. 1→0 have different delay/power
- This is the case for almost all memristive memory technologies
- ⇒ Self-terminated write[1] proposed for performance benefits
- ⇒ Data-dependent timing can be exploited by an attacker!

[1] Suzuki et al., "Cost-Efficient Self-Terminated Write Driver for Spin-Transfer-Torque RAM and Logic", IEEE Trans. Magn. 2014

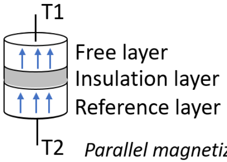
Outline

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

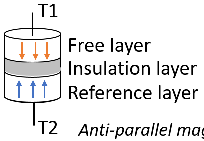
Outline

Memristive Memory Technologies

Spin Transfer Torque Magnetic RAM (STT-MRAM):

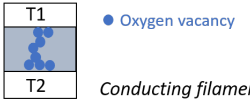


Parallel magnetization (LRS)

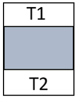


Anti-parallel magnetization (HRS)

Resistive RAM (ReRAM):

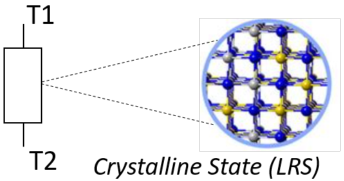


Conducting filament (LRS)

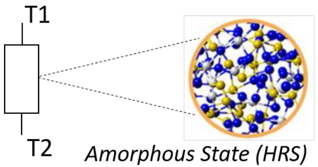


No conducting filament (HRS)

Phase Change Memory (PCM):



Crystalline State (LRS)



Amorphous State (HRS)

Self-Terminated Write Schemes

Transition	Timing	STT-MRAM/ReRAM encoding	PCM encoding
0→0	t_{ns}	LRS→LRS	HRS→HRS
0→1	t_{ss}	LRS→HRS	HRS→LRS
1→0	t_{fs}	HRS→LRS	LRS→HRS
1→1	t_{ns}	HRS→HRS	LRS→LRS

- Different transitions have different timing (technology-dependent)

Self-Terminated Write Schemes

Transition	Timing	STT-MRAM/ReRAM encoding	PCM encoding
0→0	t_{ns}	LRS→LRS	HRS→HRS
0→1	t_{ss}	LRS→HRS	HRS→LRS
1→0	t_{fs}	HRS→LRS	LRS→HRS
1→1	t_{ns}	HRS→HRS	LRS→LRS

- Different transitions have different timing (technology-dependent)
- $t_{ns} < t_{fs} < t_{ss}$

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Self-Terminated Write Schemes

Transition	Timing	STT-MRAM/ReRAM encoding	PCM encoding
0→0	t_{ns}	LRS→LRS	HRS→HRS
0→1	t_{ss}	LRS→HRS	HRS→LRS
1→0	t_{fs}	HRS→LRS	LRS→HRS
1→1	t_{ns}	HRS→HRS	LRS→LRS

- Different transitions have different timing (technology-dependent)
- $t_{ns} < t_{fs} < t_{ss}$
- Terminating the write after successful transition
  $\Rightarrow$ Energy and performance benefits

Self-Terminated Write Schemes

Transition	Timing	STT-MRAM/ReRAM encoding	PCM encoding
0→0	t_{ns}	LRS→LRS	HRS→HRS
0→1	t_{ss}	LRS→HRS	HRS→LRS
1→0	t_{fs}	HRS→LRS	LRS→HRS
1→1	t_{ns}	HRS→HRS	LRS→LRS

- Different transitions have different timing (technology-dependent)
- $t_{ns} < t_{fs} < t_{ss}$
- Terminating the write after successful transition
  $\Rightarrow$ Energy and performance benefits
- Performance benefit: Propagating the write time to architecture level

Related Work

- First technology-specific attacks: **Cold-boot attacks**[1] (exploit non-volatility)

[1] Halderman et al., ”Lest we remember: cold-boot attacks on encryption keys”, Comm. of the ACM 2009

Related Work

- First technology-specific attacks: **Cold-boot attacks**[1] (exploit non-volatility)
- **Asymmetric** read and write power can reveal data[2]

[1] Halderman et al., "Lest we remember: cold-boot attacks on encryption keys", Comm. of the ACM 2009

[2] Iyengar et al., "Side channel attacks on STTRAM and low-overhead countermeasures", DFT 2016

Related Work

- First technology-specific attacks: **Cold-boot attacks**[1] (exploit non-volatility)
- **Asymmetric** read and write power can reveal data[2]
- Power **side-channel attacks** on STT-MRAM[3]

[1] Halderman et al., ”Lest we remember: cold-boot attacks on encryption keys”, Comm. of the ACM 2009

[2] Iyengar et al., ”Side channel attacks on STTRAM and low-overhead countermeasures”, DFT 2016

[3] Khan et al., ”Side-Channel Attack on STTRAM Based Cache for Cryptographic Application”, ICCD 2017

Related Work

- First technology-specific attacks: **Cold-boot attacks**[1] (exploit non-volatility)
- **Asymmetric** read and write power can reveal data[2]
- Power **side-channel attacks** on STT-MRAM[3]
- Bit-cell design to **mitigate** data-dependent leakage[4]

[1] Halderman et al., ”Lest we remember: cold-boot attacks on encryption keys”, Comm. of the ACM 2009

[2] Iyengar et al., ”Side channel attacks on STTRAM and low-overhead countermeasures”, DFT 2016

[3] Khan et al., ”Side-Channel Attack on STTRAM Based Cache for Cryptographic Application”, ICCD 2017

[4] Dodo et al., ”Secure STT-MRAM bit-cell design resilient to differential power analysis attacks”, TVLSI 2019

Related Work

- First technology-specific attacks: **Cold-boot attacks**[1] (exploit non-volatility)
- **Asymmetric** read and write power can reveal data[2]
- Power **side-channel attacks** on STT-MRAM[3]
- Bit-cell design to **mitigate** data-dependent leakage[4]
- Cache-timing attacks with a similar threat model[5,6]
  $\Rightarrow$ But they rely on distinguishing **cached** vs. **uncached** access!

[1] Halderman et al., "Lest we remember: cold-boot attacks on encryption keys", Comm. of the ACM 2009

[2] Iyengar et al., "Side channel attacks on STTRAM and low-overhead countermeasures", DFT 2016

[3] Khan et al., "Side-Channel Attack on STTRAM Based Cache for Cryptographic Application", ICCD 2017

[4] Dodo et al., "Secure STT-MRAM bit-cell design resilient to differential power analysis attacks", TVLSI 2019

[5] Osvik et al., "Cache attacks and countermeasures: the case of AES", RSA Conference 2006

[6] Yarom et al., "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack", USENIX 2014

Outline

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Principle

- Basic principle: **Write-After-Write**

Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)

Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)
- Data resides in victim process address space, inaccessible to attacker

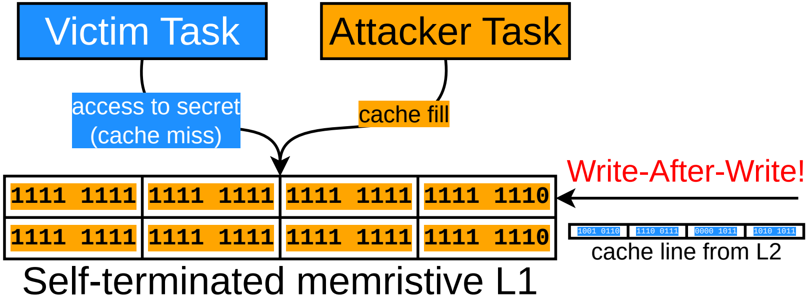
Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)
- Data resides in victim process address space, inaccessible to attacker
- Goal: Force victim to **overwrite** attacker data in cache
  $\Rightarrow$ Timing side-channel for bitwise data extraction

Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)
- Data resides in victim process address space, inaccessible to attacker
- Goal: Force victim to **overwrite** attacker data in cache
  $\Rightarrow$ Timing side-channel for bitwise data extraction

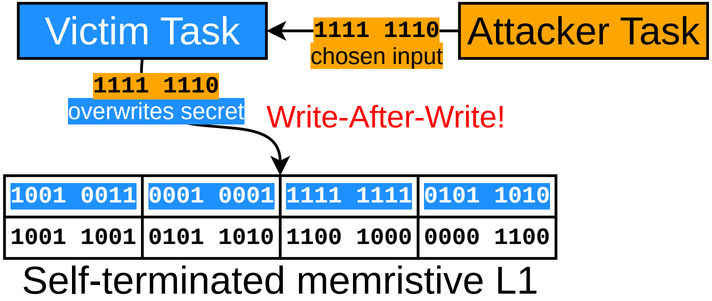
$\Rightarrow$ Can be a **cache-miss** after the attacker filled the cache...



Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)
- Data resides in victim process address space, inaccessible to attacker
- Goal: Force victim to **overwrite** attacker data in cache
  $\Rightarrow$ Timing side-channel for bitwise data extraction

... or overwriting the attacker data directly (**cache-hit**)...

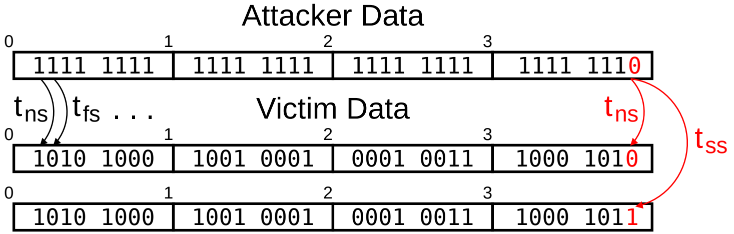


Attack Principle

- Basic principle: **Write-After-Write**
- Victim process has secret data (e.g. encryption key)
- Data resides in victim process address space, inaccessible to attacker
- Goal: Force victim to **overwrite** attacker data in cache
  $\Rightarrow$ Timing side-channel for bitwise data extraction

...or many **other variants!** (not exclusive to cache)

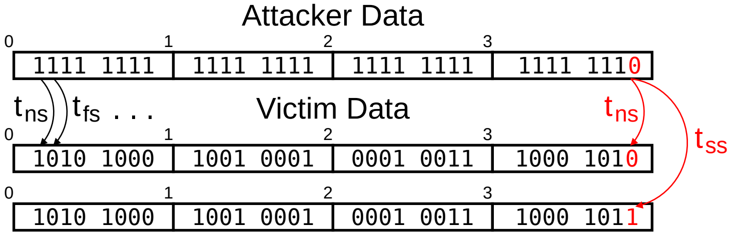
Attack Principle



- Attacker wants to know bit b_i with $i \in \{0, 1, ..., 512\}$ (cache-line size)

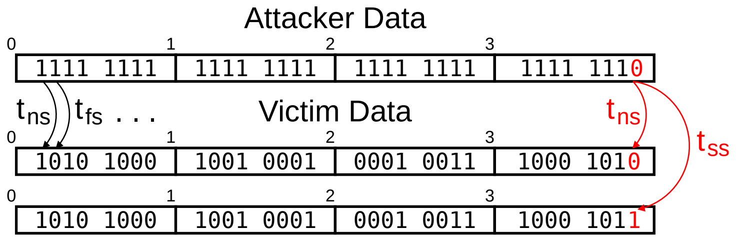
J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Principle



- Attacker wants to know bit b_i with $i \in \{0, 1, ..., 512\}$ (cache-line size)
- Cache filled with 1 except for $b_i = 0$

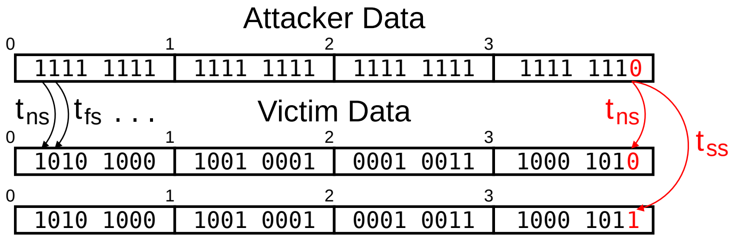
Attack Principle



- Attacker wants to know bit b_i with $i \in \{0, 1, ..., 512\}$ (cache-line size)
- Cache filled with 1 except for $b_i = 0$
- Victim overwrites cache line
  $\Rightarrow$ Write latency is t_{ns}, t_{fs} or t_{ss} depending on b_i

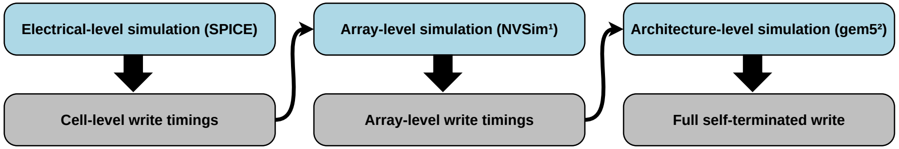
J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Principle



- Attacker wants to know bit b_i with $i \in \{0, 1, ..., 512\}$ (cache-line size)
- Cache filled with 1 except for $b_i = 0$
- Victim overwrites cache line
  $\Rightarrow$ Write latency is t_{ns}, t_{fs} or t_{ss} depending on b_i
- Victim execution time measurement using cycle counters (e.g. *rdtsc*)

Simulation

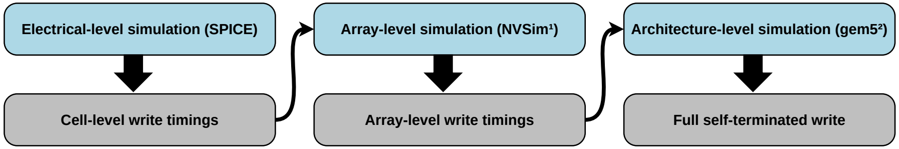

Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim¹)
Array-level write timings
Architecture-level simulation (gem5²)
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing

[1] Dong et al., ”NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory”, TCAD 2012

[2] Binkert et al., ”The Gem5 Simulator”, SIGARCH 2011

Simulation

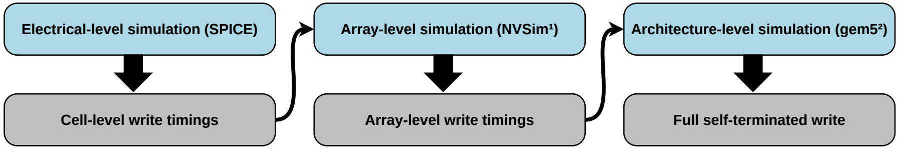

Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim[1])
Array-level write timings
Architecture-level simulation (gem5[2])
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing
- Architecture-level: Syscall Emulation and Full System

[1] Dong et al., "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory", TCAD 2012

[2] Binkert et al., "The Gem5 Simulator", SIGARCH 2011

Simulation

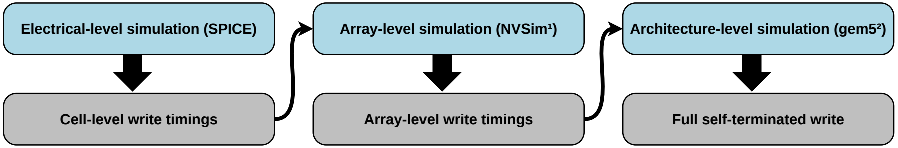

Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim¹)
Array-level write timings
Architecture-level simulation (gem5²)
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing
- Architecture-level: Syscall Emulation and Full System
- Two ISA: ARMv8, x86_64

[1] Dong et al., "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory", TCAD 2012

[2] Binkert et al., "The Gem5 Simulator", SIGARCH 2011

Simulation

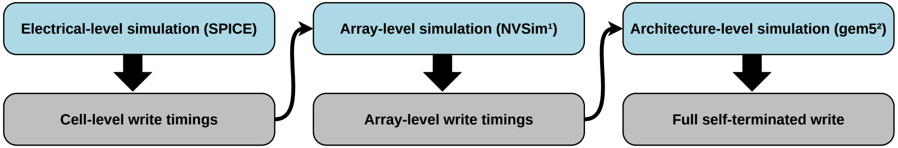

Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim[1])
Array-level write timings
Architecture-level simulation (gem5[2])
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing
- Architecture-level: Syscall Emulation and Full System
- Two ISA: ARMv8, x86_64
- 2-level cache architecture, both caches with self-terminated write

[1] Dong et al., ”NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory”, TCAD 2012

[2] Binkert et al., ”The Gem5 Simulator”, SIGARCH 2011

Simulation



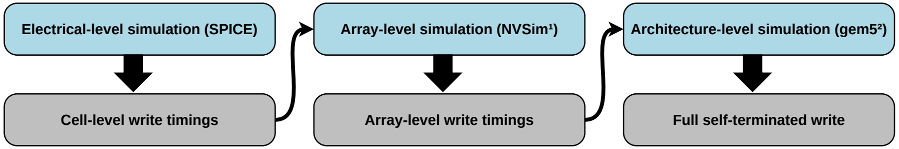
Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim¹)
Array-level write timings
Architecture-level simulation (gem5²)
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing
- Architecture-level: Syscall Emulation and Full System
- Two ISA: ARMv8, x86_64
- 2-level cache architecture, both caches with self-terminated write
- Cache line width is 64 bytes

[1] Dong et al., ”NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory”, TCAD 2012

[2] Binkert et al., ”The Gem5 Simulator”, SIGARCH 2011

Simulation


Electrical-level simulation (SPICE)
Cell-level write timings
Array-level simulation (NVSim[1])
Array-level write timings
Architecture-level simulation (gem5[2])
Full self-terminated write

- Array level = Bit-cell level + address decoding + routing
- Architecture-level: Syscall Emulation and Full System
- Two ISA: ARMv8, x86_64
- 2-level cache architecture, both caches with self-terminated write
- Cache line width is 64 bytes
- Write latency for 64 $\times$ 8 bits is **maximum** of write latency for each bit (all bits written in parallel)

[1] Dong et al., "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory", TCAD 2012

[2] Binkert et al., "The Gem5 Simulator", SIGARCH 2011

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Variant 1

- Attacker fills the cache with known pattern

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Variant 1

- Attacker fills the cache with known pattern
- Victim overwrites attacker data when *secret* is loaded into cache

```
void victim_code_v1() {
    secret[0] &= 0xFF;
}
```

Attack Variant 1

- Attacker fills the cache with known pattern
- Victim overwrites attacker data when *secret* is loaded into cache

```
void victim_code_v1() {
    secret[0] &= 0xFF;
}
```

- Attacker measures **data-dependent** victim execution time

Attack Variant 1

- Attacker fills the cache with known pattern
- Victim overwrites attacker data when *secret* is loaded into cache

```
void victim_code_v1() {
    secret[0] &= 0xFF;
}
```

- Attacker measures **data-dependent** victim execution time
- $\Rightarrow$ Attacker learns bit b_i of *secret*

Attack Variant 1

- Attacker fills the cache with known pattern
- Victim overwrites attacker data when *secret* is loaded into cache

```
void victim_code_v1() {
    secret[0] &= 0xFF;
}
```

- Attacker measures **data-dependent** victim execution time
- $\Rightarrow$ Attacker learns bit b_i of *secret*
- Improved variant: Fill only the cache set where secret data resides

Attack Variant 2

- Attacker provides known pattern as input (e.g. chosen-plaintext)

Attack Variant 2

- Attacker provides known pattern as input (e.g. chosen-plaintext)
- Victim overwrites the attacker data directly in the cache

```
void victim_code_v2(uint8_t *ptr) {
    for (unsigned int i = 0; i < SIZE; i++)
        ptr[i] ^= secret[i];
}
```

Attack Variant 2

- Attacker provides known pattern as input (e.g. chosen-plaintext)
- Victim overwrites the attacker data directly in the cache

```
void victim_code_v2(uint8_t *ptr) {
    for (unsigned int i = 0; i < SIZE; i++)
        ptr[i] ^= secret[i];
}
```

- Attacker measures **data-dependent** victim execution time

Attack Variant 2

- Attacker provides known pattern as input (e.g. chosen-plaintext)
- Victim overwrites the attacker data directly in the cache

```
void victim_code_v2(uint8_t *ptr) {
    for (unsigned int i = 0; i < SIZE; i++)
        ptr[i] ^= secret[i];
}
```

- Attacker measures **data-dependent** victim execution time
- $\Rightarrow$ Attacker learns bit b_i of *secret*

Outline

Array Level Timings

Technology	Ref.	Array-level timing (from *NVSim*)	
		t_{fs} $(1 \rightarrow 0)$	t_{ss} $(0 \rightarrow 1)$
STT-MRAM (1)	[1,2]	∼6.3 ns (7 cycles)	∼7.6 ns (8 cycles)
STT-MRAM (2)	[3]	∼4.5 ns (5 cycles)	∼9.1 ns (10 cycles)
PCM	[4]	∼50.5 ns (51 cycles)	∼100.5 ns (101 cycles)
ReRAM	[5]	∼25.5 ns (26 cycles)	∼125.5 ns (126 cycles)

- Cycles reported for a 1 GHz clock (as simulated in *gem5*)
- t_{ns} $(0 \rightarrow 0$ and $1 \rightarrow 1)$ is one clock cycle for all technologies

[1] Dong et al., "A 1Mb 28nm STT-MRAM with 2.8ns read access time at 1.2V VDD ...", ISSCC 2018

[2] Sato et al., "14ns write speed 128Mb density Embedded STT-MRAM with endurance $> 10^{10}$ and 10yrs retention...", IEDM 2018

[3] Bishnoi et al., "Avoiding unnecessary write operations in STT-MRAM for low power implementation", ISQED 2014

[4] Fong et al., "Phase-Change Memory—Towards a Storage-Class Memory", TED 2017

[5] Chen et al., "A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions...", IEDM 2017

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Attack Byte-Transfer Rates

Technology/ISA	Attack transfer rates (kB/s)		
	Variant 1 (L1 write miss)		Variant 2 (L1 write hit)
	cache fill	set fill	
Syscall Emulation Mode			
STT-MRAM(1)/x86	0.050	17.5	**18.8**
STT-MRAM(2)/x86	0.049	17.0	18.3
PCM/x86	0.022	7.4	6.8
ReRAM/x86	0.019	6.6	6.1
Full System Mode			
STT-MRAM(1)/x86	0.048	× *	2.0
STT-MRAM(1)/ARM	× *	× *	2.8

* No conclusive results were acquired, but more effort could lead to a successful attack.

J. Krautter, M. Mayahinia, D. Gnad, M. Tahoori

Outline

Discussion

- Attack depends on high-resolution timing measurement
  $\Rightarrow$ Statistical methods if not available

Discussion

- Attack depends on high-resolution timing measurement
  $\Rightarrow$ Statistical methods if not available
- Attack Variant 2 possible in systems without cache

Discussion

- Attack depends on high-resolution timing measurement
  $\Rightarrow$ Statistical methods if not available
- Attack Variant 2 possible in systems without cache
- Self-terminated write can be disabled as a countermeasure

Benchmark	Self-terminated write...		Performance loss
	...enabled	...disabled	
blackscholes	$0.277s$	$0.282s$	$\approx 1.8\%$
bodytrack	$1.388s$	$1.424s$	$\approx 2.6\%$
canneal	$1.448s$	$1.493s$	$\approx 0.3\%$
dedup	$4.600s$	$5.071s$	**$\approx 10.2\%$**

[1] Sayed et al., "Opportunistic write for fast and reliable STT-MRAM", DATE 2017

Discussion

- Attack depends on high-resolution timing measurement
  $\Rightarrow$ Statistical methods if not available
- Attack Variant 2 possible in systems without cache
- Self-terminated write can be disabled as a countermeasure

Benchmark	Self-terminated write...		Performance loss
	...enabled	...disabled	
blackscholes	0.277s	0.282s	$\approx$ 1.8%
bodytrack	1.388s	1.424s	$\approx$ 2.6%
canneal	1.448s	1.493s	$\approx$ 0.3%
dedup	4.600s	5.071s	**$\approx$ 10.2%**

- Alternatively: More agressive (but balanced) write time optimization[1]

[1] Sayed et al., "Opportunistic write for fast and reliable STT-MRAM", DATE 2017

Discussion

Karlsruhe Institute of Technology

- Attack depends on high-resolution timing measurement
  $\Rightarrow$ Statistical methods if not available
- Attack Variant 2 possible in systems without cache
- Self-terminated write can be disabled as a countermeasure

Benchmark	Self-terminated write...		Performance loss
	...enabled	...disabled	
blackscholes	$0.277s$	$0.282s$	$\approx 1.8\%$
bodytrack	$1.388s$	$1.424s$	$\approx 2.6\%$
canneal	$1.448s$	$1.493s$	$\approx 0.3\%$
dedup	$4.600s$	$5.071s$	$\approx$ **10.2%**

- Alternatively: More agressive (but balanced) write time optimization[1]
- Asymmetric power is still an issue against power side-channel attacks (but those are much harder to exploit remotely)

[1] Sayed et al., "Opportunistic write for fast and reliable STT-MRAM", DATE 2017

Conclusion

- Memristive memories soon to be adopted in many devices

Conclusion

- Memristive memories soon to be adopted in many devices
- Self-terminating write schemes proposed for energy/performance

Conclusion

- Memristive memories soon to be adopted in many devices
- Self-terminating write schemes proposed for energy/performance
- We showed a security flaw introduced by self-terminating write

Conclusion

Karlsruhe Institute of Technology

- Memristive memories soon to be adopted in many devices
- Self-terminating write schemes proposed for energy/performance
- We showed a security flaw introduced by self-terminating write
- Attackers can read secret data at up to 20 kB/s

Conclusion

- Memristive memories soon to be adopted in many devices
- Self-terminating write schemes proposed for energy/performance
- We showed a security flaw introduced by self-terminating write
- Attackers can read secret data at up to 20 kB/s
- $\Rightarrow$ Keep security in mind when optimizing performance/power!

Thank you for your attention!

Questions? Write us an email!

{jonas.krautter,mahta.mayahinia,dennis.gnad,mehdi.tahoori}@kit.edu