CIPHERLEAKS

Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel

Mengyuan Li\textsuperscript{1}, Yinqian Zhang\textsuperscript{2}, Huibo Wang\textsuperscript{3}, Kang Li\textsuperscript{3} and Yueqiang Cheng\textsuperscript{4}

\textsuperscript{1}The Ohio State University, \textsuperscript{2}Southern University of Science and Technology, \textsuperscript{3}Baidu Security, \textsuperscript{4}NIO Security Research
AMD SECURE ENCRYPTED VIRTUALIZATION (SEV)

SaaS
PaaS
IaaS

Infrastructure as a service

Cloud Computing

+ Hardware Memory Encryption

AMD Secure Encrypted Virtualization (SEV)

AMD | AWS | Microsoft Azure | Google Cloud
“SEV technology is built around a threat model where an attacker is assumed to have access to not only execute user level privileged code on the target machine, but can potentially execute malware at the higher privileged hypervisor level as well.”
HARDWARE MEMORY ENCRYPTION

- AMD Secure Processor (SP) & 128-bit AES engine.

Diagram shows:
- CPU
- VM
- AES-128 Engine
- DRAM

AM
D
SP

AMD Secure Processor (SP)
HARDWARE MEMORY ENCRYPTION

• Data are unencrypted in CPU.
HARDWARE MEMORY ENCRYPTION

- Data are encrypted in the memory.
• VM’s Memory is accessed randomly.  

Memory is independently encrypted per 128-bit block.

128-bit block

<table>
<thead>
<tr>
<th>0x0</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
128-BIT AES ENCRYPTION WITH XEX MODE

• VM’s Memory is accessed randomly.  

• To avoid inferring plaintext (m) via the same ciphertext (C).

Memory is independently encrypted per 128-bit block.

XEX mode with a tweak function $T(x)$.

$$C = T(sPA_m) \oplus Enc(m \oplus T(sPA_m))$$
SEV ITERATION

SEV — APR 2016

SEV-ES (Encrypted State) — FEB 2017

SEV-SNP (Secure Nested Paging) — JAN 2020
SEV ENCRYPTED STATE (SEV-ES)

Protecting Register State in VM Control Block (VMCB)

World Switch in SEV

Guest VM’s world

VMEXIT

Save states into VMCB

Register State

VMCB

Launch states from VMCB

VMRUN

Host’s world

Host handles VMEXIT
SEV ENCRYPTED STATE (SEV-ES)

Storing Register State into VM Save Area (VMSA)

World Switch in SEV-ES

Guest VM’s world

VMEXIT

Save Register states into VMSA

VMSA

VMCB

Host’s world

Host handles VMEXIT

Launch states from VMSA

VMRUN

VMSA

VMCB
SEV SECURE NESTED PAGING (SEV-SNP)

Protecting Nested Page Table (nPT) & VM’s memory via Reverse Map Table (RMP)

- nPT remapping[1]
- Ciphertext modification[2,3]

SEV SECURE NESTED PAGING (SEV-SNP)

Protecting Nested Page Table (nPT) & VM’s memory via Reverse Map Table (RMP)

- Prevent nPT modification
- Restrict host’s write access to VM’s memory
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
CIPHERTEXT SIDE CHANNEL

Target at ciphertext inside the VMSA

Guest VM’s world

VMEXIT

Save Register states into VMSA

- Encrypted by VM memory encryption key

VMSA

VMCB

Host’s world
CIPHERTEXT SIDE CHANNEL

Target at ciphertext inside the VMSA

Save Register states into VMSA

<table>
<thead>
<tr>
<th>Register</th>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
<td></td>
</tr>
<tr>
<td>RFLAGS</td>
<td>RIP</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>RSP</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>RAX</td>
<td></td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
<td></td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
<td></td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
<td></td>
</tr>
</tbody>
</table>

0x1f0

16-byte block
CIPHERTEXT SIDE CHANNEL

Target at ciphertext inside the VMSA

Save Register states into VMSA

8-byte

<table>
<thead>
<tr>
<th>CR3</th>
<th>CR2</th>
<th>CR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFLAGs</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RDX</td>
<td>RCX</td>
<td>RAX</td>
</tr>
<tr>
<td>RBP</td>
<td>RSP</td>
<td>RIP</td>
</tr>
<tr>
<td>RDI</td>
<td>RSI</td>
<td>R9</td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
<td>R9</td>
</tr>
</tbody>
</table>

16-byte block

Adversary

Write access

Read access

0x1f0
CIPHERTEXT SIDE CHANNEL

Force frequent VMEXIT and record VMSA

What can I get?

VMEXIT

VMSA

VMCB

B B B B B B B B B B B

VMRUN

Guest VM’s world

App execution

Host’s world

Force more VMEXITs by unsetting Present bit in the nPT

Adversary
• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
EXECUTION STATE INFEERENCE

Learn execution states by monitoring ciphertext changes

---

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
</tr>
<tr>
<td>RFLAGS</td>
<td>RIP</td>
</tr>
<tr>
<td>-</td>
<td>RSP</td>
</tr>
<tr>
<td>-</td>
<td>RAX</td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
</tr>
</tbody>
</table>

0x1f0

16-byte block

---

Adversary

What can I get?
EXECUTION STATE INFERENCE

Learn execution states by monitoring ciphertext changes

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
</tr>
<tr>
<td>RFLAGS</td>
<td>RIP</td>
</tr>
<tr>
<td>-</td>
<td>RSP</td>
</tr>
<tr>
<td>-</td>
<td>RAX</td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
</tr>
</tbody>
</table>

0x1f0

16-byte block

Indicate of context-switch. Process’s ID.

What can I get?

Adversary
EXECUTION STATE INFERENCE

Learn execution states by monitoring ciphertext changes

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
<th>Indicate of context-switch. Process’s ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
<td></td>
</tr>
<tr>
<td>RFLAGS</td>
<td>RIP</td>
<td>Rip advance.</td>
</tr>
<tr>
<td></td>
<td>RSP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAX</td>
<td>RAX change.</td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
<td></td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
<td></td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
<td></td>
</tr>
</tbody>
</table>

0x1f0

16-byte block

What can I get?

Adversary
EXECUTION STATE INFERENGE

Learn execution states by monitoring ciphertext changes

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
</tr>
<tr>
<td>RFLAGs</td>
<td>RIP</td>
</tr>
<tr>
<td>-</td>
<td>RSP</td>
</tr>
<tr>
<td>-</td>
<td>RAX</td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
</tr>
</tbody>
</table>

0x1f0

Indicate of context-switch. Process’s ID.

Rip advance.

RAX change.

NPF info, controlled channel

Identify execution state for a known binary

Adversary
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
Ciphertext side channel becomes more powerful when we can build Ciphertext-plaintext dictionary (C-P dictionary).

- One way to collect C-P dictionary is via Non-Automatic VM Exits (NAE).
Ciphertext side channel becomes more powerful when we can build Ciphertext-plaintext dictionary.
SELECTIVE PLAINTEXT RECOVERY

Ciphertext side channel becomes more powerful when we can build Ciphertext-plaintext dictionary.
Ciphertext side channel becomes more powerful when we can build Ciphertext-plaintext dictionary.

- One way to collect C-P dictionary is via Non-Automatic VM Exits (NAE).

Can collect all C-P pairs when plaintext of RAX range from $[0, 127]$ by NAE (CPUID, IOIO_Port, ...) during boot period.

......
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

Code segment in BN_mod_exp_mont_consttime

```
/*
 * Scan the exponent one window at a time starting
 * from the most significant bits.
 */

while (bits > 0) {
    bn_power5(tmp.d, tmp.d, powerbuf, np, n0, top,
    bn_get_bits5 p->d, bits -= 5));
}
```

.globl bn_get_bits5
......
cmoa %r11,%r10
cmoa %eax,%ecx
movzw (%r10,%num,2),%eax
shrl %cl,%eax
and $31,%eax
ret
......
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

/*
 * Scan the exponent one window at a time starting from the most significant bits.
 */
while (bits > 0) {
    bn_power5(tmp.d, tmp.d, powerbuf, np, n0, top,
             bn_get_bits5, p->d, bits -= 5));
}

.globl bn_get_bits5
......
cmova %r11,%r10
cmova %eax,%ecx
movzw (%r10,$num,2),%eax
shrl %cl,%eax
and \$31,%eax
ret
......
**END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY**

---

**Step1:** Use execution state inference method to locate the gPA of Func1 & Func2.

```c
/*
 * Scan the exponent one window at a time starting from the most significant bits.
 */

while (bits > 0) {
    bn_power5(tmp.d, tmp.d, powerbuf, np, n0, top,
    bn_get_bits5(p->d, bits -= 5));
}
```
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

**Step 2:** Iteratively unset the present bit of Func1 & Func2. (e.g., unset Func2’s present bit when intercept Func1’s NPF)
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

Step2: Iteratively unset the present bit of Func1 & Func2. (e.g., unset Func2’s present bit when intercept Func1’s NPF)

410 iterations for 2048-bit private key. (2048-bit/5-bit)

Adversary
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

Step3: Get RAX’s ciphertext when intercept Func2’s NPF. Each time the ciphertext represents 5-bit of the private key.

410 iterations for 2048-bit private key

NPF of gPA2 (F2)
NPF of gPA1 (F1)
NPF of gPA2 (F2)
...
NPF of gPA1 (F1)
NPF of gPA2 (F2)
END-TO-END ATTACK #1: STEAL RSA PRIVATE KEY

Step 4: compare with pre-collected ciphertext-plaintext pair
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  ● Execution State Inference
  ● Selective Plaintext Recovery

• End-to-end Attacks
  ● RSA
  ● ECDSA

• Discussion

• Conclusion
END-TO-END ATTACK #2: STEAL ECDSA NONCE K

```c
for (i = cardinality_bits - 1; i >= 0; i--) {
    kbit = BN_is_bit_set(k, i) ^ pbit;
    EC_POINT_CSWAP(kbit, r, s, group_top, Z_is_one);
    // Perform a single step of the Montgomery ladder
    if (!ec_point_ladder_step(group, r, s, p, ctx)) {
        ERR_raise(ERR_LIB_EC,
                  EC_R_LADDERS_STEP_FAILURE);
        goto err;
    }
    // pbit logic merges this cswap with that of the
    // next iteration
    pbit ^= kbit;
}
```

The code segment in `ec_sclar Mul_ladder()`

```
000f8e20 <BN_is_bit_set>:              .......
      f8e38: 48 8b 04 d0  mov (%rax,%rdx,8),%rax
      f8e3c: 48 d3 e8  shr %cl,%rax
      f8e3f: 83 e0 01 and $0x1,%eax
      f8e42: f3 c3    repz retq
```

Adversary
END-TO-END ATTACK #2: STEAL ECDSA NONCE K

**Step1**: locate the gPA of Func1 & Func2.

**Step2**: Iteratively unset the present bit.

**Step3**: Get RAX’s ciphertext. Each time the ciphertext represents 1-bit of the nonce k.

---

```c
for (i = cardinality_bits - 1; i >= 0; i--) {
    kbit = BN_is_bit_set(k, i) ^ pbit;
    EC_POINT1_CSWAP(kbit, r, s, group_top, Z_is_one);
    // Perform a single step of the Montgomery ladder
    if (!ec_point_ladder_step(group, r, s, p, ctx)) {
        ERR_raise(ERR_LIB_EC,
                  EC_R_LADDER_STEP_FAILURE);
        goto err;
    }
    // pbit logic merges this cswap with that of the next iteration
    pbit ^= kbit;
}
```

Code segment in `ec_scalar_mul_ladder()`
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  ● Execution State Inference
  ● Selective Plaintext Recovery

• End-to-end Attacks
  ● RSA
  ● ECDSA

• Discussion

• Conclusion
Use APIC to get instruction-level ciphertext side channel.
DISCUSSION - HARDWARE FIX

- Restrict software-level access to VM’s memory via RMP.

![Diagram showing restricted access to VM's memory via VMSA and VMCB]

- Adversary cannot read or write to VM’s memory.

Read access

Write access
DISCUSSION - HARDWARE FIX

- Add randomization in stored register value via hardware.

### Register Table

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR3</td>
<td>CR0</td>
</tr>
<tr>
<td>RFLAGS</td>
<td>RIP</td>
</tr>
<tr>
<td>-</td>
<td>RSP</td>
</tr>
<tr>
<td>-</td>
<td>RAX</td>
</tr>
<tr>
<td>CR2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>RCX</td>
</tr>
<tr>
<td>RDX</td>
<td>RBX</td>
</tr>
<tr>
<td>RBP</td>
<td>-</td>
</tr>
<tr>
<td>RSI</td>
<td>RDI</td>
</tr>
<tr>
<td>R8</td>
<td>R9</td>
</tr>
</tbody>
</table>

**0x1f0**

- Randomize
  - Write access
    - Adversary
  - Read access

**16-byte block**
CIPHERLEAKS - OUTLINE

• Ciphertext Side Channel

• Attack Primitives
  • Execution State Inference
  • Selective Plaintext Recovery

• End-to-end Attacks
  • RSA
  • ECDSA

• Discussion

• Conclusion
CONCLUSION

- First exploration of a novel ciphertext side channel on SEV processors.

**CIPHERLEAKS attacks:**
- End-to-end attacks towards constant-time RSA and ECDSA implementation with 100% accuracy.
- First known attack against SEV-SNP.
- AMD will announce a security bulletin and assign a CVE for CIPHERLEAKS (CVE-2020-12966).

- Proposed hardware countermeasures.
Q & A

Mengyuan Li: li.7533@osu.edu
Yinqian Zhang: yinqianz@acm.org