SESSION ID: CRYP-T07

BREAKING ED25519 IN WOLFSSL

Niels Samwel¹, Lejla Batina¹, Guido Bertoni², Joan Daemen¹,² and Ruggero Susella²

Radboud University¹
STMicroelectronics²
Introduction

- ECDSA is a signature scheme that uses random ephemeral scalars
- Weak keys can be attacked with lattice attacks
- Ed25519 is a deterministic signature scheme that prevents weak keys
- When an adversary has access to a device this is becomes a weakness
Contributions

- Side-channel attack on Ed25519.
- Side-channel attack on the message schedule of SHA512.
- Countermeasure for this attack.
The Attack Components

Three components

- Attacking Ed25519 to recover long term secret
- Attack on SHA512
- DPA on modular addition
Algorithm 1 Ed25519 key setup and signature generation

Key setup.
1: Hash \( k \) such that \( H(k) = (h_0, h_1, \ldots, h_{2b-1}) = (a, b) \)
2: \( a = (h_0, \ldots, h_{b-1}) \), Private scalar
3: \( b = (h_b, \ldots, h_{2b-1}) \), Auxiliary key
4: Compute public key: \( A = aB \).

Signature generation.
5: Compute ephemeral private key: \( r = H(b, M) \).
6: Compute ephemeral public key: \( R = rB \).
7: Compute \( h = H(R, A, M) \) and convert to integer.
8: Compute: \( S = (r + ha) \mod l \).
9: Signature pair: \( (R, S) \).
Using auxiliary key \( b \) that was successfully recovered:

1. Compute \( r = H(b, M) \).
2. Compute \( h = H(R, A, M) \).
3. Compute \( a = (S - r)h^{-1} \bmod l \).
Algorithm 2 Merkle Damgård

**Input:** Message $M$ with $0 \leq \text{bit-length} < 2^{128}$

**Output:** Hash value of $M$

1. Pad message $M$ by appending an encoding of the message length
2. Initialize chaining value $CV$ with constant $IV$
3. Split padded message $M$ into blocks
4. for all blocks $M_i$ do
5. $CV_{i+1} \leftarrow CF(CV_i, M_i)$
6. end for
7. return $H \leftarrow CV$
Attack on SHA512
Attack on SHA512
DPA on modular addition

\[ w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0] \]
\[ w[17] \leftarrow \sigma_1(w[15]) + w[10] + \sigma_0(w[2]) + w[1] \]
\[ w[18] \leftarrow \sigma_1(w[16]) + w[11] + \sigma_0(w[3]) + w[2] \]
\[ w[19] \leftarrow \sigma_1(w[17]) + w[12] + \sigma_0(w[4]) + w[3] \]
DPA on modular addition

Setup
DPA on modular addition

Discrete power consumption values

Round 16

Time samples

Discrete power consumption values

Round 16

Time samples
DPA on modular addition

\[ w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0] \]
DPA on modular addition

![Graph showing success probability vs. number of traces for different values of $k$.]
Countermeasure
Conclusion

Side-channel attack on Ed25519

- High success rate
- Inexpensive countermeasure
DPA on modular addition

![Correlation value graph showing key candidates and symmetric counterparts.](image)
DPA on modular addition

-0.1
0
0.1

Correlation values

Correct key candidate

 Incorrect key candidate

0 2 4 6 8 10 12
Time samples × 10^4

0 2 4 6 8 10 12
Time samples × 10^4
DPA on modular addition
MEMJAM: A FALSE DEPENDENCY ATTACK AGAINST CONSTANT-TIME CRYPTO IMPLEMENTATIONS IN SGX

Daniel Moghimi
Ph.D. Student
Worcester Polytechnic Institute
@danielmgmi
MemJam: A False Dependency Attack against Constant-Time Crypto Implementations in SGX

Ahmad “Daniel” Moghimi
Thomas Eisenbarth
Berk Sunar

April 17, 2018
CT-RSA 2018 - San Francisco, CA
Data Dependency

• Data dependency: Instruction $\rightarrow$ Data of a preceding instruction

```
add %ebx, %eax
sub %eax, %edx
xor %ecx, %ecx
add %eax, %edi
sub %ecx, %edi
```
Data Dependency – Pipelined Execution

• Data dependency: Instruction → Data of a preceding instruction

```
add %ebx, %eax
sub %eax, %edx
xor %ecx, %ecx
add %eax, %edi
sub %ecx, %edi
```

IF  Instruction Fetch
ID  Instruction Decode
EX  Execute
WB  Write Back
Data Dependency – Pipelined Execution

- Data dependency: Instruction → Data of a preceding instruction

```
add %ebx, %eax
sub %eax, %edx
xor %ecx, %ecx
add %eax, %edi
sub %ecx, %edi
```

IF: Instruction Fetch
ID: Instruction Decode
EX: Execute
WB: Write Back
Data Dependency – Pipelined Execution

• Data dependency: Instruction $\rightarrow$ Data of a preceding instruction
Data Dependency – Pipelined Execution

• Data dependency: Instruction → Data of a preceding instruction
Data Dependency – Pipelined Execution

• Data dependency: Instruction → Data of a preceding instruction

Instruction Fetch
Instruction Decode
Execute
Write Back
Data Dependency – Pipelined Execution

- Data dependency: Instruction $\rightarrow$ Data of a preceding instruction
Data False Dependency

• Pipeline stalls without true dependency.
• Reasons:
  • Register Reuse
  • Limited Address Space

\[
\begin{align*}
\text{add} & \ %ebx, %eax \\
\text{xor} & \ %ecx, %ecx \\
\text{sub} & \ %eax, %edx \\
\text{mov} & \ $100, %edx \\
\text{sub} & \ %edx, %ecx
\end{align*}
\]
False Dependency – Register Renaming

- Pipeline stalls without true dependency.
- Reasons:
  - Register Reuse
  - Limited Address Space

```
add %ebx, %eax  1
xor %ecx, %ecx  2
sub %eax, %edx  3
mov $100, %edx  4
sub %edx, %ecx  5
```

```
add %ebx, %eax  1
xor %ecx, %ecx  2
sub %eax, %edx  3
mov $100, %bat  4
sub %bat, %ecx  5
```
Memory False Dependency – 4K Aliasing

- Memory loads/stores are executed out of order and speculatively.
- The dependency is verified after the execution!

- 4K Aliasing: Addresses that are 4K apart are assumed dependent.
- Re-execute the load and corresponding instructions due to false dependency.
- Virtual-to-physical address translation → Memory disambiguation
# 4K Aliasing – Hyperthreading

<table>
<thead>
<tr>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>HT – Thread A</td>
</tr>
<tr>
<td>HT – Thread B</td>
</tr>
</tbody>
</table>
4K Aliasing – Hyperthreading
4K Aliasing – Hyperthreading

Core

HT – Thread A
- Load 0xFECD1 234
- Load 0xFECD2 234
- Load 0xFECD3 234
- Load 0xFECD4 234
- Load 0xFECD5 234
- Load 0xFECD6 234
- Load 0xFECD7 234
- Load 0xFECD8 234

HT – Thread B

Execute & Time
4K Aliasing – Hyperthreading

Core

HT – Thread A
- Load 0xFECD1
- Load 0xFECD2
- Load 0xFECD3
- Load 0xFECD4
- Load 0xFECD5
- Load 0xFECD6
- Load 0xFECD7
- Load 0xFECD8

HT – Thread B
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF
- Store 0x12ABCDEF

Execute & Time

Cycles

100
90
80
70
60
50
40
30
0
100,000
80,000
60,000
40,000
20,000
4K Aliasing – Hyperthreading

Core

HT – Thread A

Load 0xFECD1234
Load 0xFECD2234
Load 0xFECD3234
Load 0xFECD4234
Load 0xFECD5234
Load 0xFECD6234
Load 0xFECD7234
Load 0xFECD8234

HT – Thread B

Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
Store 0x12ABC200
4K Aliasing – Hyperthreading

Core

HT – Thread A
- Load 0xFECD1234
- Load 0xFECD2234
- Load 0xFECD3234
- Load 0xFECD4234
- Load 0xFECD5234
- Load 0xFECD6234
- Load 0xFECD7234
- Load 0xFECD8234

HT – Thread B
- Store 0x12ABC234
- Store 0x12ABC234
- Store 0x12ABC234
- Store 0x12ABC234
- Store 0x12ABC234
- Store 0x12ABC234
- Store 0x12ABC234

Execute & Time

Graph showing cycles vs. number of events.
MemJam – Intra Cache Line Resolution

Rest of the bits (Virtual != Physical)

Least 12 bits (Virtual Address = Physical Address)
MemJam – Intra Cache Line Resolution

Least 12 bits (Virtual Address = Physical Address)

Rest of the bits (Virtual ! = Physical)

L1 Cache Attacks
MemJam – Intra Cache Line Resolution

Rest of the bits (Virtual != Physical)

Least 12 bits (Virtual Address = Physical Address)

L1 Cache Attacks

L2/LLC Cache Attacks
MemJam – Intra Cache Line Resolution

Rest of the bits (Virtual != Physical)  Least 12 bits (Virtual Address = Physical Address)

L1 Cache Attacks

L2/LLC Cache Attacks

2015 – Irazoqui
2014 – Yarom – Flush+Reload

2005 – Percival – Cache Missing for Fun

2006 – Osvik – Cache attacks
MemJam – Intra Cache Line Resolution

- Intra-cache line Leakage (4-byte granularity)
- Higher time correlates → Memory accesses with the same bit 3 to 12
- 4 bits of intra-cache level leakage
MemJam Attack
MemJam Attack

```assembly
mov [%target], %rax;
write_loop:
.rept 100;
movb $0, (%rax);
.endr;
jmp write_loop;
```
MemJam Attack

mov [%target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
  jmp write_loop;

Request Encryption
MemJam Attack

```assembly
mov [%target], %rax;
write_loop:
    .rept 100;
    movb $0, (%rax);
    .endr;
    jmp write_loop;
```
MemJam Attack

mov [%target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
  jmp write_loop;
MemJam Attack

```assembly
mov [%target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
  jmp write_loop;
```
MemJam Attack

```
mov %[target], %rax;
write_loop:
    .rept 100;
movb $0, (%rax);
.endr;
jmp write_loop;
```
MemJam Attack

```
mov %[target], %rax;
write_loop:
.rept 100;
movb $0, (%rax);
.endr;
jmp write_loop;
```

Higher time if there are more number of 4K conflicts
Constant time AES – Safe2Encrypt_RIJ128

• Scatter-gather implementation of AES
  • 256 S-Box – 4 Cache Line
  • Cache independent access pattern

• Implemented and distributed as part of Intel products
  • Intel SGX Linux Software Development Kit (SDK)
  • Intel IPP Cryptography Library
Constant time AES – Safe2Encrypt_RIJ128

• Scatter-gather implementation of AES
  • 256 S-Box – 4 Cache Line
  • Cache independent access pattern

• Implemented and distributed as part of Intel products
  • Intel SGX Linux Software Development Kit (SDK)
  • Intel IPP Cryptography Library
Constant time AES – Safe2Encrypt_RIJ128

• Scatter-gather implementation of AES
  • 256 S-Box – 4 Cache Line
  • Cache independent access pattern

• Implemented and distributed as part of Intel products
  • Intel SGX Linux Software Development Kit (SDK)
  • Intel IPP Cryptography Library
MemJam Attack on AES
MemJam Attack on AES

- 64 Bytes
- 4 Cache Lines
MemJam Attack on AES

64 Bytes

4 Cache Lines
MemJam Attack on AES

\[ index = S^{-1}(c \oplus k) \]
MemJam Attack on AES

\[ \text{index} = S^{-1}(c \oplus k) \rightarrow \text{index} < 4. \]
AES Key Recovery
AES Key Recovery
SM4 Block cipher – cpSMS4_Cipher

- Standard Cipher support by Intel
  - Chinese National Standard for Wireless LAN WAPI
- S-Box + Unbalanced Feistel Structure
- Protected by Cache State Normalization

- Recursive attack → Full key recovery with 40K observations
MemJaming Intel SGX Secure Enclave

mov [%target], %rax;
write_loop:
  .rept 100;
  movb $0, (%rax);
  .endr;
  jmp write_loop;

Higher time if there are more number of 4K conflicts
## Intel SGX – AES Key Recovery

<table>
<thead>
<tr>
<th>Key Bytes (Ranks)</th>
<th>5 M</th>
<th>10 M</th>
<th>15 M</th>
<th>20 M</th>
<th>25 M</th>
<th>30 M</th>
<th>35 M</th>
<th>40 M</th>
<th>45 M</th>
<th>46.8 M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>83</td>
<td>79</td>
<td>25</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>85</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>99</td>
<td>59</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>4</td>
<td>11</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>92</td>
<td>98</td>
<td>37</td>
<td>13</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>99</td>
<td>71</td>
<td>35</td>
<td>29</td>
<td>40</td>
<td>42</td>
<td>42</td>
<td>25</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>81</td>
<td>39</td>
<td>99</td>
<td>95</td>
<td>84</td>
<td>76</td>
<td>46</td>
<td>45</td>
<td>43</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>99</td>
<td>99</td>
<td>51</td>
<td>66</td>
<td>47</td>
<td>43</td>
<td>40</td>
<td>50</td>
<td>81</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>99</td>
<td>85</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion

• New Side-Channel Attack Applicable to all Intel Processors
  • Intel SGX extensions

• Bypass of Constant-Time Implementations Techniques
  • Scatter-Gather
  • Cache State Normalization

• Agnostic to other Cache Attack Defense Mechanism

• Intel Trilogy
  • Intel Hardware
  • Intel Trusted Execution Environment
  • Intel Hardened Crypto Implementation
## Responsible Disclosure

<table>
<thead>
<tr>
<th>Date</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/02/2017</td>
<td>Reported</td>
</tr>
<tr>
<td>08/04/2017</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>11/07/2017</td>
<td>Safe2Encrypt_RIJ128 got removed from SGX SDK.</td>
</tr>
<tr>
<td>11/17/2017</td>
<td>CVE-2017-5737 Assigned</td>
</tr>
<tr>
<td>work-in-progress</td>
<td>Patch</td>
</tr>
</tbody>
</table>
Questions?! 

Vernam Group
v.wpi.edu

@VernamGroup
@danielmgmi
<table>
<thead>
<tr>
<th>Implementation Technique</th>
<th>Function Name</th>
<th>l9 n0 y8 k0 e9</th>
<th>m7 mx</th>
<th>n8</th>
<th>Linux SGX SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-NI</td>
<td>Encrypt_RIJ128_AES_NI</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓ (pre-built)</td>
</tr>
<tr>
<td>AES Bitsliced</td>
<td>SafeEncrypt_RIJ128</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓ (pre-built)</td>
</tr>
<tr>
<td>AES Constant-Time</td>
<td>Safe2Encrypt_RIJ128</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓ (source)</td>
</tr>
<tr>
<td>SM4 Bitsliced using AES-NI</td>
<td>cpSMS4_ECB_aesni</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>N/A</td>
</tr>
<tr>
<td>SM4 Cache Normalization</td>
<td>cpSMS4_Cipher</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>N/A</td>
</tr>
<tr>
<td>Release</td>
<td>Family</td>
<td>Cache Bank Conflicts</td>
<td>4K Aliasing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
<td>----------------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>Core</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Nehalem</td>
<td>×</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Sandy bridge</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>Silvermont, Haswell, Broadwell</td>
<td>×</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>Skylake</td>
<td>×</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>KabyLake</td>
<td>×</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>