Why SGX design flaws hinder its application in cloud computing

Daniel Gruss
06.11.2017

Graz University of Technology
Malware in SGX
• Ledger SGX Enclave for blockchain applications
• BitPay Copay Bitcoin wallet
• Teechain payment channel using SGX
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

### Teechain

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]
\[ M = C^d \mod n \]
Signatures (RSA)

\[ M = C^d \mod n \]

Result = C
Signatures (RSA)

\[ M = C^d \mod n \]

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \ldots
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result} \times C
\]

\[ \text{square} \quad \text{multiply} \]
Signatures (RSA)

\[ M = C^d \mod n \]

Result = Result \times Result

\[ \text{square} \]

Daniel Gruss — Graz University of Technology
Signatures (RSA)

\[ M = C^d \mod n \]

Result = Result \times Result

\text{square}
\[ M = C^d \mod n \]

Result = Result × Result × C

\[ \text{square} \]
\[ \text{multiply} \]
Signatures (RSA)

\[ M = C^d \mod n \]

Result = Result \times Result \times C

\[
\begin{array}{ccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\
\end{array}
\]

Daniel Gruss — Graz University of Technology
\[ M = C^d \mod n \]
Prime+Probe Attack

Prime+Probe exploits the timing difference when accessing:
• cached data (fast)
• uncached data (slow)

is used to attack secret-dependent memory accesses
is applied to a part of the CPU cache, a cache set
works across CPU cores as the last-level cache is shared
Prime+Probe Attack

- exploits the timing difference when accessing...

Prime+Probe Osvik2006; Liu2015; Maurice2017Hello..
Prime+Probe Osvik2006; Liu2015; Maurice2017Hello

• exploits the timing difference when accessing...
  • cached data (fast)
Prime+Probe Cache Attack

Prime+Probe Osvik2006; Liu2015; Maurice2017Hello.

- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
Prime+Probe Cache Attack

Prime+Probe *Osvik2006; Liu2015; Maurice2017Hello*.

- exploits the *timing difference* when accessing...
  - cached data (fast)
  - uncached data (slow)
- is used to attack *secret-dependent* memory accesses
Prime+Probe Attack

Prime+Probe Osvik2006; Liu2015; Maurice2017Hello...

- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is used to attack secret-dependent memory accesses
- is applied to a part of the CPU cache, a cache set
Prime+Probe exploits the timing difference when accessing...
- cached data (fast)
- uncached data (slow)

- is used to attack secret-dependent memory accesses
- is applied to a part of the CPU cache, a cache set
- works across CPU cores as the last-level cache is shared
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
**Step 0:** Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
**Step 0:** Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
**Step 0**: Attacker fills the cache (prime)

**Step 1**: Victim evicts cache lines by accessing own data
**Step 0:** Attacker fills the cache (prime)
**Step 1:** Victim evicts cache lines by accessing own data
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed
Victim
Attack Settings

Victim

SGX

Daniel Gruss — Graz University of Technology
Attack Settings

Victim

SGX

Transaction Signature + private key

Wallet API
Attack Settings

Attacker

Victim

SGX

Transaction Signature
+ private key

Wallet API
Attack Settings

Attacker

- SGX

Victim

- SGX
  - Transaction Signature + private key

Wallet API
Attack Settings

Attacker
- SGX
- Key Extractor
- Loader

Victim
- SGX
- Transaction Signature + private key
- Wallet API

Daniel Gruss — Graz University of Technology
Attack Settings

Attacker

SGX

Key Extractor

Loader

Victim

SGX

Transaction Signature
+ private key

Wallet API

Daniel Gruss — Graz University of Technology
Attack Settings

Attacker

SGX

Key Extractor

Loader

L1/L2 Cache

Victim

SGX

Transaction Signature + private key

Wallet API

L1/L2 Cache

Daniel Gruss — Graz University of Technology
### Attack Settings

**Attacker**
- **SGX**
- **Key Extractor**
  - *(Prime+Probe)*
- **Loader**
- **L1/L2 Cache**

**Victim**
- **SGX**
- **Transaction Signature**
  - + private key
- **Wallet API**
- **L1/L2 Cache**

---

**Shared LLC**
Let’s use Docker for Isolation

SGX

Malware

(Prime+Probe)

RSA

(+ private key)

SGX

Loader

API
Let’s use Docker for Isolation

Attacker container

Loader

SGX

Malware

(Prime + Probe)

Victim container

RSA

(+ private key)

Docker engine

Daniel Gruss — Graz University of Technology
Let's use Docker for Isolation

SGX
Malware
(Prime+Probe)
Attacker container
Loader
Docker engine
SGX driver

SGX
RSA (+ private key)
Victim container
API
Classical Prime+Probe cannot be mounted within SGX:

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
- No physical addresses
- No 2 MB large pages
Classical Prime+Probe cannot be mounted within SGX:
- No access to high-precision timer (rdtsc)
Classical Prime+Probe cannot be mounted within SGX:

- No access to high-precision timer (rdtsc)
- No syscalls
Classical Prime+Probe cannot be mounted within SGX:

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
Classical Prime+Probe cannot be mounted within SGX:

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
- No physical addresses
Classical Prime+Probe cannot be mounted within SGX:

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
- No physical addresses
- No 2 MB large pages
• We have to build our own timer
• We have to build our own timer
• Timer resolution must be in the order of cycles
- We have to build our own timer
- Timer resolution must be in the order of cycles
- Start a thread that continuously increments a global variable
• We have to build our own timer
• Timer resolution must be in the order of cycles
• Start a thread that continuously increments a global variable
• The global variable is our timestamp
• We have to build our own timer
• Timer resolution must be in the order of cycles
• Start a thread that continuously increments a global variable
• The global variable is our timestamp
• This is even 15% faster than the native timestamp counter

```
1 mov &timestamp, %rcx
2 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```
• Cache set is determined by part of physical address Maurice2015RAID
• **Cache set** is determined by part of physical address *Maurice2015RAID*

• We have no knowledge of **physical addresses**
• Cache set is determined by part of physical address Maurice2015RAID
• We have no knowledge of physical addresses
• Use the reverse-engineered DRAM mapping Pessl2016
- Cache set is determined by part of physical address\textit{Maurice2015RAID}
- We have no knowledge of physical addresses
- Use the reverse-engineered DRAM mapping\textit{Pessl2016}
- Exploit timing differences to find DRAM row borders
• Cache set is determined by part of physical address Maurice2015RAID
• We have no knowledge of physical addresses
• Use the reverse-engineered DRAM mapping Pessl2016
• Exploit timing differences to find DRAM row borders
• The 18 LSBs are ‘0’ at a row border
How reading from DRAM works

CPU wants to access row 1
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer

row buffer
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
How reading from DRAM works

DRAM bank

CPU wants to access row 2

row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
→ slow (row conflict)
How reading from DRAM works

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
1 1 1 1 1 1 1 1 1 1 1 1 1 1

row buffer

CPU wants to access row 2—again

Daniel Gruss — Graz University of Technology
How reading from DRAM works

DRAM bank

1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
1 1 1 1 1 1 1 1 1 1 1 1 1 1

row buffer

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
→ fast (row hit)
How reading from DRAM works

row buffer = cache
Physical Addresses

8 kB row x in BG0 (1) and channel (1)

8 kB row x in BG0 (0) and channel (1)

8 kB row x in BG0 (1) and channel (0)

8 kB row x in BG0 (0) and channel (0)
Physical Addresses

8 kB row x in BG0 (1) and channel (1)

8 kB row x in BG0 (0) and channel (1)

8 kB row x in BG0 (1) and channel (0)

8 kB row x in BG0 (0) and channel (0)
Physical Addresses

row $n$
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$
Physical Addresses

row $n$

row $n + 1$

row $n + 2$

row $n + 3$

row $n + 4$

row $n + 5$
Physical Addresses
Physical Addresses

row $n$
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$
Physical Addresses

row $n$

row $n + 1$

row $n + 2$

row $n + 3$

row $n + 4$

row $n + 5$
Physical Addresses

![Diagram of physical addresses: row n, row n+1, row n+2, row n+3, row n+4, row n+5.](image)
Physical Addresses

row $n$
row $n + 1$
row $n + 2$
row $n + 3$
row $n + 4$
row $n + 5$
Physical Addresses

row n
row n + 1
row n + 2
row n + 3
row n + 4
row n + 5
Physical Addresses

row $n$
row $n + 1$
row $n + 2$
row $n + 3$
row $n + 4$
row $n + 5$
Physical Addresses

row \( n \)

row \( n + 1 \)

row \( n + 2 \)

row \( n + 3 \)

row \( n + 4 \)

row \( n + 5 \)
Physical Addresses

![Diagram of physical addresses]

- row $n$
- row $n + 1$
- row $n + 2$
- row $n + 3$
- row $n + 4$
- row $n + 5$

Daniel Gruss — Graz University of Technology
Physical Addresses

![Diagram of physical addresses with rows labeled as row n, row n+1, row n+2, row n+3, row n+4, and row n+5. There are waveforms and arrows indicating jumps across rows.](image-url)
Physical Addresses

row $n$

row $n + 1$

row $n + 2$

row $n + 3$

row $n + 4$

row $n + 5$
Physical Addresses
Physical Addresses

Daniel Gruss — Graz University of Technology
Result on an Intel i5-6200U

Array index [kB]

Latency [cycles]
1. Use the **counting primitive** to measure DRAM accesses
1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
1. Use the **counting primitive** to measure DRAM accesses
2. Through the DRAM side channel, determine the **row borders**
3. Row borders have the 18 LSBs set to ‘0’ → maps to **cache set ‘0’**
1. Use the **counting primitive** to measure DRAM accesses
2. Through the DRAM side channel, determine the **row borders**
3. Row borders have the 18 LSBs set to ‘0’ → maps to **cache set ‘0’**
4. Build the **eviction set** for the Prime+Probe attack
1. Use the **counting primitive** to measure DRAM accesses
2. Through the DRAM side channel, determine the **row borders**
3. Row borders have the 18 LSBs set to ‘0’ → maps to **cache set ‘0’**
4. Build the **eviction set** for the Prime+Probe attack
5. Mount **Prime+Probe** on the buffer containing the multiplier **Schwarz2017SGX**
Raw Prime+Probe trace...
...processed with a simple moving average...
...allows to clearly see the bits of the exponent
Performance Counters

- L1 Hits
- L1 Misses
- L3 Hits
- L3 Misses

Performance counter value

Native

- L1 Hits: $10^9$
- L1 Misses
- L3 Hits
- L3 Misses

Daniel Gruss — Graz University of Technology
Performance Counters

Performance counter value

L1 Hits
L1 Misses
L3 Hits
L3 Misses

Native
SGX

\(10^9\)
SGX + Rowhammer: Denial-of-Service as a Service
• cells leak $\rightarrow$ repetitive refresh necessary
• refresh $\approx$ reading (destructive) $+$ writing same data again
• maximum interval between refreshes to guarantee data integrity
• cells leak $\rightarrow$ repetitive refresh necessary
• refresh $\approx$ reading (destructive) + writing same data again
• maximum interval between refreshes to guarantee data integrity

• cells leak faster upon proximate accesses $\rightarrow$ Rowhammer
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
How widespread is the issue?

DDR3:
- Kim et al.: 110/129 modules from 3 vendors, all but 3 since mid-2011
- Seaborn and Dullien: 15/29 laptops

DDR4 believed to be safe:
- but it’s not (Pessl2016)

Figure 1: *

Prevalence, by Kim2014
• ECC eliminates most Rowhammer bit flips
• Intel Core CPUs (i3/i5/i7) generally do not support ECC
SGX Encrypted Memory

Physical Memory

0 GB to 16 GB
SGX EPC (Enclave Page Cache)

- a region in DRAM
- encrypted against physical attackers (cold-boot, bus-sniffing)
- integrity-checked against replay attacks
WHAT HAPPENS WHEN

A BIT FLIPS IN THE EPC?
Integrity check fails? Lock up the memory controller
Integrity check fails? Lock up the memory controller

→ Not a single further memory access!

→ System halts immediately!
Integrity check fails? Lock up the memory controller
→ Not a single further memory access!
→ System halts immediately!

Sounds unsafe?
Integrity check fails? Lock up the memory controller

→ Not a single further memory access!

→ System halts immediately!

Sounds unsafe? It is unsafe!
DoS with SGX and Rowhammer using clflush

Gruss2017Another
DoS with SGX and Rowhammer using \texttt{clflush}

Gruss2017Another
DoS with SGX and Rowhammer using clflush

Gruss2017Another
DoS with SGX and Rowhammer using \textit{clflush}
DoS with SGX and Rowhammer using clflush
DoS with SGX and Rowhammer using clflush
DoS with SGX and Rowhammer using `clflush`

- Cache set 1
- Cache set 2

Gruss2017Another
DoS with SGX and Rowhammer using `clflush`

Gruss2017Another
DoS with SGX and Rowhammer using \texttt{clflush}
DoS with SGX and Rowhammer using clflush

Gruss2017Another
DoS with SGX and Rowhammer using clflush

cache set 1

cache set 2

EPC region

Gruss2017Another
DoS with SGX and Rowhammer using clflush

Gruss2017Another
DoS with SGX and Rowhammer using `clflush`

Gruss2017Another
DoS with SGX and Rowhammer using clflush

System halts unsafely!

Gruss2017Another
Conclusion
• Side channels can attack your SGX-based wallets
• Side channels can attack your SGX-based wallets
• Do not consider side channels out-of-scope – explain why you’re unaffected!
• Side channels can attack your SGX-based wallets
• Do not consider side channels out-of-scope – explain why you’re unaffected!
• Exploitable code $+$ SGX = exploitable SGX enclave
• Side channels can attack your SGX-based wallets
• Do not consider side channels out-of-scope – explain why you’re unaffected!
• Exploitable code + SGX = exploitable SGX enclave
• Rowhammer + SGX = cheap and powerful denial-of-service attacks
Thank you!
Why SGX design flaws hinder its application in cloud computing

Daniel Gruss
06.11.2017

Graz University of Technology