Cash Attacks on SGX

Daniel Gruss, Michael Schwarz
September 9, 2017

Graz University of Technology
Application

Untrusted part

Operating System
Application

Untrusted part

Create Enclave

Trusted part

Call Gate

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
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

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

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
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

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

Daniel Gruss, Michael Schwarz — Graz University of Technology
- 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. [...]

Daniel Gruss, Michael Schwarz — Graz University of Technology
\[ M = C^d \mod n \]
M = C^d \mod n
Signatures (RSA)

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

Result = Result \times Result \times C

square multiply
Signatures (RSA)

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

Result = Result \times Result

square
Signatures (RSA)

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

Result = Result \times Result

\text{square}
Signatures (RSA)

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

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

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

\underline{square} \hspace{2cm} \underline{multiply}
M = C^d \mod n

\begin{array}{c}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \ldots
\end{array}

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

\underline{\text{square}} \quad \underline{\text{multiply}}
Signatures (RSA)

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

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

square
- Used to sign transactions
ECDSA

- Used to sign transactions
- Point multiplication is similar to RSA exponentiation
ECDSA

- 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+Probes [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

SGX
Attack Settings

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
SGX

Victim
SGX
Transaction Signature
+ private key

Wallet API
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

9

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

SGX Key Extractor (Prime+Probe)

Loader

SGX Transaction Signature + private key

Wallet API

L1/L2 Cache

Shared LLC
Classical Prime+Probe cannot be mounted within SGX:
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 ($\text{rdtsc}$)
- No syscalls
- No shared memory
SGX Limitations

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 [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

0
BG0 (0), Channel (0)
BG0 (1), Channel (0)

127
BG0 (0), Channel (0)
BG0 (1), Channel (0)

4095
BG0 (0), Channel (0)
BG0 (1), Channel (0)

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

4 kB Page #1 4095

Daniel Gruss, Michael Schwarz — Graz University of Technology
Physical Addresses

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

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

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

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

4 kB Page #1

0 127
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
Physical Addresses

![Physical Addresses Diagram]
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
Physical Addresses
Result on an Intel i5-6200U

![Latency vs Array Index Graph]

- **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’ $\rightarrow$ 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: $10^9$
- L1 Misses: 1
- L3 Hits: 0.5
- L3 Misses: 0

Native

Daniel Gruss, Michael Schwarz — Graz University of Technology
Performance Counters

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

Performance counter value

- Native
- SGX

Daniel Gruss, Michael Schwarz — Graz University of Technology
Countermeasures
Source Level

- Cache attacks can be prevented on 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
- 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
• Side channels can cost you money
• Do not consider side channels out-of-scope
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 4096-bit key]
Error probability depends on which cache set of the key we attack.

- **Bit-error ratio [%]**
  - 4096-bit key
  - Traces:
    - Bit-errors: 69, 15, 4, 1, 0

---

Daniel Gruss, Michael Schwarz — Graz University of Technology
Full recovery of a 4096-bit RSA key in approximately 5 minutes
CPU cycles one increment takes

\[ \text{timestamp} = \text{rdtsc}(); \]
CPU cycles one increment takes

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

\begin{align*}
\text{rdtsc} & \quad 1 \\
C & \quad 4.7 \\
\text{Assembly} & \quad 4.67
\end{align*}

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

rdtsc  1

C          4.7

Assembly  4.67

Optimized  0.87

1 mov &timestamp, %rcx
2 inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
Bonus: Docker

Malware (Prime+Probe)

RSA (+ private key)
Bonus: Docker

Attacker container

Loader

Malware

(Prime+Probe)

SGX

Victim container

RSA

(+ private key)

SGX

API

Docker engine

Daniel Gruss, Michael Schwarz — Graz University of Technology
Bonus: Docker

SGX
Malware
*(Prime+Probe)*

RSA
(+ private key)

Attacker container
Loader

Victim container
API

Docker engine

SGX driver

Daniel Gruss, Michael Schwarz — Graz University of Technology