

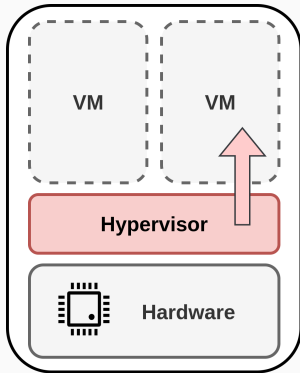
When Memory Lies

Breaching Processor Security via Rogue Memory Modules

Luca Wilke & Jesse De Meulemeester

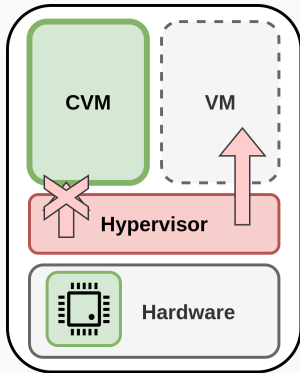
2025-11-21, [Hardwear.io](https://www.hardwear.io)

Why Trusted Execution Environments?



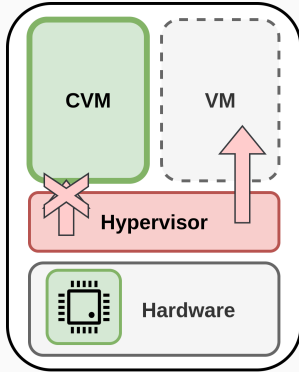
- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference

Why Trusted Execution Environments?



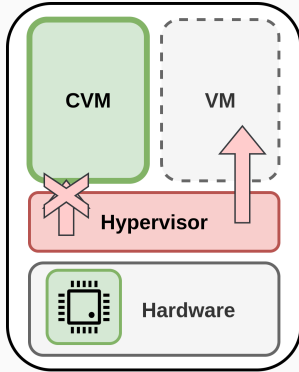
- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference

Why Trusted Execution Environments?



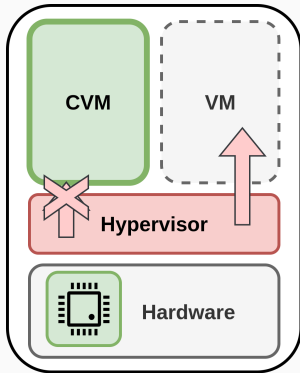
- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference

Why Trusted Execution Environments?



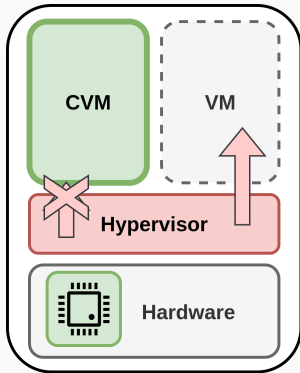
- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference

Why Trusted Execution Environments?



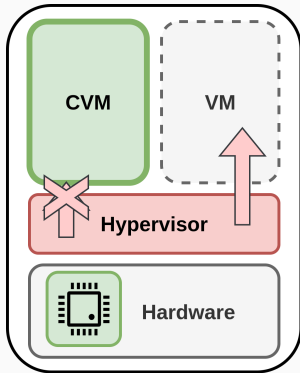
- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference
- **However** strong attacker model enables a vast amount of attacks

Why Trusted Execution Environments?



- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference
- **However** strong attacker model enables a vast amount of attacks
 - Privileged software-level adversaries

Why Trusted Execution Environments?



- Without TEEs cloud provider has full access to VMs
- Pitch: TEEs lock out the cloud provider
- Enable computing on private data in the cloud
- Use Case: Confidential AI inference
- **However** strong attacker model enables a vast amount of attacks
 - Privileged software-level adversaries
 - Hardware-level adversaries

Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

[illegible]

Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

TEE	Encryption	Guarantees		
		<i>Confidentiality</i>	<i>Integrity</i>	<i>Freshness</i>
Intel Classic SGX	AES-CTR	✓	✓	✓

Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

TEE	Encryption	Guarantees		
		<i>Confidentiality</i>	<i>Integrity</i>	<i>Freshness</i>
Intel Classic SGX	AES-CTR	✓	✓	✓
Intel Scalable SGX	AES-XTS	✓	✗	✗

Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

TEE	Encryption	Guarantees		
		<i>Confidentiality</i>	<i>Integrity</i>	<i>Freshness</i>
Intel Classic SGX	AES-CTR	✓	✓	✓
Intel Scalable SGX	AES-XTS	✓	✗	✗
AMD SEV-SNP	AES-XEX	✓	✗	✗

Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

TEE	Encryption	Guarantees		
		<i>Confidentiality</i>	<i>Integrity</i>	<i>Freshness</i>
Intel Classic SGX	AES-CTR	✓	✓	✓
Intel Scalable SGX	AES-XTS	✓	✗	✗
AMD SEV-SNP	AES-XEX	✓	✗	✗
Intel TDX	AES-XTS	✓	✓	✗

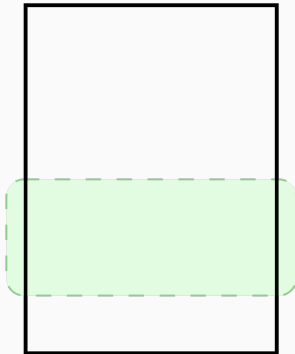
Memory Encryption in TEEs

- Fundamental properties
 - Confidentiality
 - Integrity
 - Freshness

TEE	Encryption	Guarantees		
		<i>Confidentiality</i>	<i>Integrity</i>	<i>Freshness</i>
Intel Classic SGX	AES-CTR	✓	✓	✓
Intel Scalable SGX	AES-XTS	✓	✗	✗
AMD SEV-SNP	AES-XEX	✓	✗	✗
Intel TDX	AES-XTS	✓	✓	✗
Arm CCA	AES-XEX/QARMA	✓	✗	✗

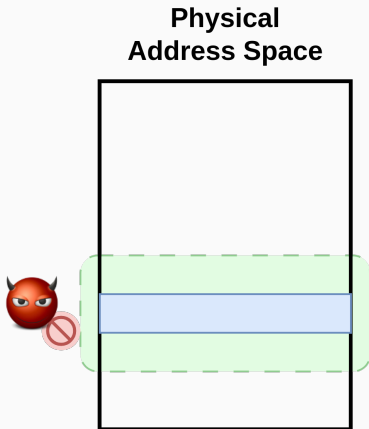
Memory Isolation in TEEs

Physical
Address Space



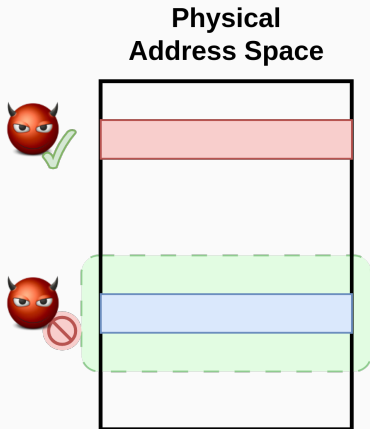
- Prevent access to TEE memory
 - SEV: *Ciphertext hiding*
 - SGX/TDX: Return fixed value & poison on write

Memory Isolation in TEEs



- Prevent access to TEE memory
 - SEV: *Ciphertext hiding*
 - SGX/TDX: Return fixed value & poison on write

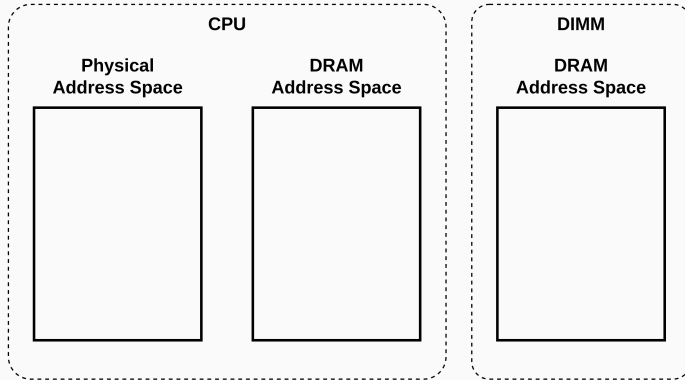
Memory Isolation in TEEs



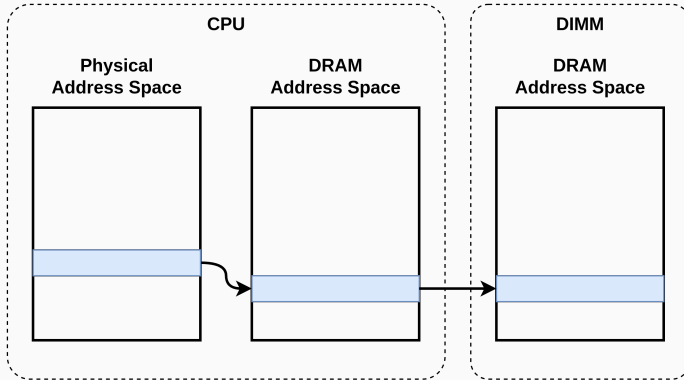
- Prevent access to TEE memory
 - SEV: *Ciphertext hiding*
 - SGX/TDX: Return fixed value & poison on write

Can DIMMs be manipulated to break integrity protections in scalable TEE designs?

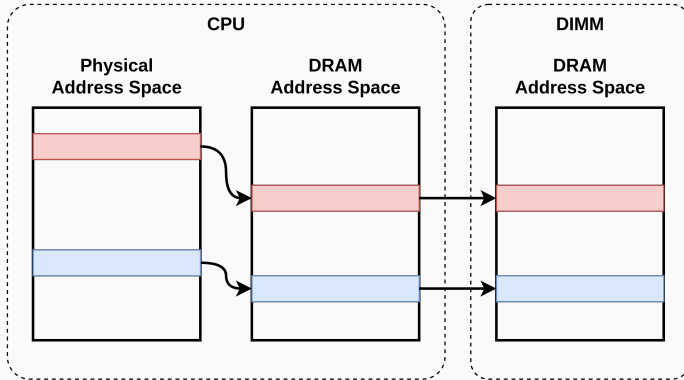
DRAM Addressing



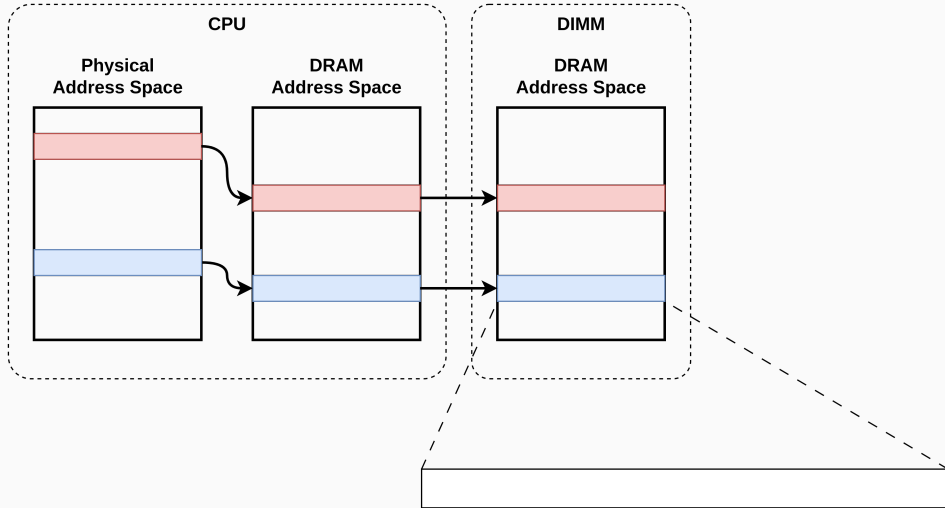
DRAM Addressing



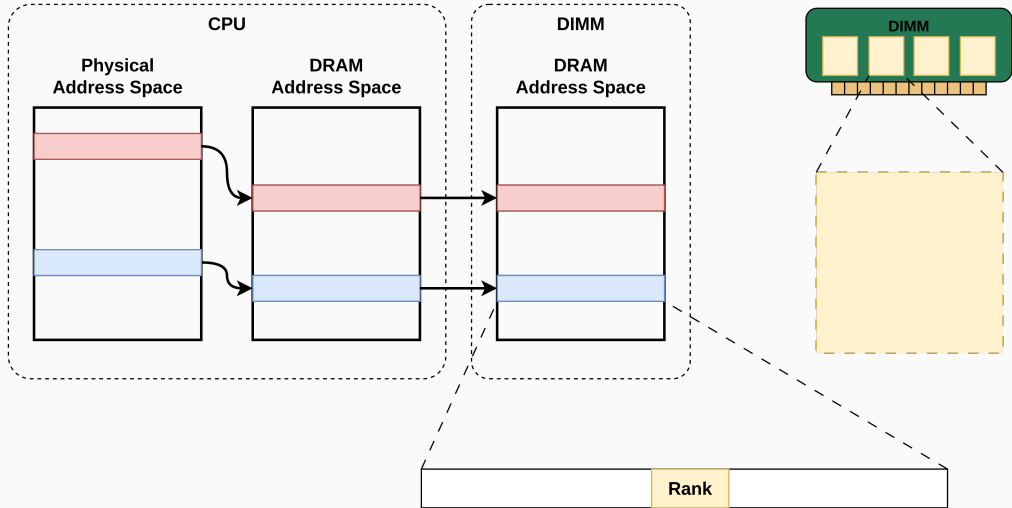
DRAM Addressing



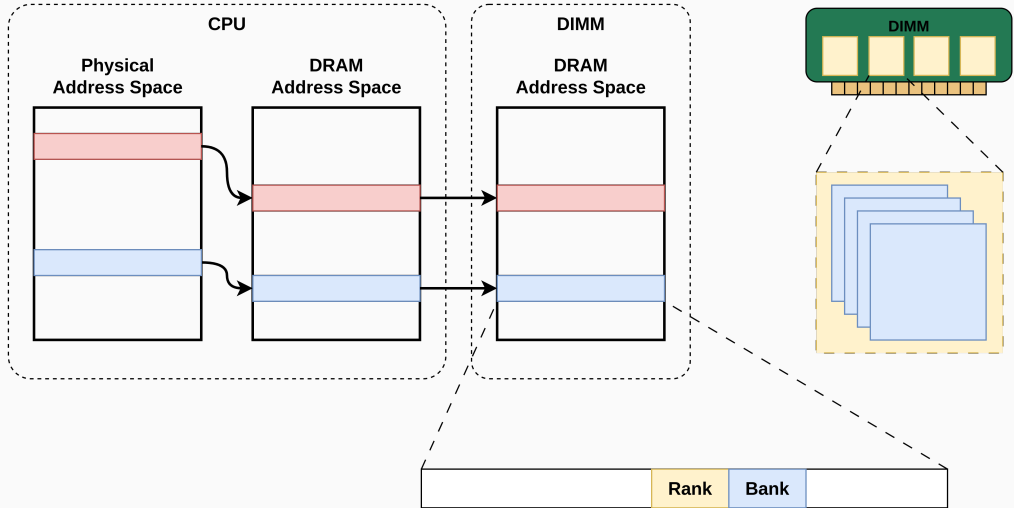
DRAM Addressing



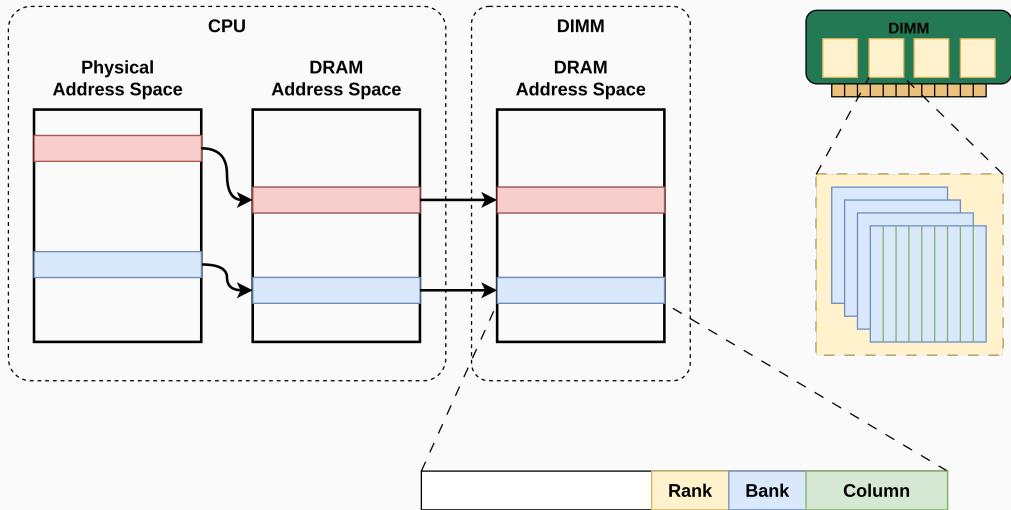
DRAM Addressing



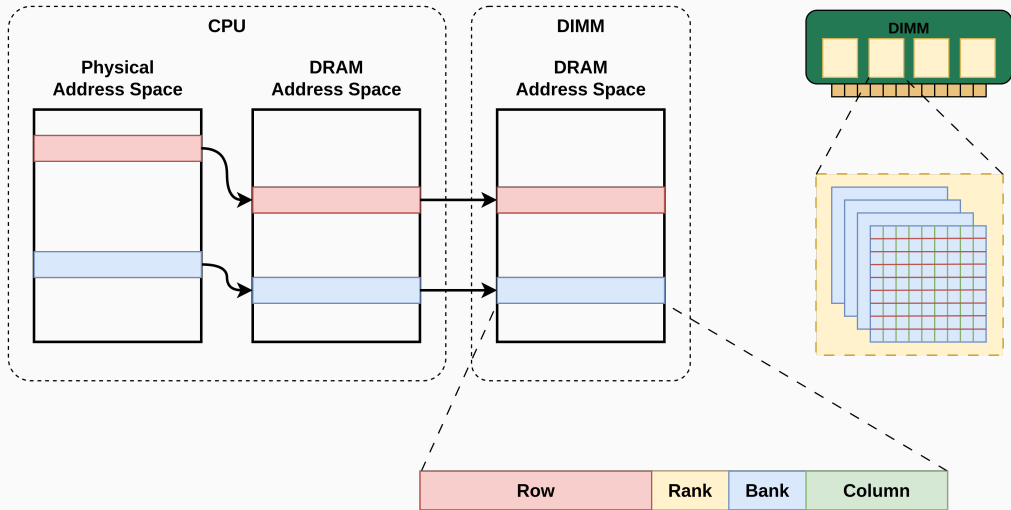
DRAM Addressing



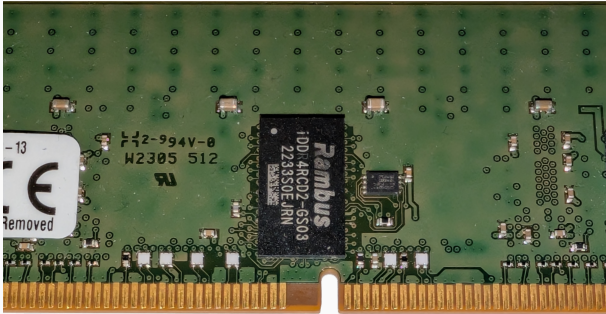
DRAM Addressing



DRAM Addressing

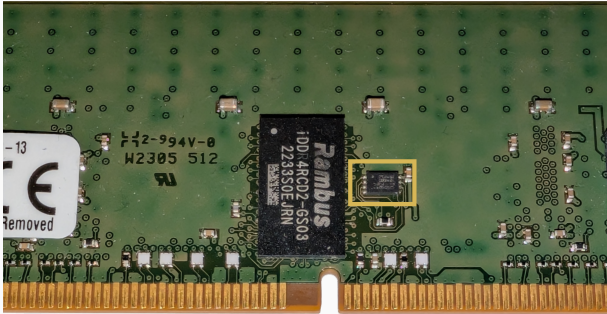


Serial Presence Detect



- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect



- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect

Byte Number		Function Described	Notes
0	0x000	Number of Serial PD Bytes Written / SPD Device Size	1, 2
1	0x001	SPD Revision	
2	0x002	Key Byte / DRAM Device Type	
3	0x003	Key Byte / Module Type	
4	0x004	SDRAM Density and Banks	3
5	0x005	SDRAM Addressing	3
6	0x006	Primary SDRAM Package Type	3
7	0x007	SDRAM Optional Features	3
8	0x008	SDRAM Thermal and Refresh Options	3
9	0x009	Other SDRAM Optional Features	3
10	0x00A	Secondary SDRAM Package Type	3
11	0x00B	Module Nominal Voltage, VDD	3
12	0x00C	Module Organization	
13	0x00D	Module Memory Bus Width	
14	0x00E	Module Thermal Sensor	
15	0x00F	Extended module type	
16	0x010	Reserved -- must be coded as 0x00	
17	0x011	Timebases	
18	0x012	SDRAM Minimum Cycle Time ($t_{CKAVGmin}$)	3
19	0x013	SDRAM Maximum Cycle Time ($t_{CKAVGmax}$)	3
20	0x014	CAS Latencies Supported, First Byte	3
21	0x015	CAS Latencies Supported, Second Byte	3

- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect

Byte Number		Function Described	Notes
0	0x000	Number of Serial PD Bytes Written / SPD Device Size	1, 2
1	0x001	SPD Revision	
2	0x002	Key Byte / DRAM Device Type	
3	0x003	Key Byte / Module Type	
4	0x004	SDRAM Density and Banks	3
5	0x005	SDRAM Addressing	3
6	0x006	Primary SDRAM Package Type	3
7	0x007	SDRAM Optional Features	3
8	0x008	SDRAM Thermal and Refresh Options	3
9	0x009	Other SDRAM Optional Features	3
10	0x00A	Secondary SDRAM Package Type	3
11	0x00B	Module Nominal Voltage, VDD	3
12	0x00C	Module Organization	
13	0x00D	Module Memory Bus Width	
14	0x00E	Module Thermal Sensor	
15	0x00F	Extended module type	
16	0x010	Reserved -- must be coded as 0x00	
17	0x011	Timebases	
18	0x012	SDRAM Minimum Cycle Time ($t_{CKAVGmin}$)	3
19	0x013	SDRAM Maximum Cycle Time ($t_{CKAVGmax}$)	3
20	0x014	CAS Latencies Supported, First Byte	3
21	0x015	CAS Latencies Supported, Second Byte	3

- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect

Byte Number		Function Described	Notes
0	0x000	Number of Serial PD Bytes Written / SPD Device Size	1, 2
1	0x001	SPD Revision	
2	0x002	Key Byte / DRAM Device Type	
3	0x003	Key Byte / Module Type	
4	0x004	SDRAM Density and Banks	3
5	0x005	SDRAM Addressing	3
6	0x006	Primary SDRAM Package Type	3

Byte 5 (0x005): SDRAM Addressing

This byte describes the row addressing and the column addressing in the SDRAM device. Bits 2~0 encode the number of column address bits, and bits 5~3 encode the number of row address bits. These values come from the DDR4 SDRAM data sheet.

Bits 7~6	Bits 5~3	Bits 2~0
Reserved	Row Address Bits	Column Address Bits
Reserved; must be coded as 00	Bit [5, 4, 3] : 000 = 12 001 = 13 010 = 14 011 = 15 100 = 16 101 = 17 110 = 18 All others reserved	Bit [2, 1, 0] : 000 = 9 001 = 10 010 = 11 011 = 12 All others reserved

17	0x011	Timebases	
18	0x012	SDRAM Minimum Cycle Time ($t_{CKAVGmin}$)	3
19	0x013	SDRAM Maximum Cycle Time ($t_{CKAVGmax}$)	3
20	0x014	CAS Latencies Supported, First Byte	3
21	0x015	CAS Latencies Supported, Second Byte	3

- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect

Byte Number		Function Described	Notes
0	0x000	Number of Serial PD Bytes Written / SPD Device Size	1, 2
1	0x001	SPD Revision	
2	0x002	Key Byte / DRAM Device Type	
3	0x003	Key Byte / Module Type	
4	0x004	SDRAM Density and Banks	3
5	0x005	SDRAM Addressing	3
6	0x006	Primary SDRAM Package Type	3

Byte 5 (0x005): SDRAM Addressing

This byte describes the row addressing and the column addressing in the SDRAM device. Bits 2~0 encode the number of column address bits, and bits 5~3 encode the number of row address bits. These values come from the DDR4 SDRAM data sheet.

Bits 7~6	Bits 5~3	Bits 2~0
Reserved	Row Address Bits	Column Address Bits
Reserved; must be coded as 00	Bit [5, 4, 3] : 000 = 12 001 = 13 010 = 14 011 = 15 100 = 16 101 = 17 110 = 18 All others reserved	Bit [2, 1, 0] : 000 = 9 001 = 10 010 = 11 011 = 12 All others reserved

17	0x011	Timebases	
18	0x012	SDRAM Minimum Cycle Time ($t_{CKAVGmin}$)	3
19	0x013	SDRAM Maximum Cycle Time ($t_{CKAVGmax}$)	3
20	0x014	CAS Latencies Supported, First Byte	3
21	0x015	CAS Latencies Supported, Second Byte	3

- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...

Serial Presence Detect

Byte Number		Function Described	Notes
0	0x000	Number of Serial PD Bytes Written / SPD Device Size	1, 2
1	0x001	SPD Revision	
2	0x002	Key Byte / DRAM Device Type	
3	0x003	Key Byte / Module Type	
4	0x004	SDRAM Density and Banks	3
5	0x005	SDRAM Addressing	3
6	0x006	Primary SDRAM Package Type	3

Byte 5 (0x005): SDRAM Addressing

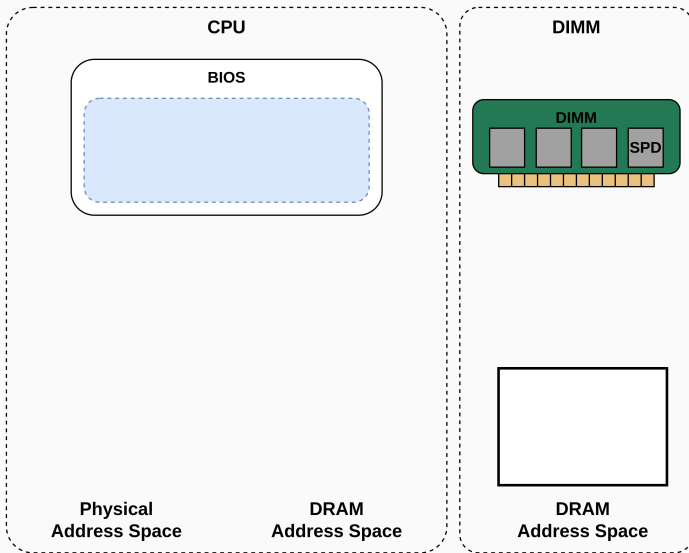
This byte describes the row addressing and the column addressing in the SDRAM device. Bits 2~0 encode the number of column address bits, and bits 5~3 encode the number of row address bits. These values come from the DDR4 SDRAM data sheet.

Bits 7~6	Bits 5~3	Bits 2~0
Reserved	Row Address Bits	Column Address Bits
Reserved; must be coded as 00	Bit [5, 4, 3] : 000 = 12 001 = 13 010 = 14 011 = 15 100 = 16 101 = 17 110 = 18 All others reserved	Bit [2, 1, 0] : 000 = 9 001 = 10 010 = 11 011 = 12 All others reserved

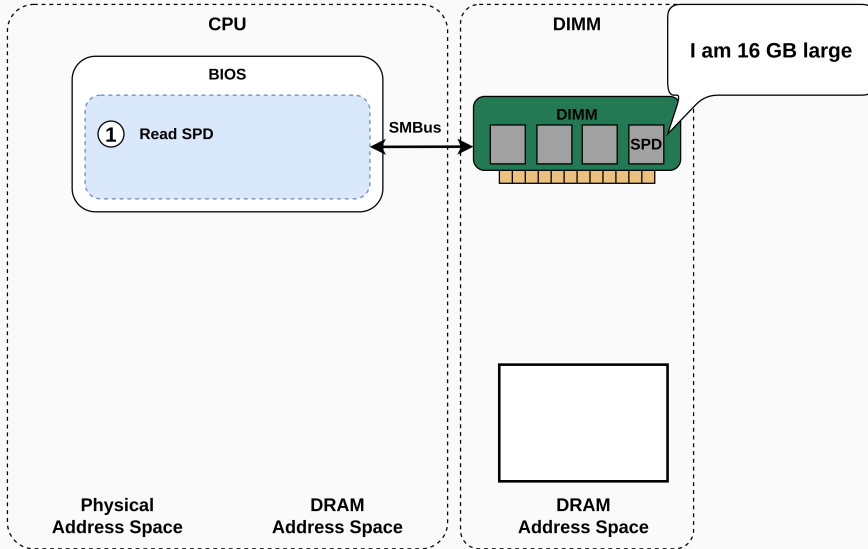
17	0x011	Timebases	
18	0x012	SDRAM Minimum Cycle Time ($t_{CKAVGmin}$)	3
19	0x013	SDRAM Maximum Cycle Time ($t_{CKAVGmax}$)	3
20	0x014	CAS Latencies Supported, First Byte	3
21	0x015	CAS Latencies Supported, Second Byte	3

- EEPROM
- Contains DIMM configuration
 - Size
 - Speed
 - Manufacturer
 - ...
- *What if we overwrite this data?*

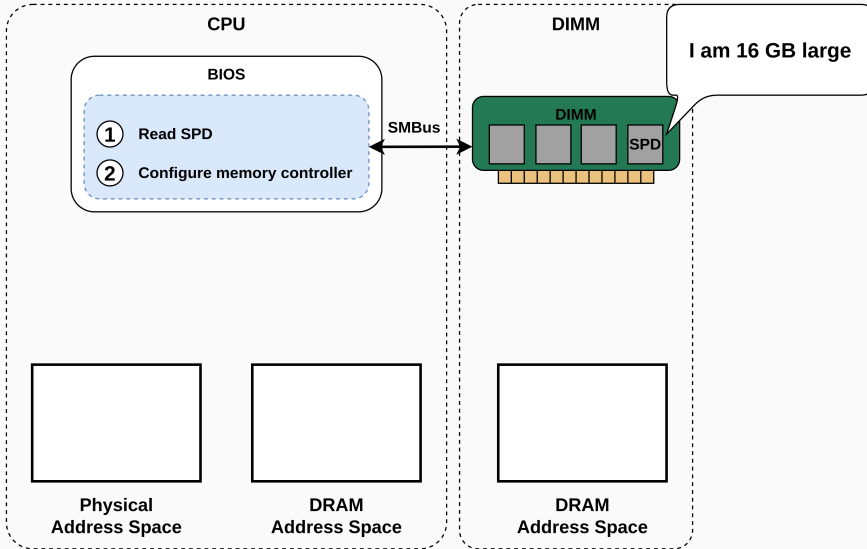
Introducing Aliases



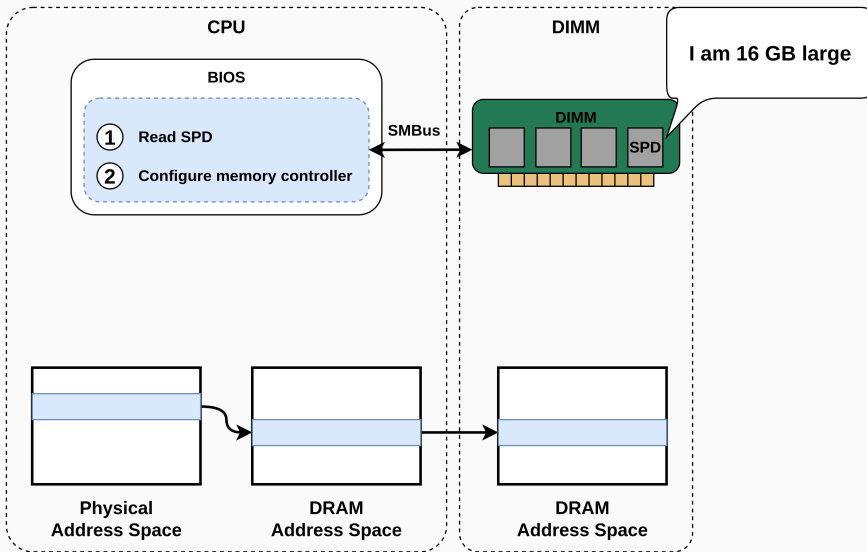
Introducing Aliases



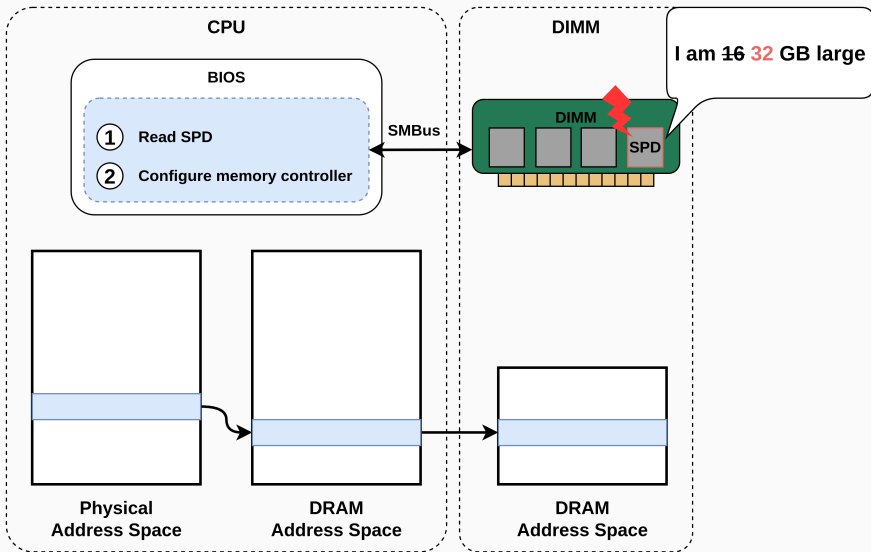
Introducing Aliases



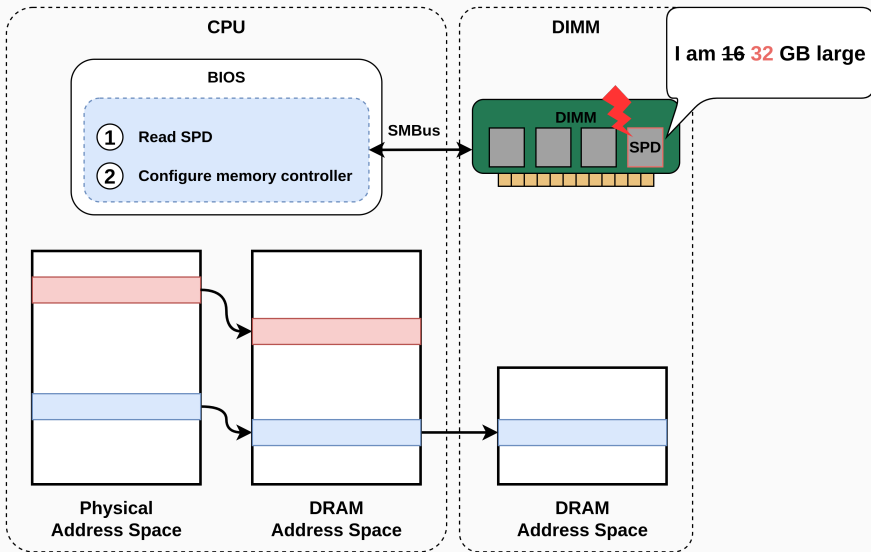
Introducing Aliases



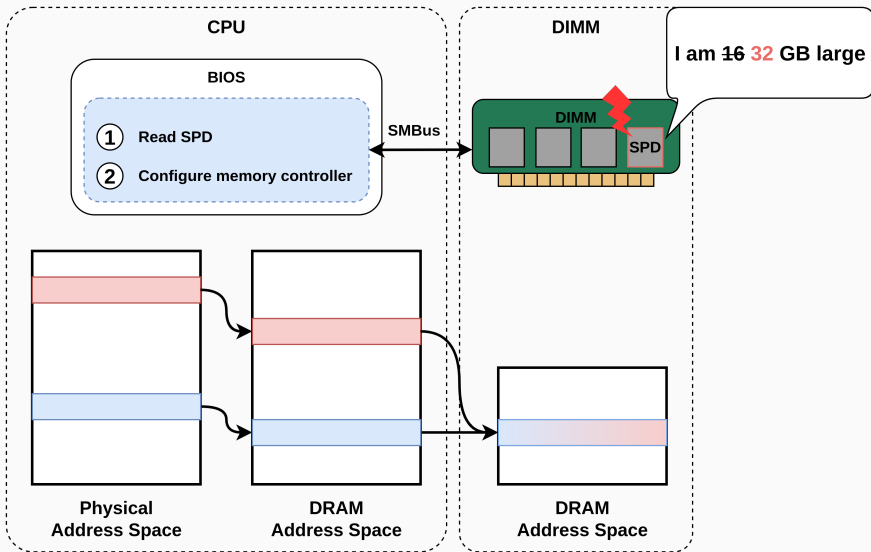
Introducing Aliases



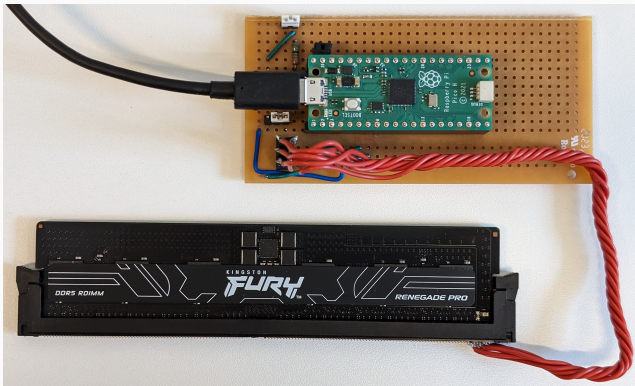
Introducing Aliases



Introducing Aliases

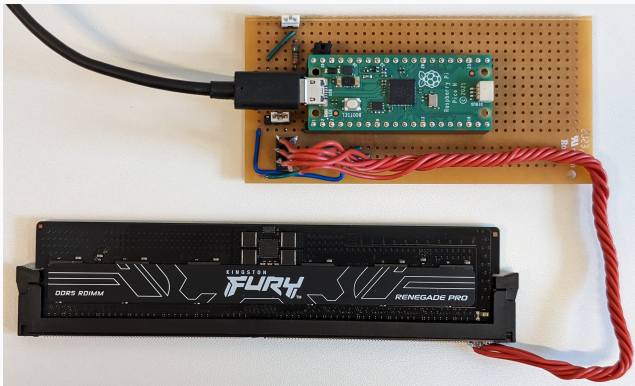


Modifying SPD — Through Hardware



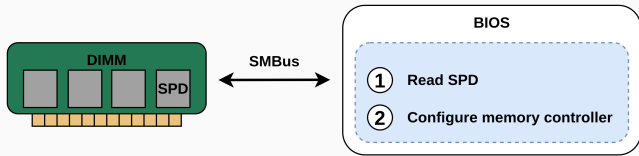
- I²C pins exposed on DIMM
- Trivial to unlock and overwrite

Modifying SPD — Through Hardware



- I²C pins exposed on DIMM
- Trivial to unlock and overwrite
- Total cost: ~10\$

Modifying SPD — Through Software



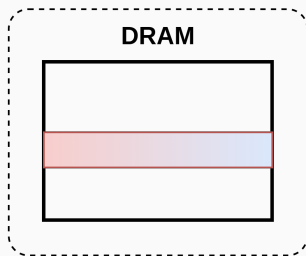
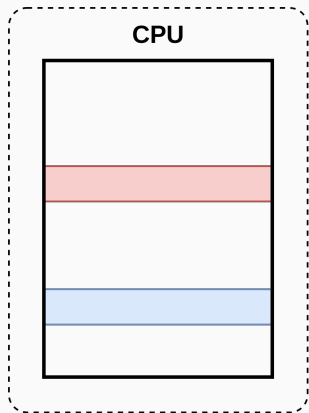
- BIOS
 - Spoof SPD contents

Modifying SPD — Through Software

	Manufacturer	Type	Protected
DDR4	Corsair	UDIMM	✗
	Crucial	SODIMM	✓
	G.SKILL	UDIMM	✗
	Kingston	RDIMM	✓
	Micron	RDIMM	✓
	Samsung	UDIMM	✓
	Samsung	SODIMM	✓
	SK hynix	UDIMM	✓
	SK hynix	RDIMM	✓
	SK hynix	SODIMM	✓
DDR5	Kingston	RDIMM	✓
	Samsung	RDIMM	✓
	SK hynix	RDIMM	✓
	SK hynix	UDIMM	✓

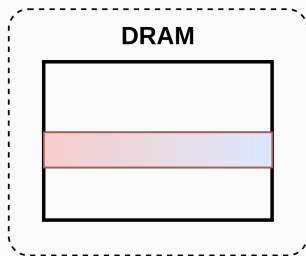
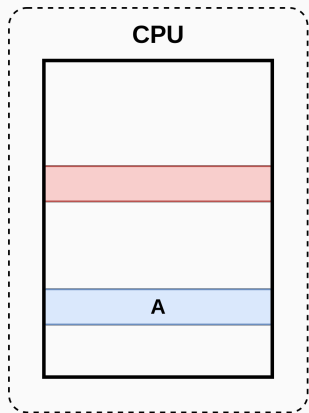
- BIOS
 - Spoof SPD contents
- SMBus & SidebandBus
 - Overwrite SPD contents
 - Unlocked DIMMs

Aliasing with Static Memory Encryption



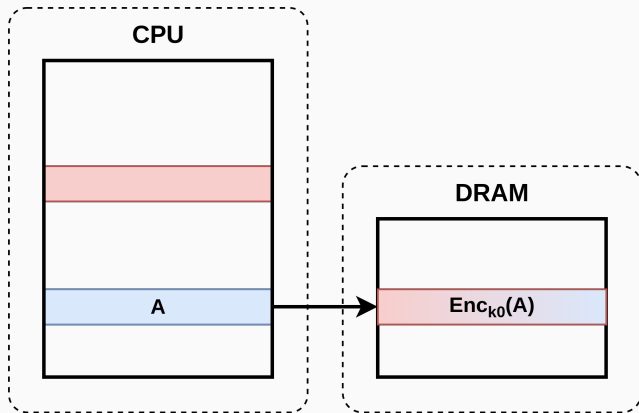
- Ciphertext access through alias

Aliasing with Static Memory Encryption



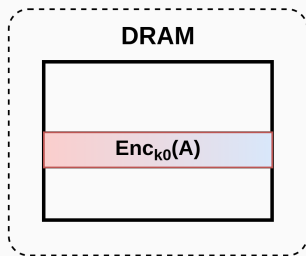
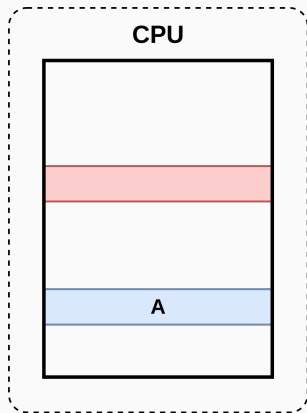
- Ciphertext access through alias

Aliasing with Static Memory Encryption



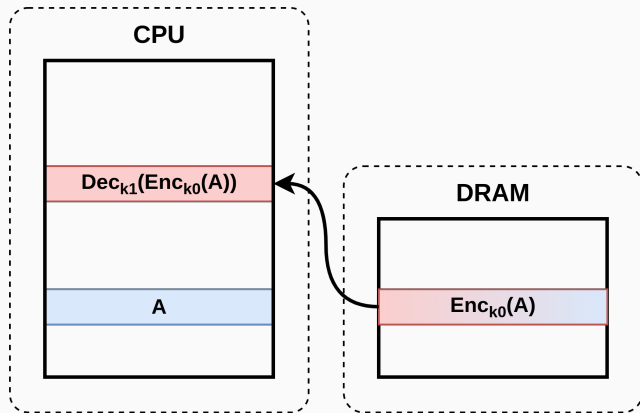
- Ciphertext access through alias

Aliasing with Static Memory Encryption



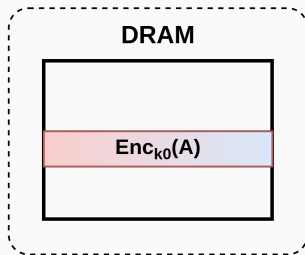
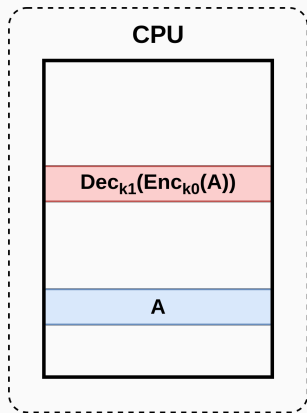
- Ciphertext access through alias

Aliasing with Static Memory Encryption



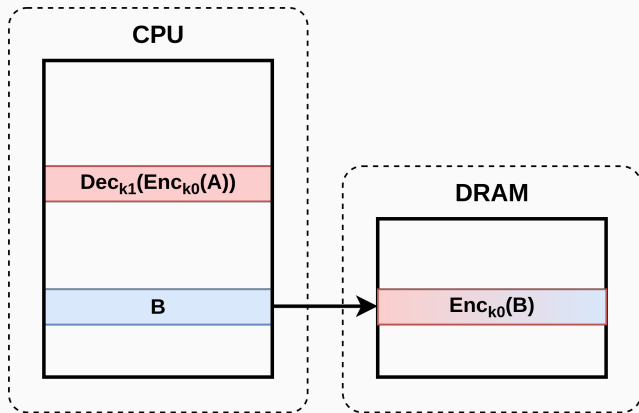
- Ciphertext access through alias

Aliasing with Static Memory Encryption



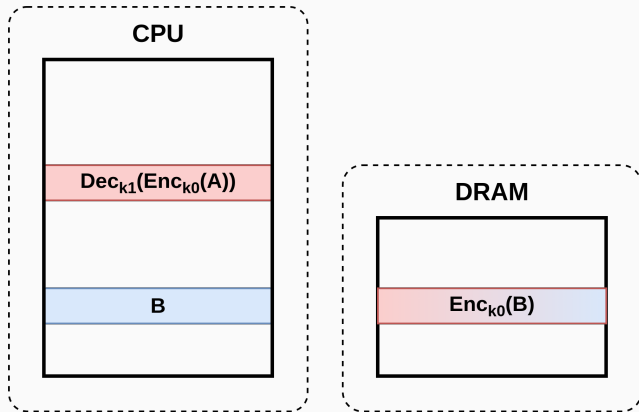
- Ciphertext access through alias

Aliasing with Static Memory Encryption



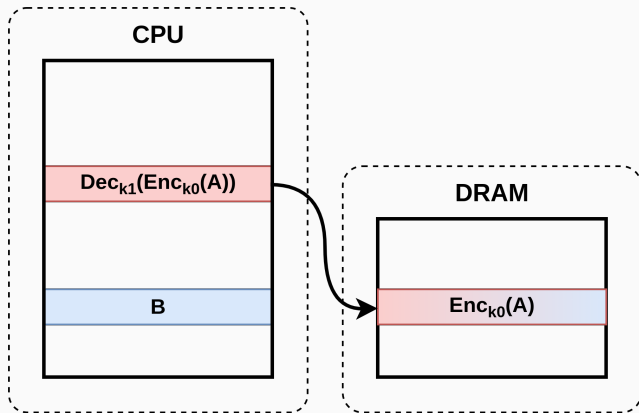
- Ciphertext access through alias

Aliasing with Static Memory Encryption



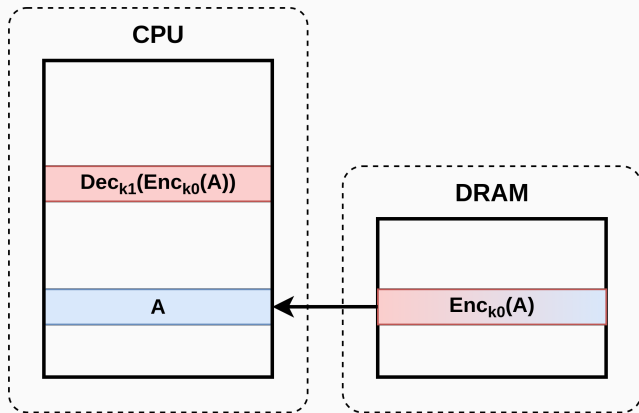
- Ciphertext access through alias
- Static encryption enables replay

Aliasing with Static Memory Encryption



- Ciphertext access through alias
- Static encryption enables replay

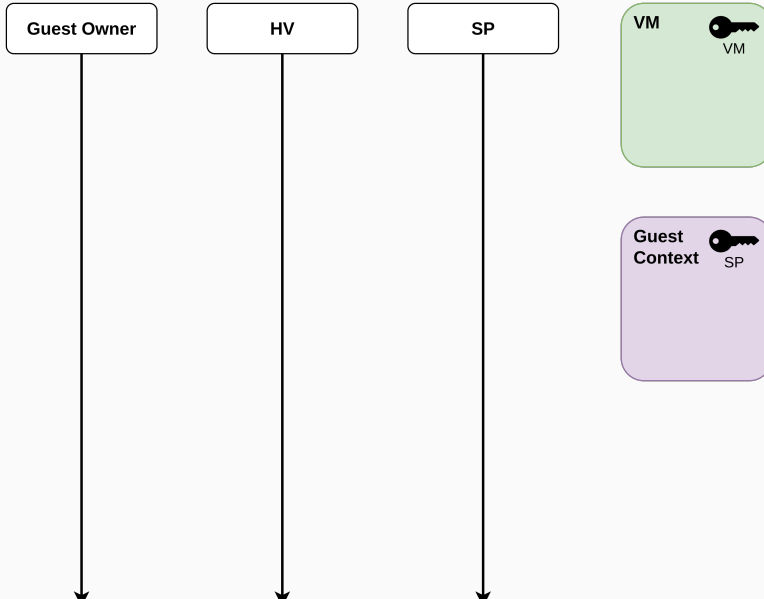
Aliasing with Static Memory Encryption



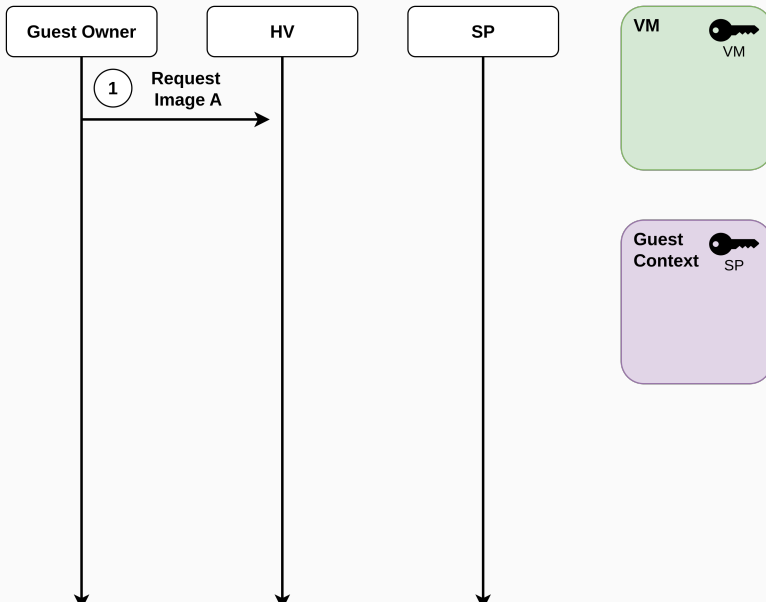
- Ciphertext access through alias
- Static encryption enables replay

Demo Time

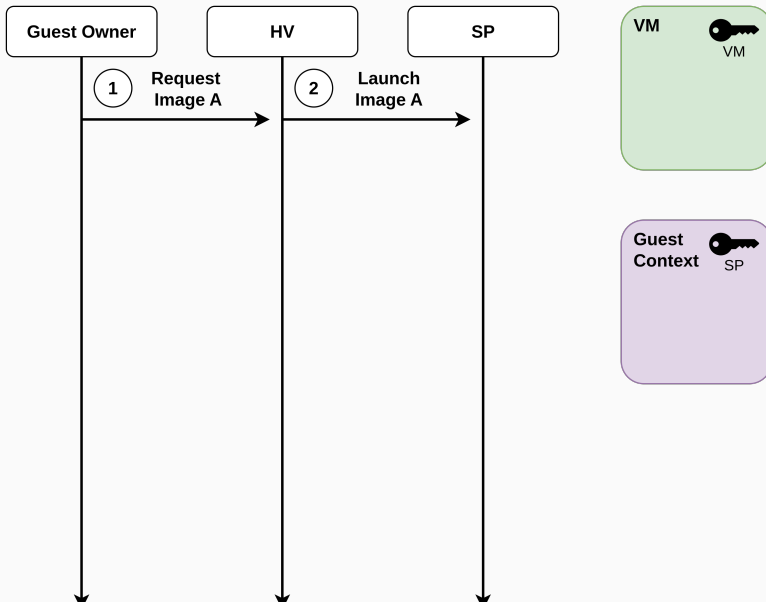
SEV-SNP Attestation Attack: Phase 1



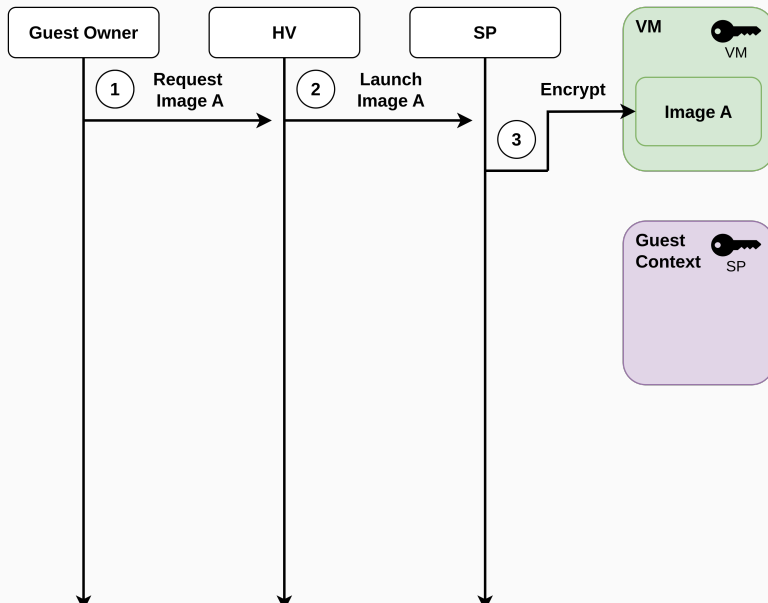
SEV-SNP Attestation Attack: Phase 1



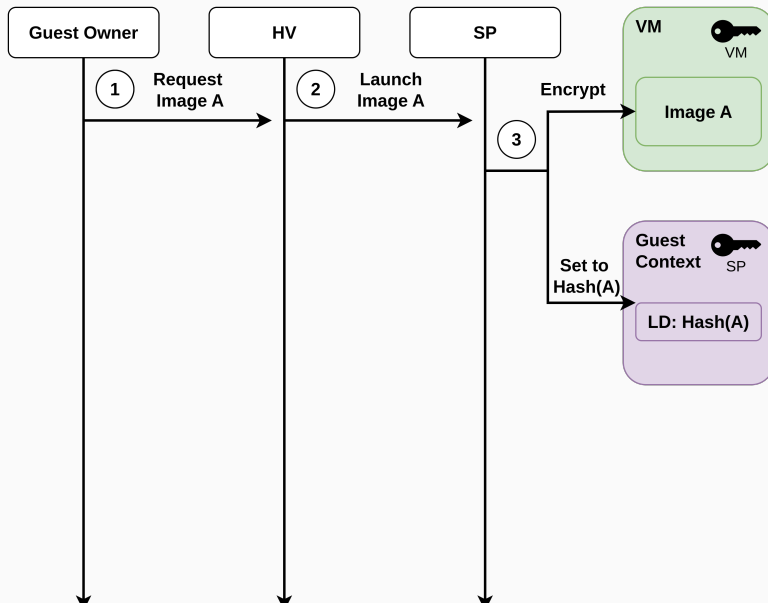
SEV-SNP Attestation Attack: Phase 1



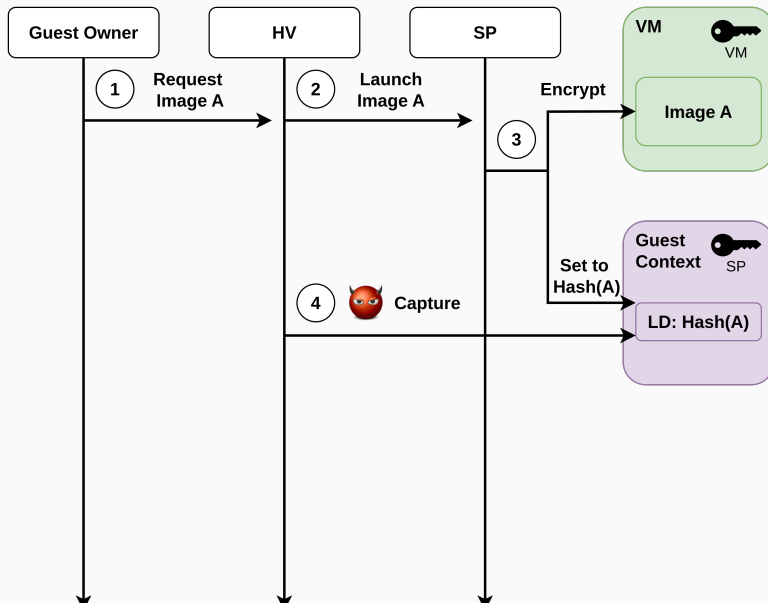
SEV-SNP Attestation Attack: Phase 1



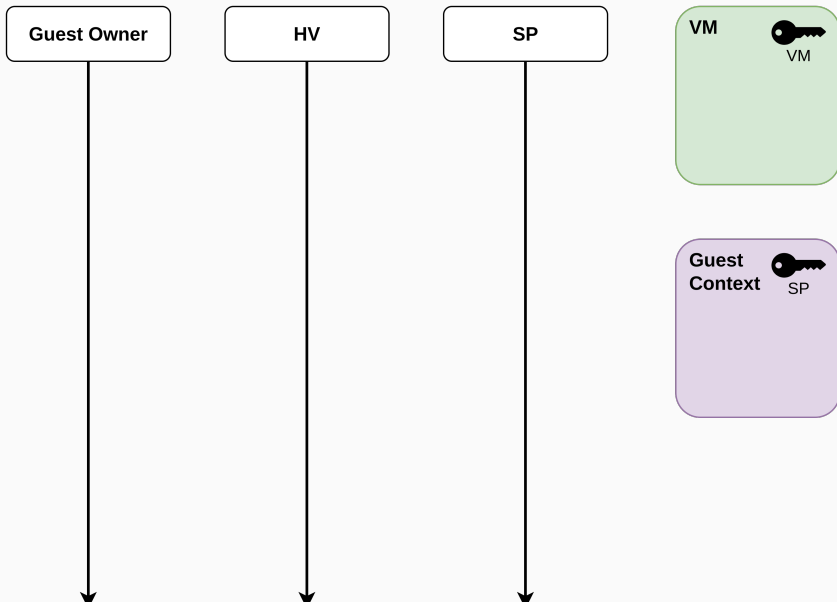
SEV-SNP Attestation Attack: Phase 1



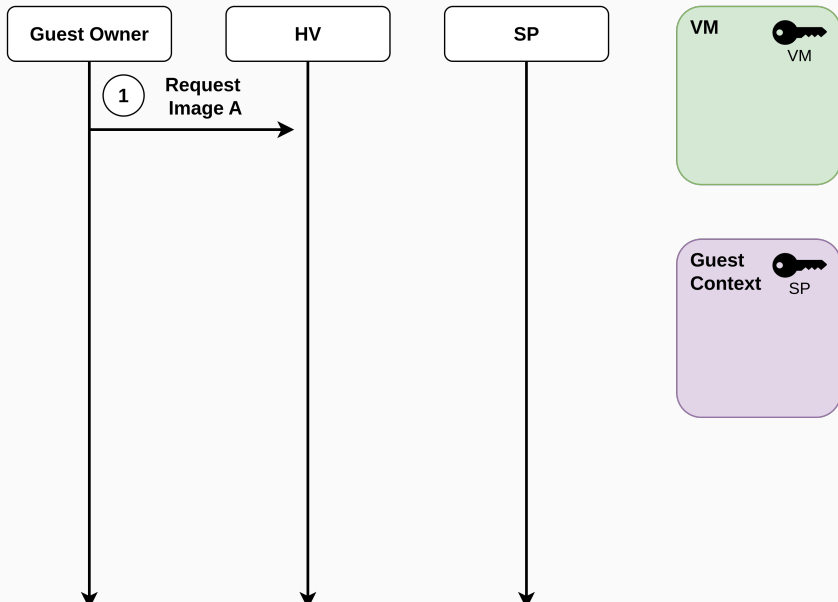
SEV-SNP Attestation Attack: Phase 1



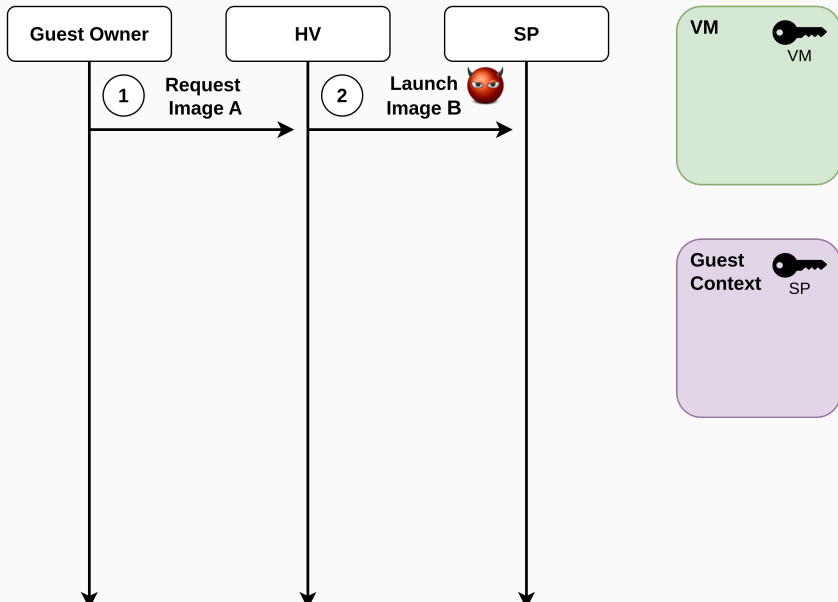
SEV-SNP Attestation Attack: Phase 2



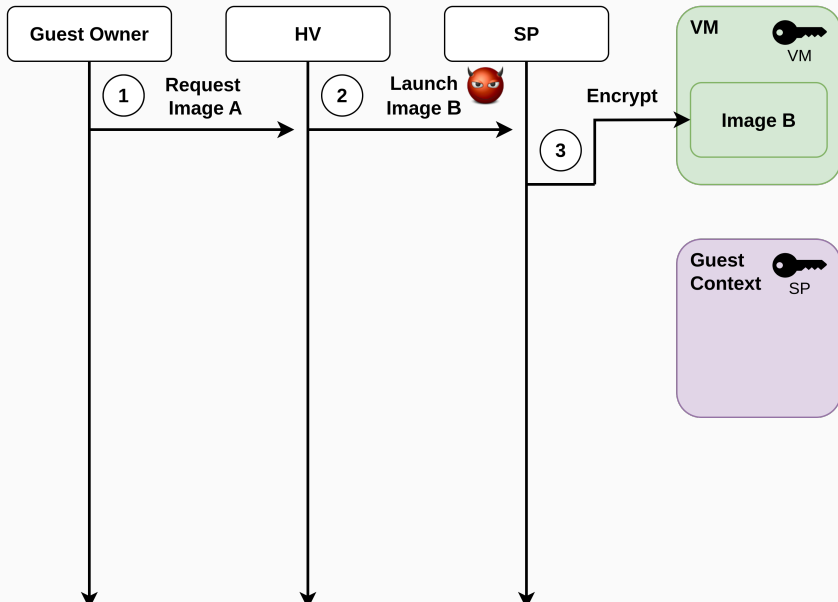
SEV-SNP Attestation Attack: Phase 2



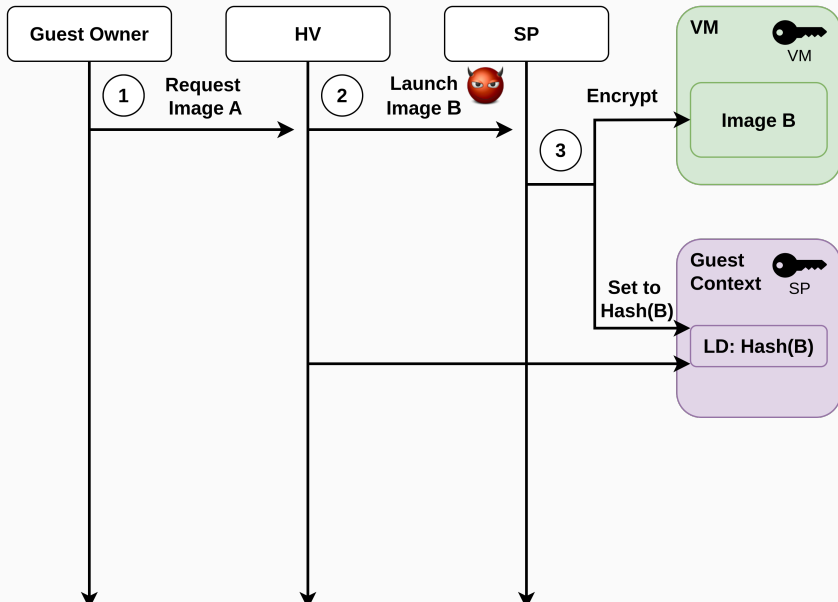
SEV-SNP Attestation Attack: Phase 2



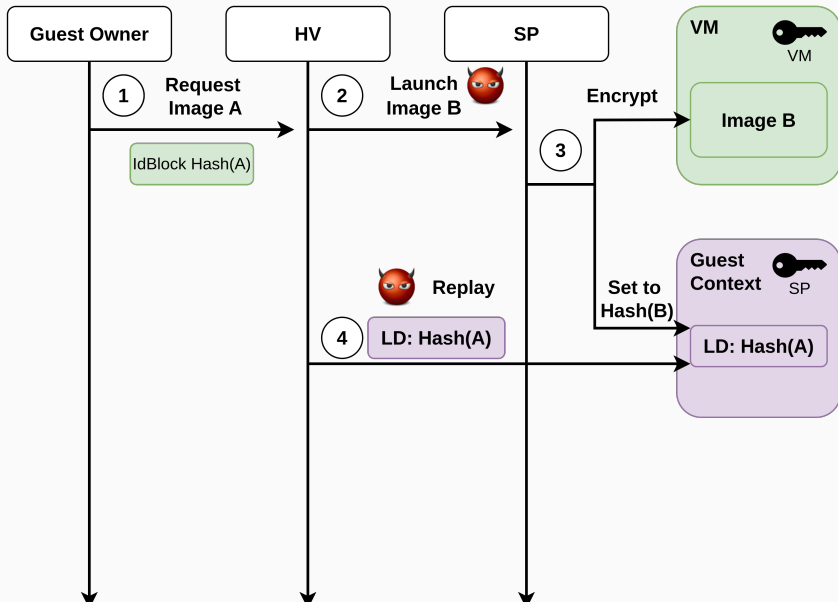
SEV-SNP Attestation Attack: Phase 2



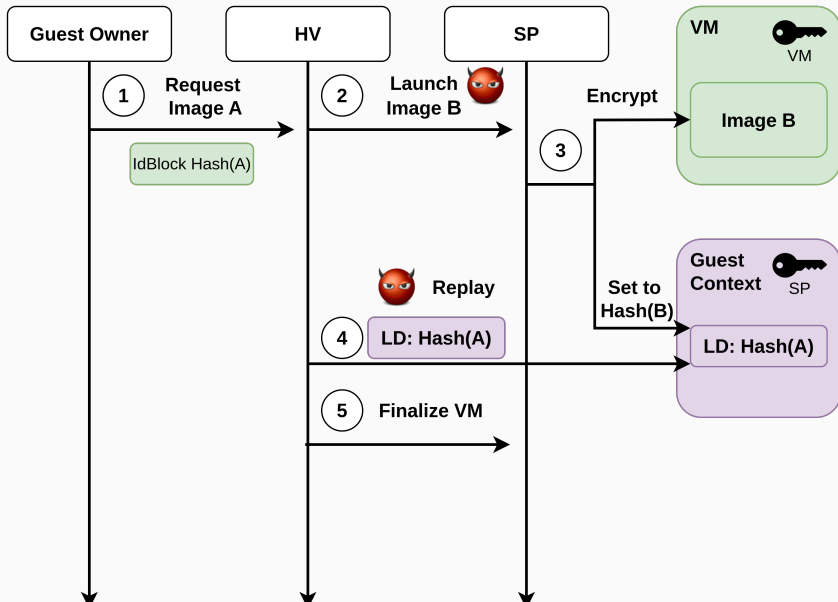
SEV-SNP Attestation Attack: Phase 2



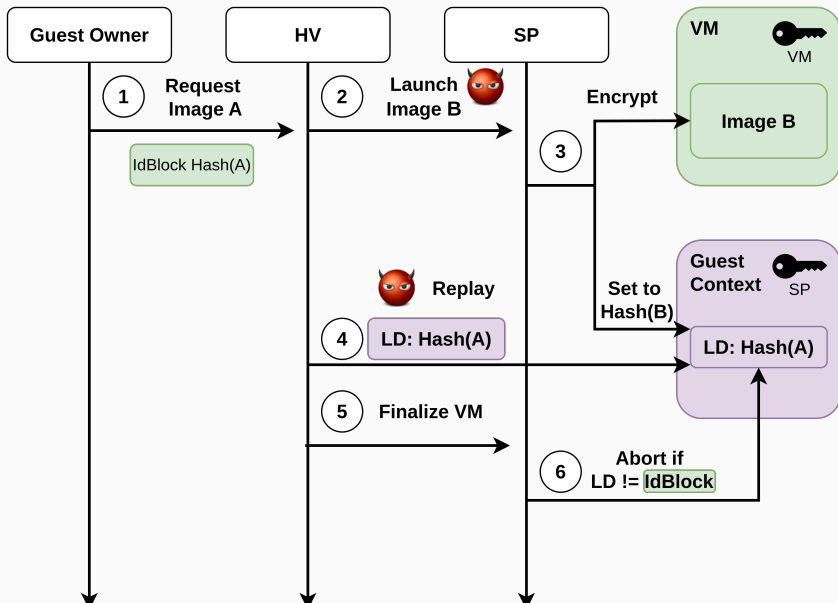
SEV-SNP Attestation Attack: Phase 2



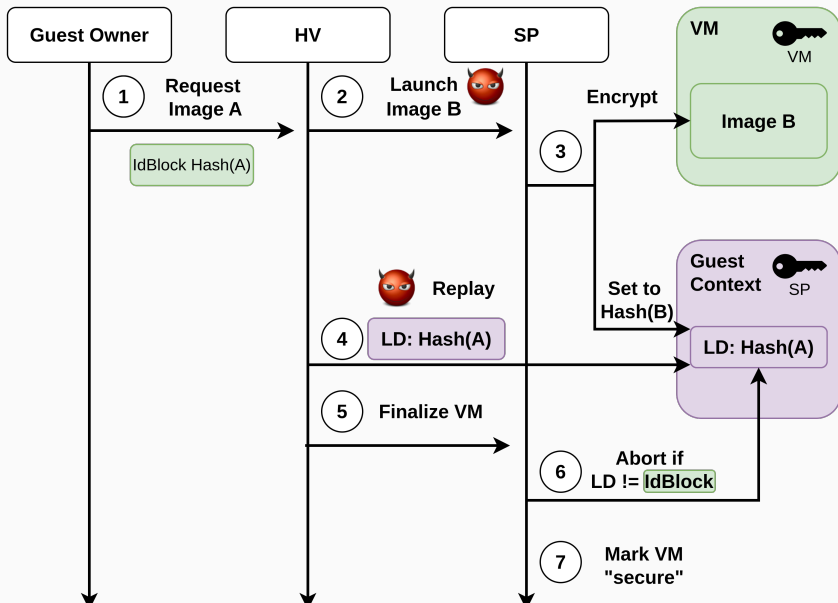
SEV-SNP Attestation Attack: Phase 2



SEV-SNP Attestation Attack: Phase 2



SEV-SNP Attestation Attack: Phase 2



Demo Time

1. Alias Checking^{1,2}

2. ECC-based MAC/Owner bit¹

¹Intel

²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

¹Intel

²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

```
1: for each page  $\mathcal{A}$  do  
2:   write_mem( $\mathcal{A}$ , marker)  
3:   for each page  $\mathcal{B} \neq \mathcal{A}$  do  
4:     if read_mem( $\mathcal{B}$ ) = marker then  
5:       Terminate System  
6:     end if  
7:   end for  
8: end for
```

¹Intel

²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

- Idea: Store metadata in ECC bits
- **Owner bit** Mark TDX/SGX pages
- **MAC** integrity protection

¹Intel

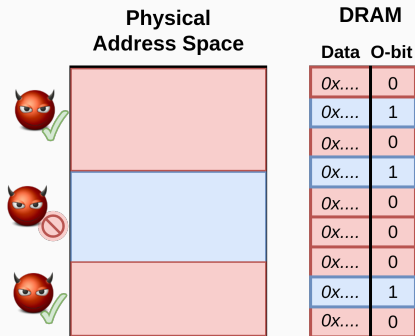
²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

- Idea: Store metadata in ECC bits
- **Owner bit** Mark TDX/SGX pages
- **MAC** integrity protection



¹Intel

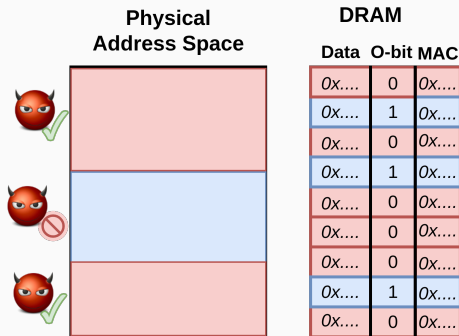
²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

- Idea: Store metadata in ECC bits
- **Owner bit** Mark TDX/SGX pages
- **MAC** integrity protection



¹Intel

²AMD

1. Alias Checking^{1,2}

- Idea: Search for aliases at boot time
- Optimized: $\mathcal{O}(\log_2(\text{mem. size}))$
- TOCTOU?

2. ECC-based MAC/Owner bit¹

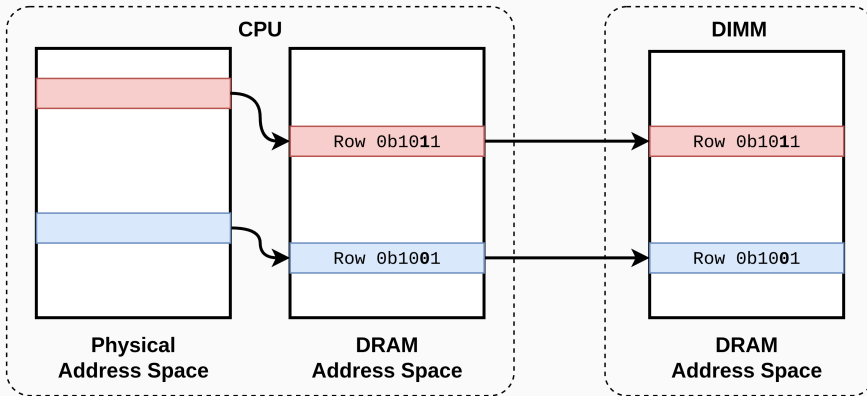
- Idea: Store metadata in ECC bits
- **Owner bit** Mark TDX/SGX pages
- **MAC** integrity protection

¹Intel

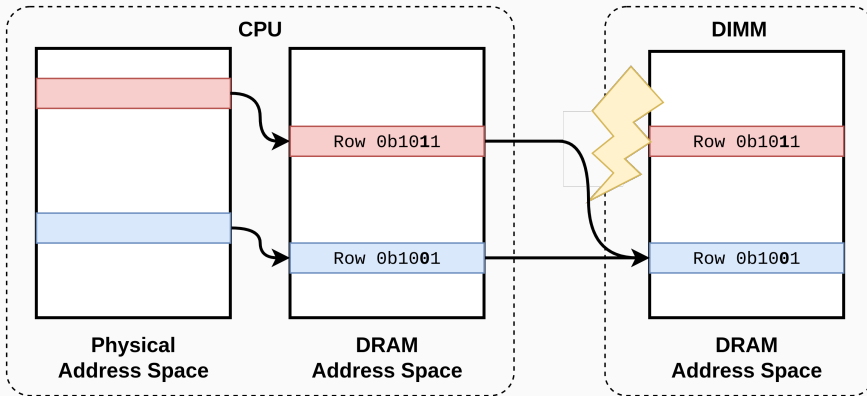
²AMD

Is it possible to introduce aliases at runtime?

Introducing Dynamic Aliases



Introducing Dynamic Aliases



DRAM Command Encoding

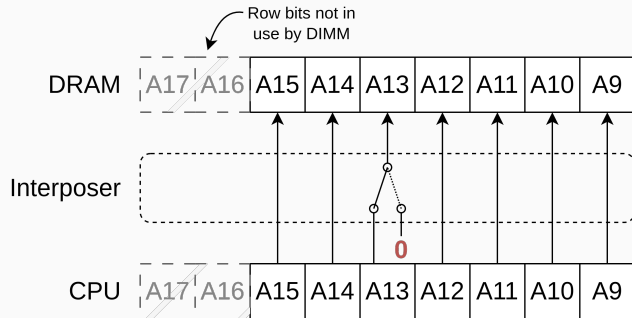
	$\overline{CS}$	$\overline{ACT}$	BG	BA	CID	A17	A16	A15	A14	A13	A12	A11	A10	A9-0
ACT	L	L	Bank		CID	Row								
MRS	L	H	Reg		V	RFU	L	L	L	OP				
REF	L	H	V	V	CID	V	L	L	H	V	V	V	V	V
PRE	L	H	Bank		CID	V	L	H	L	V	V	V	L	V
PREA	L	H	V	V	CID	V	L	H	L	V	V	V	H	V
RFU	L	H	RFU				L	H	H	RFU				
WR	L	H	Bank		CID	V	H	L	L	V	$\overline{BC}$	V	AP	CA
RD	L	H	Bank		CID	V	H	L	H	V	$\overline{BC}$	V	AP	CA
ZQCL	L	H	V	V	V	V	H	H	L	V	V	V	H	V
ZQCS	L	H	V	V	V	V	H	H	L	V	V	V	L	V
NOP	L	H	V	V	V	V	H	H	H	V	V	V	V	V
DES	H	X	X	X	X	X	X	X	X	X	X	X	X	X

DRAM Command Encoding

	$\overline{CS}$	$\overline{ACT}$	BG	BA	CID	A17	A16	A15	A14	A13	A12	A11	A10	A9-0
ACT	L	L	Bank		CID		Row							
MRS	L	H	Reg		V	RFU	L	L	L			OP		
REF	L	H	V	V	CID	V	L	L	H	V	V	V	V	V
PRE	L	H	Bank		CID	V	L	H	L	V	V	V	L	V
PREA	L	H	V	V	CID	V	L	H	L	V	V	V	H	V
RFU	L	H	RFU				L	H	H			RFU		
WR	L	H	Bank		CID	V	H	L	L	V	$\overline{BC}$	V	AP	CA
RD	L	H	Bank		CID	V	H	L	H	V	$\overline{BC}$	V	AP	CA
ZQCL	L	H	V	V	V	V	H	H	L	V	V	V	H	V
ZQCS	L	H	V	V	V	V	H	H	L	V	V	V	L	V
NOP	L	H	V	V	V	V	H	H	H	V	V	V	V	V
DES	H	X	X	X	X	X	X	X	X	X	X	X	X	X

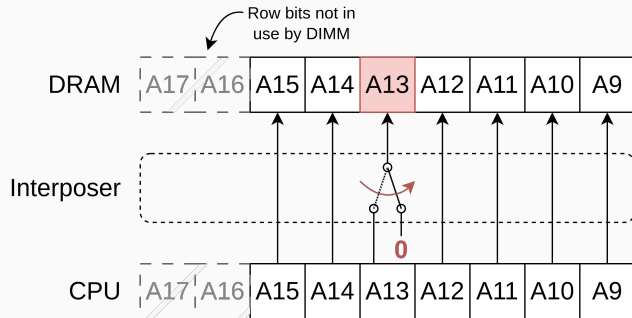
- Some address bits *only* used for row address

Introducing Dynamic Memory Aliases



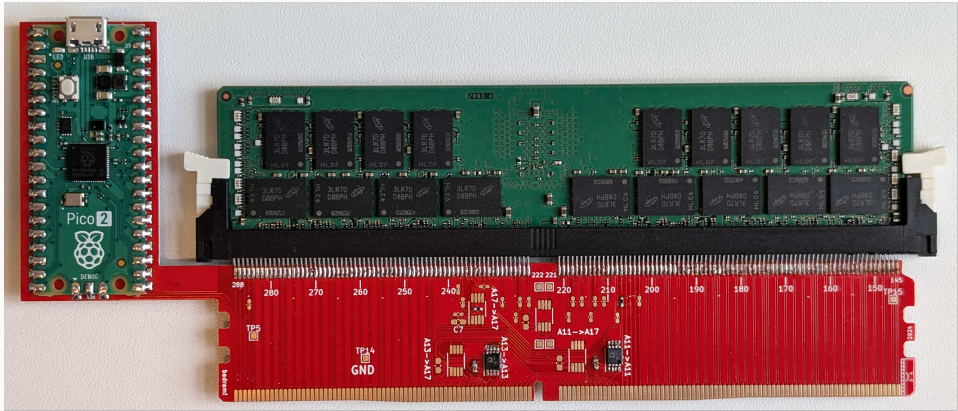
- Dynamically switch line to GND
- Boot-time: **Inactive**
 - Pass boot-time checks

Introducing Dynamic Memory Aliases

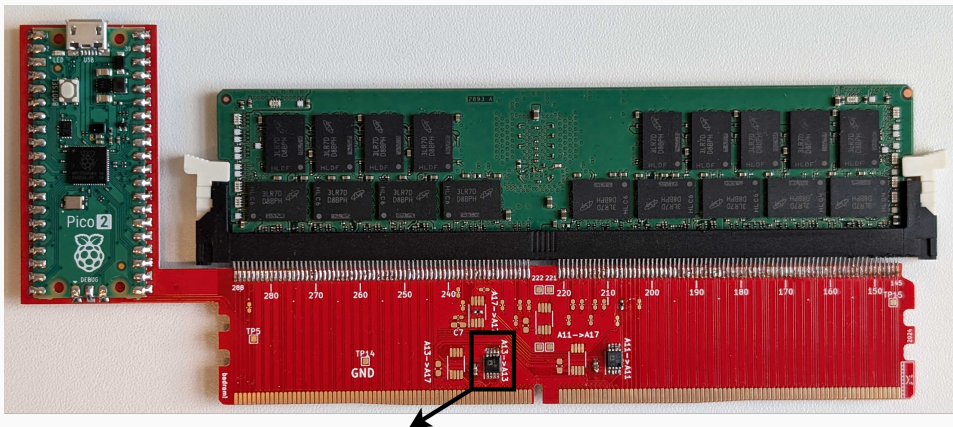


- Dynamically switch line to GND
- Boot-time: Inactive
 - Pass boot-time checks
- Attack: **Active**
 - Introduce memory aliases

Introducing Dynamic Memory Aliases

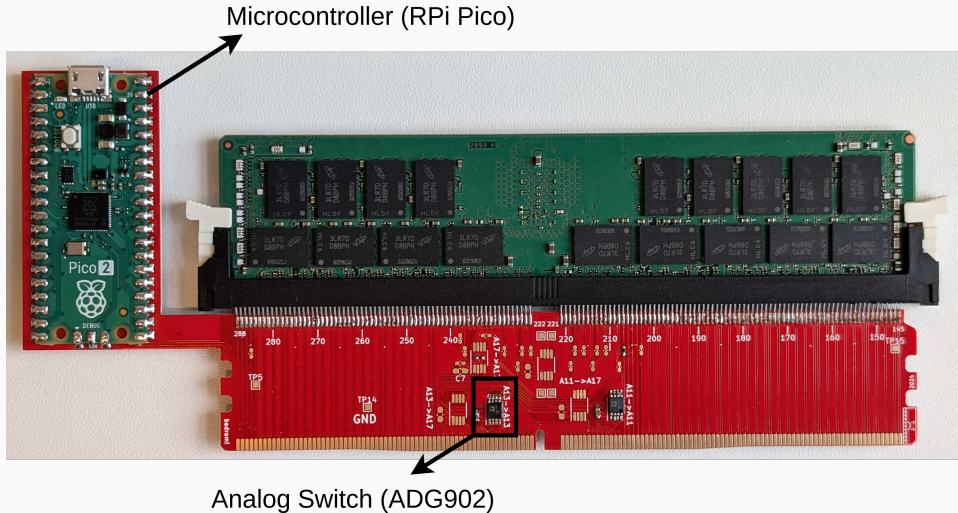


Introducing Dynamic Memory Aliases



Analog Switch (ADG902)

Introducing Dynamic Memory Aliases



Interposer evaluation



- Stability
 - Works at any DDR4 frequency

Interposer evaluation



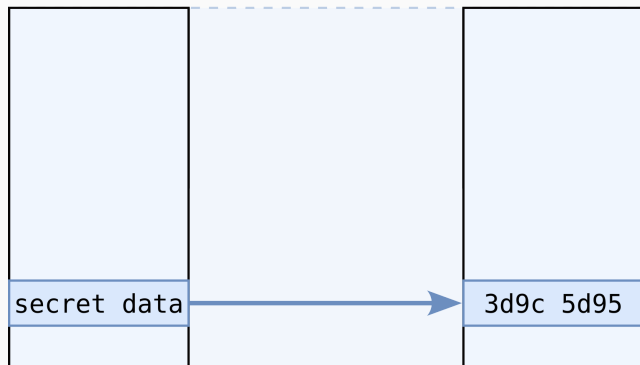
- Stability
 - Works at any DDR4 frequency
- Feasibility
 - Entirely invisible to CPU
 - Deterministic memory aliasing

Interposer evaluation

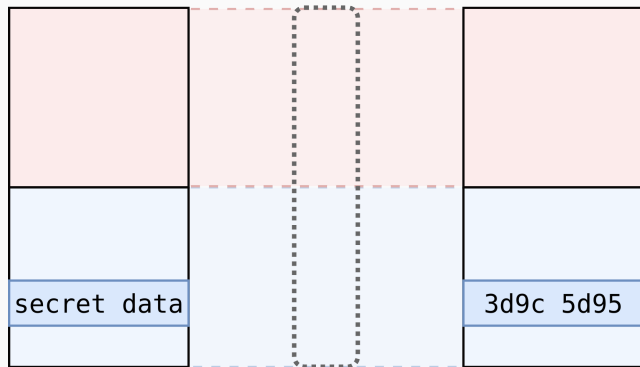


- Stability
 - Works at any DDR4 frequency
- Feasibility
 - Entirely invisible to CPU
 - Deterministic memory aliasing
- Limitations
 - Creates unstable memory regions
 - Memory interleaving complicates attack

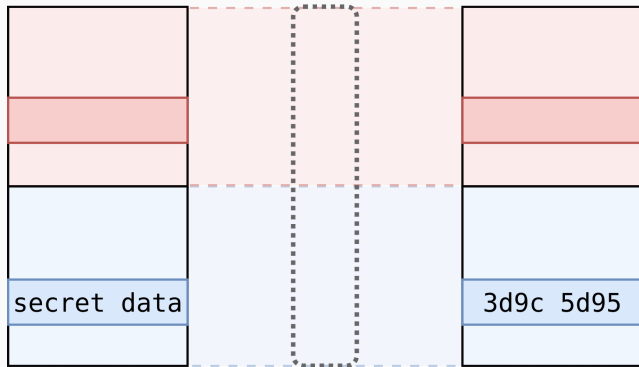
Breaking Intel Scalable SGX — Arbitrary Plaintext Access



Breaking Intel Scalable SGX — Arbitrary Plaintext Access

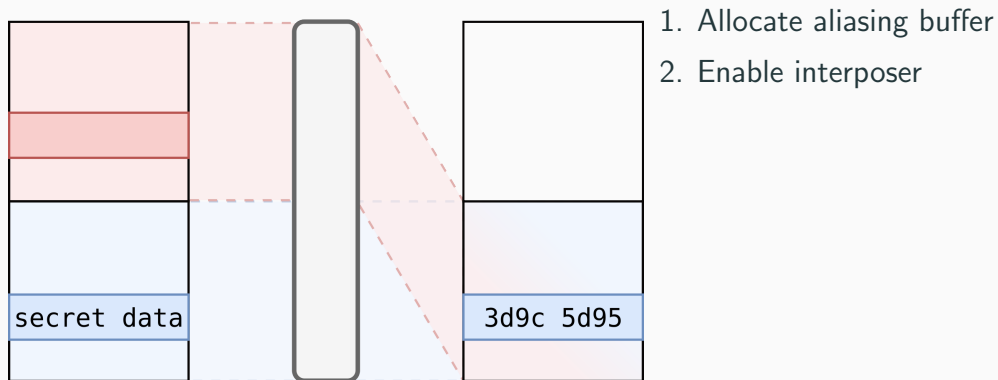


Breaking Intel Scalable SGX — Arbitrary Plaintext Access

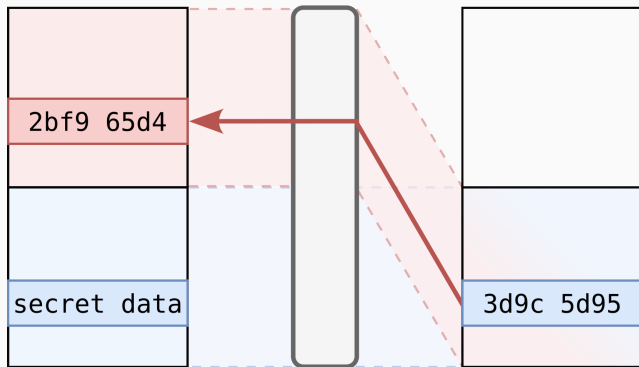


1. Allocate aliasing buffer

Breaking Intel Scalable SGX — Arbitrary Plaintext Access

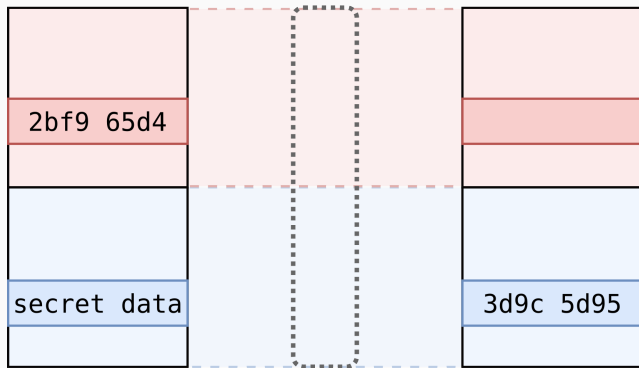


Breaking Intel Scalable SGX — Arbitrary Plaintext Access



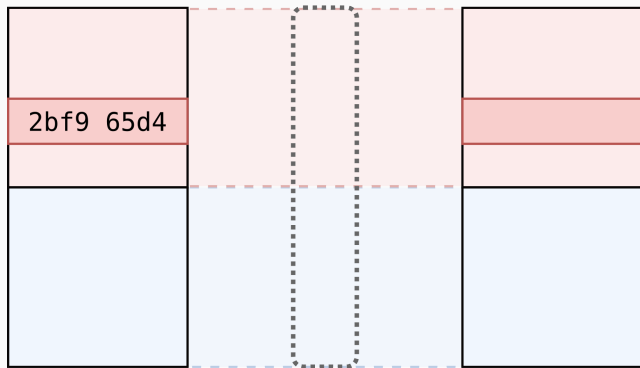
1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext

Breaking Intel Scalable SGX — Arbitrary Plaintext Access



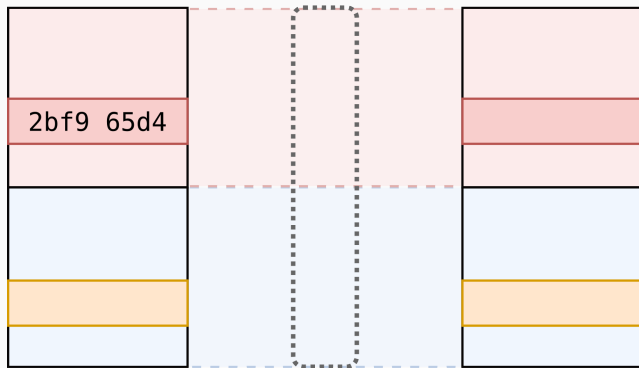
1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext
4. Disable interposer

Breaking Intel Scalable SGX — Arbitrary Plaintext Access



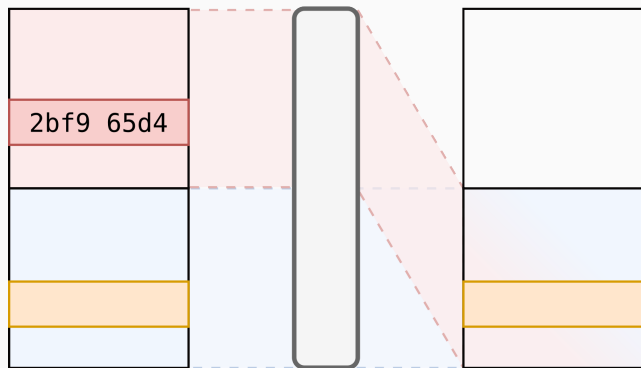
1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext
4. Disable interposer

Breaking Intel Scalable SGX — Arbitrary Plaintext Access



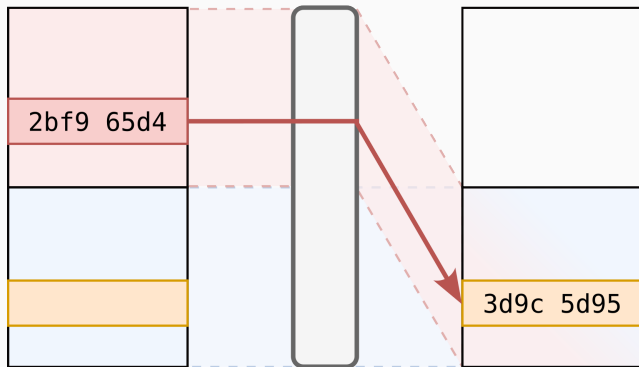
1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext
4. Disable interposer
5. Allocate buffer at victim PA

Breaking Intel Scalable SGX — Arbitrary Plaintext Access



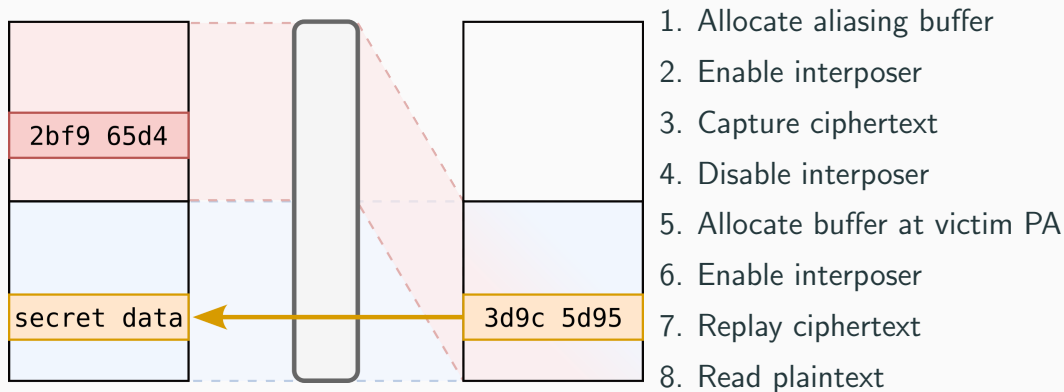
1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext
4. Disable interposer
5. Allocate buffer at victim PA
6. Enable interposer

Breaking Intel Scalable SGX — Arbitrary Plaintext Access



1. Allocate aliasing buffer
2. Enable interposer
3. Capture ciphertext
4. Disable interposer
5. Allocate buffer at victim PA
6. Enable interposer
7. Replay ciphertext

Breaking Intel Scalable SGX — Arbitrary Plaintext Access

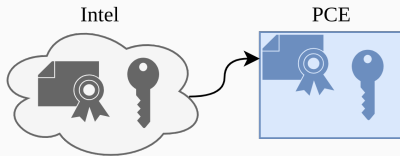


Breaking Intel Scalable SGX — Extracting Provisioning Key



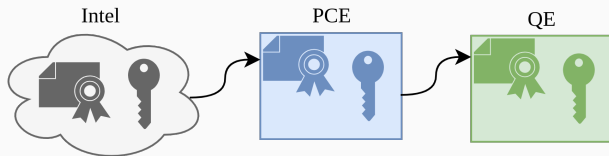
- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key

Breaking Intel Scalable SGX — Extracting Provisioning Key



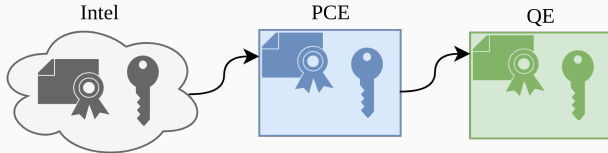
- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key

Breaking Intel Scalable SGX — Extracting Provisioning Key



- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key
 - Responsible for Quoting Enclave Certification

Breaking Intel Scalable SGX — Extracting Provisioning Key



PCK derivation

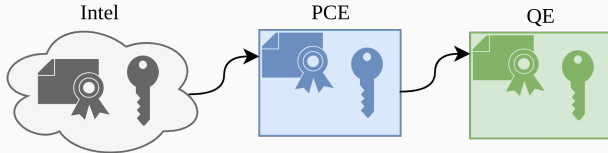
```
get_pce_priv_key
get_provision_key(&tmp, psvn);

// Seed generation
sgx_rijndael128_cmac_msg(&tmp, &seed);
...

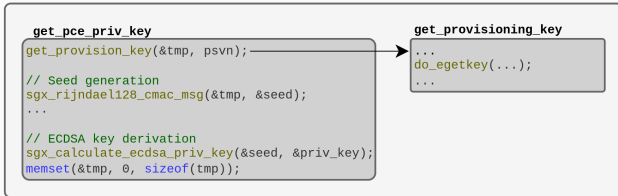
// ECDSA key derivation
sgx_calculate_ecdsa_priv_key(&seed, &priv_key);
memset(&tmp, 0, sizeof(tmp));
```

- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key
 - Responsible for Quoting Enclave Certification

Breaking Intel Scalable SGX — Extracting Provisioning Key

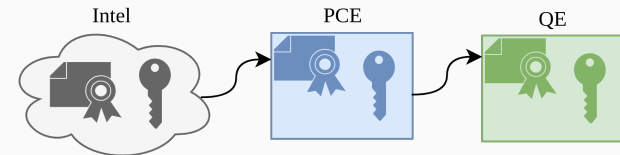


PCK derivation

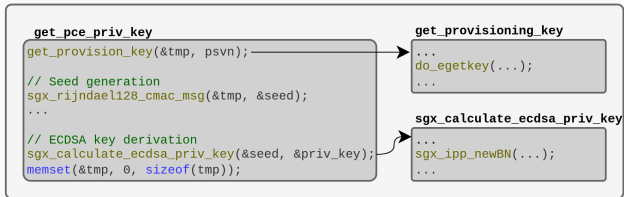


- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key
 - Responsible for Quoting Enclave Certification

Breaking Intel Scalable SGX — Extracting Provisioning Key

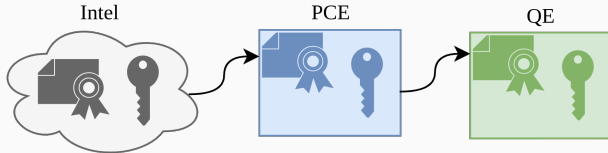


PCK derivation

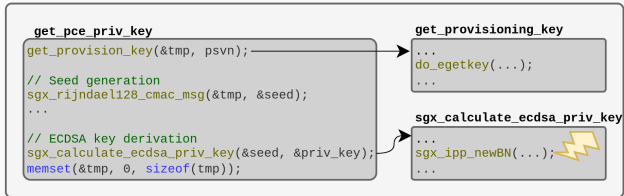


- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key
 - Responsible for Quoting Enclave Certification

Breaking Intel Scalable SGX — Extracting Provisioning Key



PCK derivation



- Data Center Attestation Primitives
- Provisioning Certification Enclave (PCE)
 - Provisioning Certification Key, derived from hardware key
 - Responsible for Quoting Enclave Certification
- Arbitrary plaintext access to extract hardware provisioning key

- Strong Crypto
 - Abandoned by Intel, AMD, and Arm
- Highly Integrated Memory
 - Inflexible, Size Constraints

Summary

- **BadRAM creates aliases** in physical address space
 - One-time physical access to DIMM
 - Total cost: ~10\$



Summary

- **BadRAM creates aliases** in physical address space
 - One-time physical access to DIMM
 - Total cost: ~10\$
- E2E attack: **Break SEV-SNP attestation**



Summary

- **BadRAM creates aliases** in physical address space
 - One-time physical access to DIMM
 - Total cost: ~10\$
- E2E attack: **Break SEV-SNP attestation**
- Deployed Countermeasures
 - Alias check: Scalable SGX, TDX, **SEV-SNP (new)**
 - ECC metadata: Scalable SGX, TDX



- **BadRAM creates aliases** in physical address space
 - One-time physical access to DIMM
 - Total cost: ~10\$
- E2E attack: **Break SEV-SNP attestation**
- Deployed Countermeasures
 - Alias check: Scalable SGX, TDX, **SEV-SNP (new)**
 - ECC metadata: Scalable SGX, TDX
- Interposers **bypass boot-time mitigations**
 - **Plaintext access** on Intel Scalable SGX



When Memory Lies

Breaching Processor Security via Rogue
Memory Modules

Jesse De Meulemeester* Luca Wilke*

David Oswald Thomas Eisenbarth

Ingrid Verbauwhede Jo Van Bulck

* Equal Contribution



UNIVERSITY OF
BIRMINGHAM

DistrINet