Cash Attacks on SGX

Daniel Gruss, Michael Schwarz

September 9, 2017

Graz University of Technology
Outline

Daniel Gruss, Michael Schwarz — Graz University of Technology
Daniel Gruss, Michael Schwarz — Graz University of Technology
Application

Untrusted part

Create Enclave

Operating System
Application

Untrusted part

Create Enclave

Call Gate

Trusted part

Trusted Fnc.

Operating System

Daniel Gruss, Michael Schwarz — Graz University of Technology
Application

Untrusted part
- Create Enclave
- Call Trusted Fnc.

Trusted part
- Call Gate
- Trusted Fnc.

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Operating System

Daniel Gruss, Michael Schwarz — Graz University of Technology
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Daniel Gruss, Michael Schwarz — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System
- 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)

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

Result = Result \times Result \times C

\begin{array}{cccccc}
1 & 1 & 0 & 0 & 1 & 1 \\
\end{array}

\text{square multiply}
$$M = C^d \mod n$$

Result = Result × Result

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

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

Result = Result \times Result \times C

- **square**
- **multiply**
Signatures (RSA)

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

Result \[\times\] Result \[\times\] C

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

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

Result = Result \times Result

\boxed{1 1 0 0 1 1 0 \ldots}

square
• Used to sign transactions
- Used to sign transactions
- Point multiplication is similar to RSA exponentiation
• Used to sign transactions
• Point multiplication is similar to RSA exponentiation
• Simplest implementation **double-and-add** or constant-time
  *Montgomery ladder*
ECDSA

- Used to sign transactions
- Point multiplication is similar to RSA exponentiation
- Simplest implementation double-and-add or constant-time Montgomery ladder
- Both algorithms have secret-dependent memory accesses
Prime+Probe [OST06; Liu+15; Mau+17]...
Prime+Probe [OST06; Liu+15; Mau+17]...

- exploits the **timing difference** when accessing...
Prime+Probe [OST06; Liu+15; Mau+17]...

- exploits the timing difference when accessing...
  - cached data (fast)
Prime+Probe [OST06; Liu+15; Mau+17]...

- exploits the **timing difference** when accessing...
  - cached data (fast)
  - uncached data (slow)
Prime+Probe [OST06; Liu+15; Mau+17]...

- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is used to attack secret-dependent memory accesses
Prime+Probe [OST06; Liu+15; Mau+17]...

- 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 [OST06; Liu+15; Mau+17]...

- 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
Attack
Attack Settings

Victim
Attack Settings

Victim

SGX
Victim

SGX

Transaction Signature
+ private key

Wallet API
Attack Settings

Attacker

Victim

SGX

Transaction Signature
+ private key

Wallet API

Daniel Gruss, Michael Schwarz — Graz University of Technology
Attack Settings

Attacker

Victim

SGX

Transaction Signature + private key

Wallet API

Daniel Gruss, Michael Schwarz — Graz University of Technology
Attack Settings

Attacker

SGX

Key Extractor

Loader

Victim

SGX

Transaction Signature + private key

Wallet API
Attack Settings

Attacker

SGX

Key Extractor

Loader

Victim

SGX

Transaction Signature + private key

Wallet API
Attacker Settings

Attacker

SGX

Key Extractor

Loader

L1/L2 Cache

Victim

SGX

Transaction Signature
+ private key

Wallet API

L1/L2 Cache
**Attack Settings**

Attacker

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

Victim

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

---

Shared LLC
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 (\texttt{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 [Mau+15]
- **Cache set** is determined by part of physical address [Mau+15]
- We have no knowledge of physical addresses
Physical Addresses

- Cache set is determined by part of physical address [Mau+15]
- We have no knowledge of physical addresses
- Use the reverse-engineered DRAM mapping [Pes+16]
• Cache set is determined by part of physical address [Mau+15]
• We have no knowledge of physical addresses
• Use the reverse-engineered DRAM mapping [Pes+16]
• Exploit timing differences to find DRAM row borders
• **Cache set** is determined by part of physical address [Mau+15]
• We have no knowledge of **physical addresses**
• Use the reverse-engineered **DRAM mapping** [Pes+16]
• Exploit timing differences to find **DRAM row borders**
• The 18 LSBs are ‘0’ at a row border
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)

Page #2  Page #3  Page #4  Page #5  Page #6  Page #7  Page #8

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

Page #2  Page #3  Page #4  Page #5  Page #6  Page #7  Page #8

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

Page #1  Page #2  Page #3  Page #4  Page #5  Page #6  Page #7  Page #8

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

Page #1  Page #2  Page #3  Page #4  Page #5  Page #6  Page #7  Page #8

Daniel Gruss, Michael Schwarz — Graz University of Technology
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

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

row $n$

row $n + 1$

row $n + 2$

row $n + 3$

row $n + 4$

row $n + 5$
Physical Addresses
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

row $n$
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$
Physical Addresses
Result on an Intel i5-6200U
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 [Sch+17]
Results
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
<table>
<thead>
<tr>
<th>Performance Counter</th>
<th>Native</th>
<th>SGX</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Hits</td>
<td>10^9</td>
<td>1</td>
</tr>
<tr>
<td>L1 Misses</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L3 Hits</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>L3 Misses</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Countermeasures
• Cache attacks can be prevented on source level
Source Level

- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
Source Level

- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
- Exponent blinding for RSA prevents multi-trace attacks
- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
- Exponent blinding for RSA prevents multi-trace attacks
- Bit-sliced implementations are not vulnerable to cache attacks
• Trusting the operating system weakens SGX threat model
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Enclave coloring to prevent cross-enclave attacks
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Enclave coloring to prevent cross-enclave attacks
• Heap randomization to randomize cache sets
- Intel could prevent attacks by changing the hardware

Combine Cache Allocation Technology (CAT) with SGX
- Instead of controlling CAT from the OS, combine it with enenter
- Entering an enclave would automatically activate CAT for this core
- L3 is then isolated from all other enclaves and applications
- Provide a non-shared secure memory element which is not cached
• Intel could prevent attacks by changing the hardware
• Combine Cache Allocation Technology (CAT) with SGX
- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with eenter
- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with eenter
  - Entering an enclave would automatically activate CAT for this core
Intel could prevent attacks by changing the hardware

Combine Cache Allocation Technology (CAT) with SGX
- Instead of controlling CAT from the OS, combine it with eenter
- Entering an enclave would automatically activate CAT for this core
- L3 is then isolated from all other enclaves and applications
- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with eenter
  - Entering an enclave would automatically activate CAT for this core
  - L3 is then isolated from all other enclaves and applications
- Provide a non-shared secure memory element which is not cached
Conclusion
• Side channels can cost you money
• Do not consider side channels out-of-scope
• Exploitable code + SGX = exploitable SGX enclave
Thank you!
Cash Attacks on SGX

Daniel Gruss, Michael Schwarz
September 9, 2017

Graz University of Technology


Error probability depends on which cache set of the key we attack.

![Graph showing bit-error ratio for different cache sets with 4096-bit key.]
Error probability depends on which cache set of the key we attack.
Full recovery of a 4096-bit RSA key in approximately 5 minutes
CPU cycles one increment takes

```
rdtsc 1
```

```
timestamp = rdtsc();
```
CPU cycles one increment takes

```c
while(1) {
    timestamp++;
}
```
CPU cycles one increment takes

C:

```c
rdtsc
```

Assembly:

```assembly
1 mov &timestamp, %rcx
2 incl (%rcx)
3 jmp 1b
```
CPU cycles one increment takes

**C**

```
rdtsc 1
```

4.7

**Assembly**

```
mov &timestamp, %rcx
1: inc %rax
mov %rax, (%rcx)
jmp 1b
```

4.67

**Optimized**

0.87
Bonus: Docker

SGX

Malware

(Prime+Probe)

SGX

RSA

(+ private key)

Loader

API
Bonus: Docker

SGX
Malware
(Prime+Probe)

RSA
(+ private key)

Docker engine

Daniel Gruss, Michael Schwarz — Graz University of Technology
Malware \((Prime+Probe)\)