Telling Your Secrets Without Page Faults:

Stealthy Page Table-Based Attacks on Enclaved Execution

Jo Van Bulck[1] Nico Weichbrodt[2] Rüdiger Kapitza[2]
Frank Piessens[1] Raoul Strackx[1]

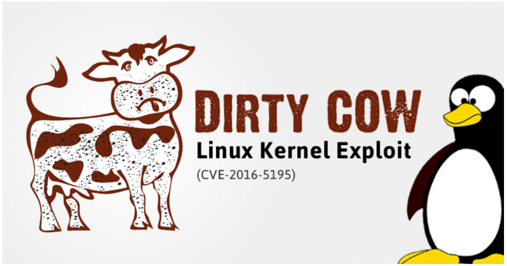
[1]imec-DistriNet, KU Leuven [2]IBR DS, TU Braunschweig

August 18, 2017

Road Map

thehackernews.com/2015/10/windows-patch-update.html

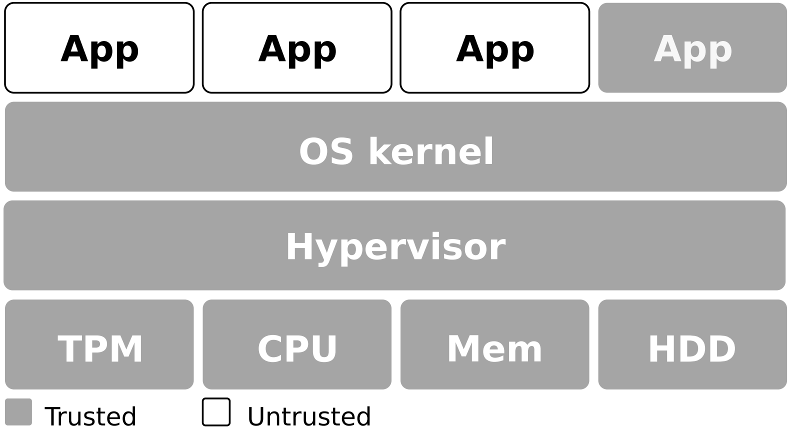
thehackernews.com/2017/06/cia-linux-hacking-tool-malware.html



thehackernews.com/2016/10/linux-kernel-exploit.html

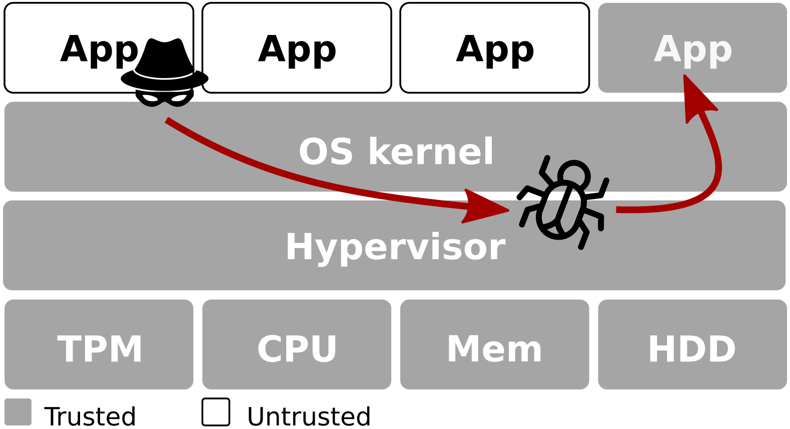
thehackernews.com/2015/04/rootpipe-mac-os-x-vulnerability.html

Motivation: Application Attack Surface



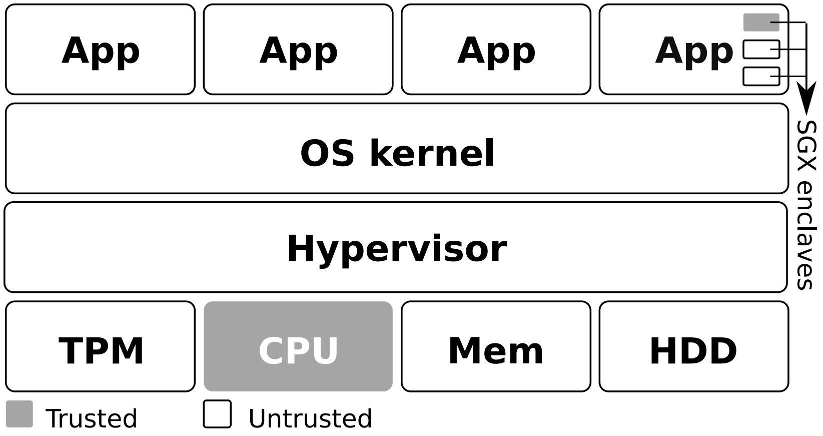
Layered architecture $\rightarrow$ large **trusted computing base**

Motivation: Application Attack Surface



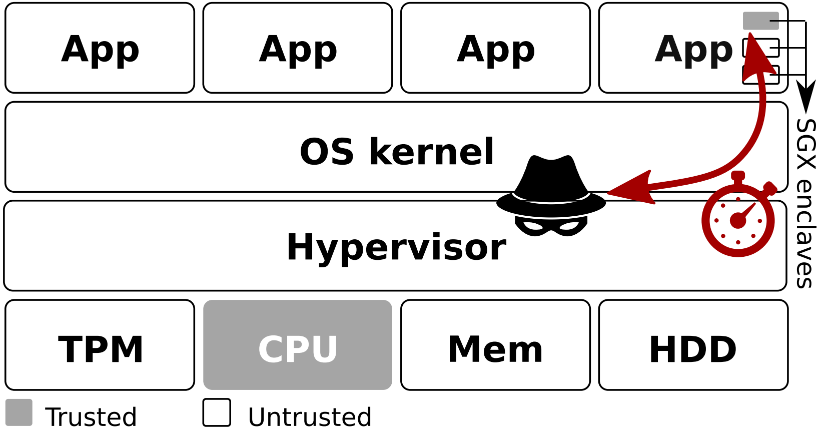
Layered architecture → large **trusted computing base**

Motivation: Application Attack Surface



Intel SGX promise: hardware-level **isolation and attestation**

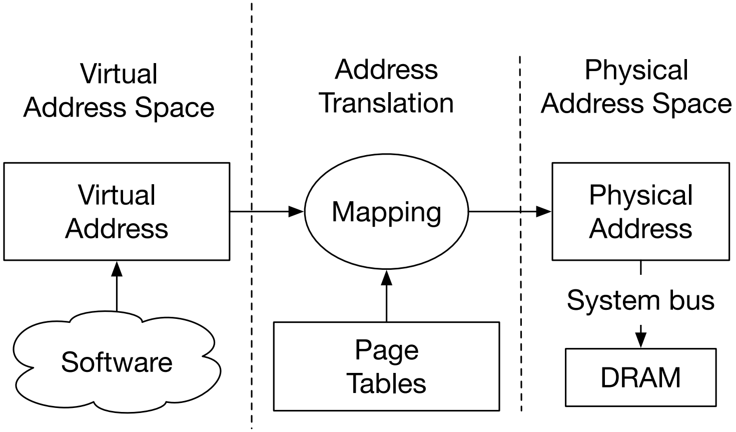
Motivation: Application Attack Surface



Untrusted OS → new class of powerful **side-channels**

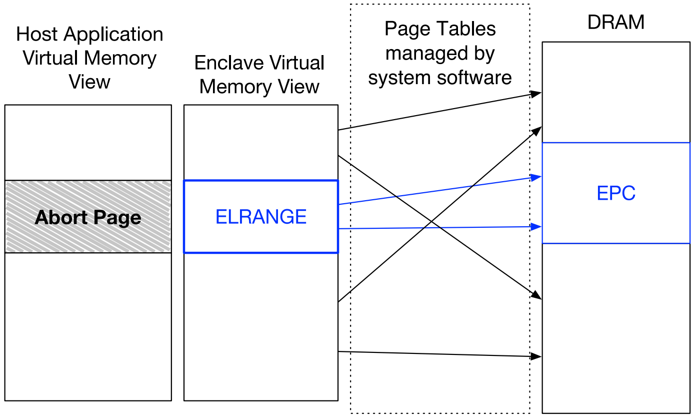
Road Map

The Virtual Memory Abstraction



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

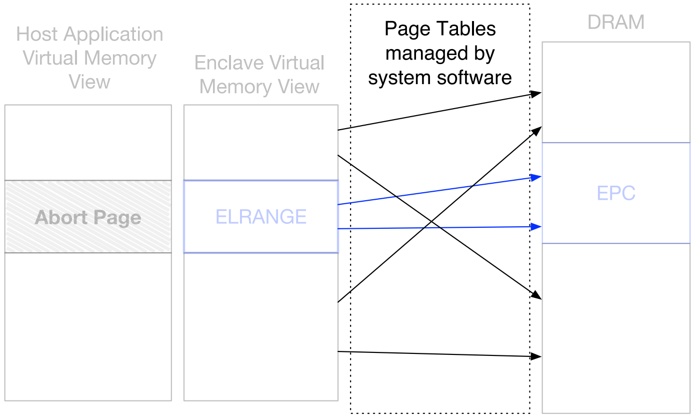
How Enclave Accesses are Enforced


Host Application Virtual Memory View
Abort Page
Enclave Virtual Memory View
ELRANGE
Page Tables managed by system software
DRAM
EPC

Costan et al. "Intel SGX explained", IACR 2016 [CD16]

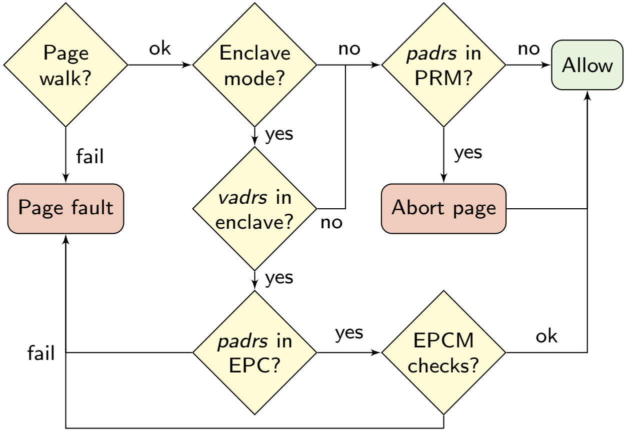
How Enclave Accesses are Enforced

Note: Untrusted OS controls *virtual-to-physical mapping*


Host Application Virtual Memory View
Abort Page
Enclave Virtual Memory View
ELRANGE
Page Tables managed by system software
DRAM
EPC

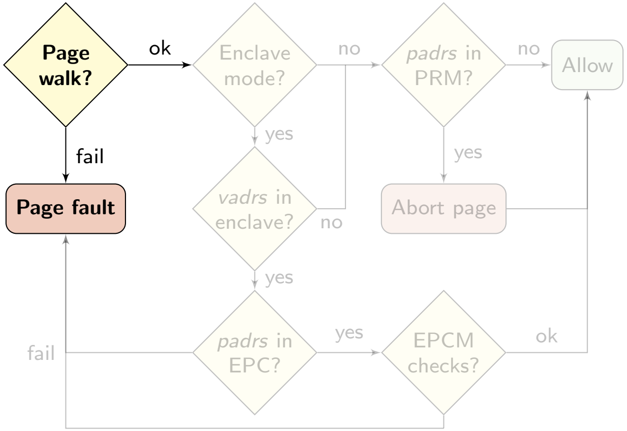
Costan et al. "Intel SGX explained", IACR 2016 [CD16]

How Enclave Accesses are Enforced

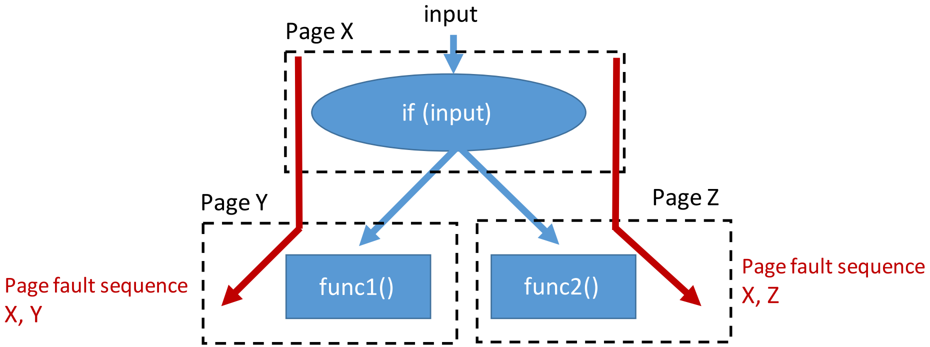


How Enclave Accesses are Enforced

Note: Additional checks *after* address translation



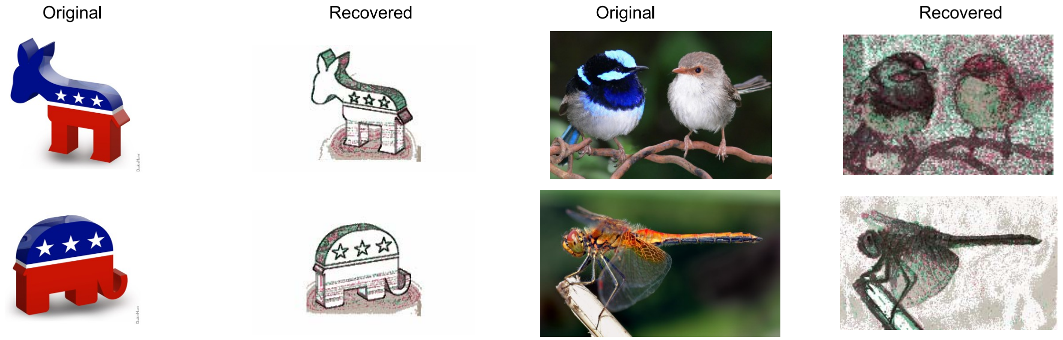
Page Faults as a Side-Channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

⇒ *Page fault traces leak private control flow/data accesses*

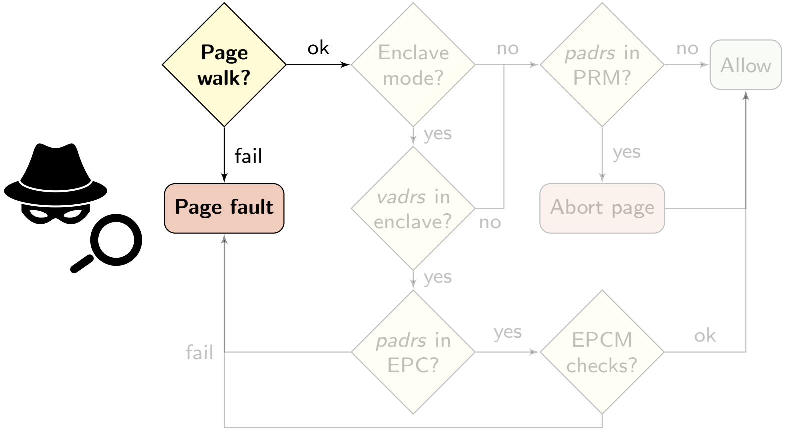
Page Faults as a Side-Channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

$\Rightarrow$ *Low-noise, single-run exploitation of legacy applications*

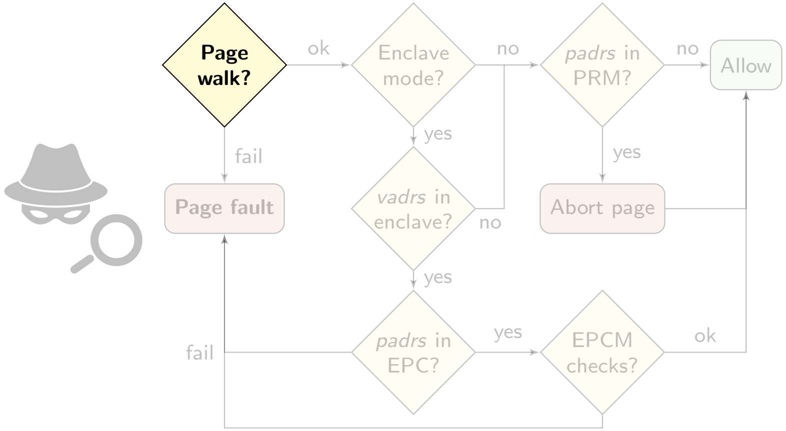
Current Solutions: Hiding Enclave Page Faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

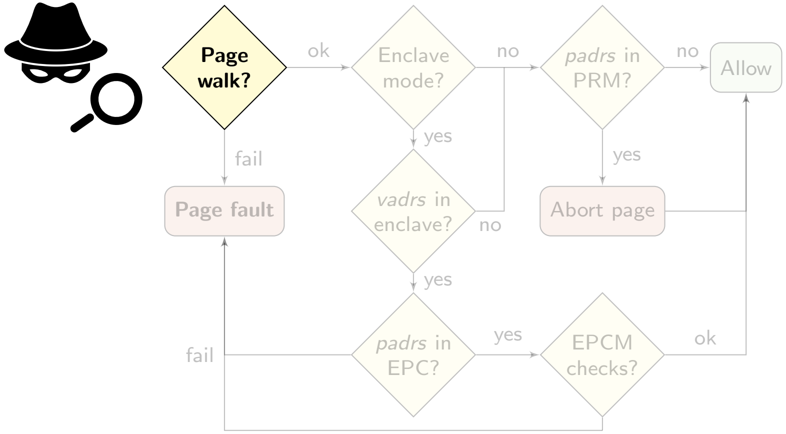
Current Solutions: Hiding Enclave Page Faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

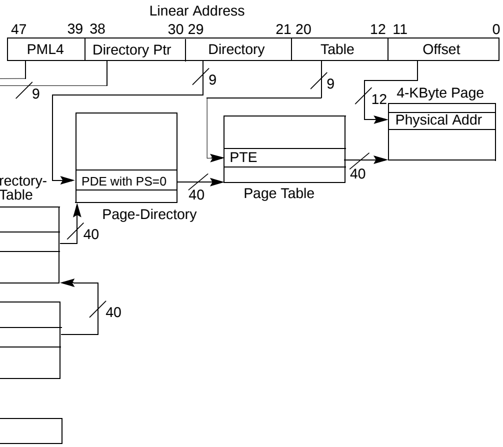
Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

Current Solutions: Hiding Enclave Page Faults



Defenses do not hold when attacker learns page accesses without triggering faults!

Current Solutions: Hiding Enclave Page Faults

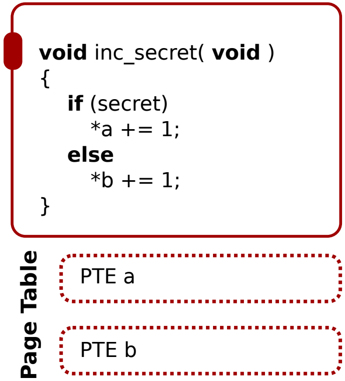


Defenses do not hold when attacker learns page accesses without triggering faults!

Road Map

SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

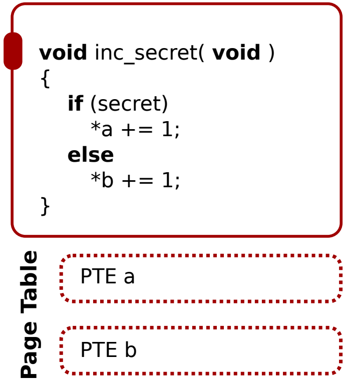


```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!



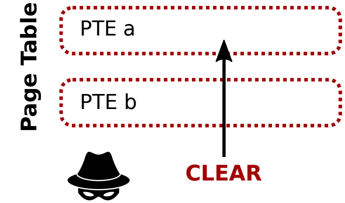
```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

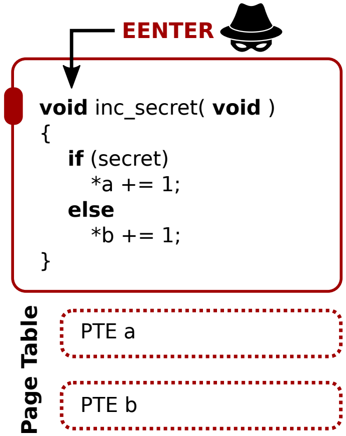
```
void inc_secret( void )
{
    if (secret)
      *a += 1;
    else
      *b += 1;
}
```



SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

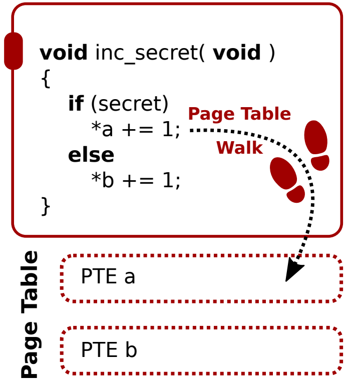


```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

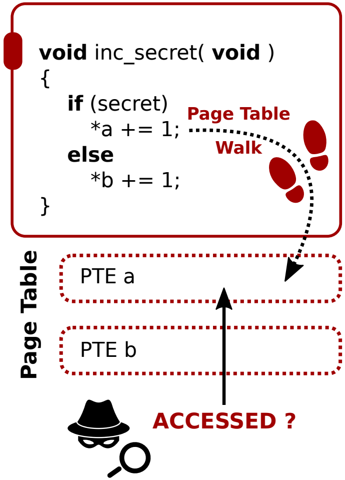
   ⇝ Also updated in enclave mode!


```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

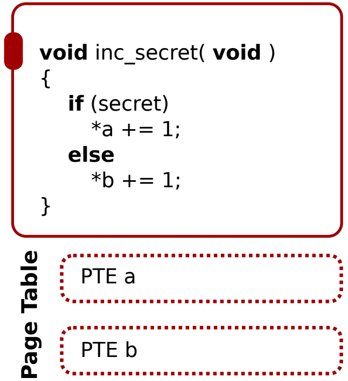


SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

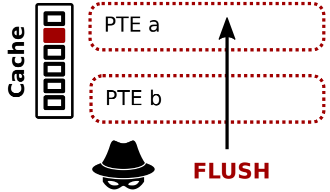
1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```



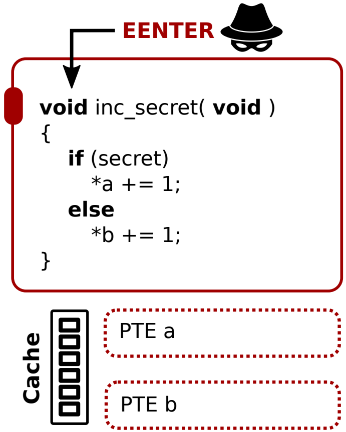
SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

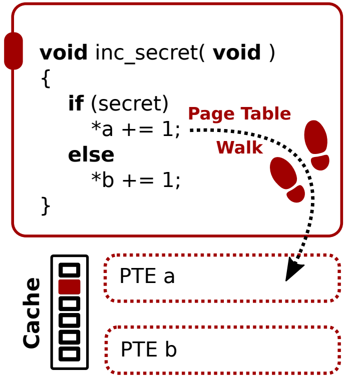
SGX Side-Channel Leakage: Page Table Entries

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



```
void inc_secret( void )
{
    if (secret)
      *a += 1;
    else
      *b += 1;
}
```

SGX Side-Channel Leakage: Page Table Entries

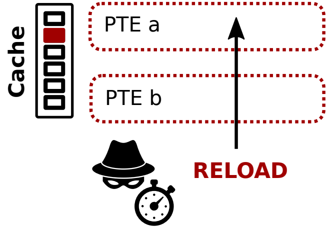
1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

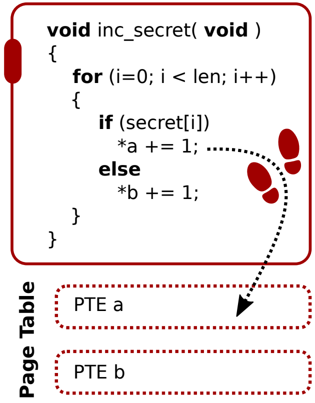
   ⇝ Flush+Reload cache timing attack!

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```



#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access



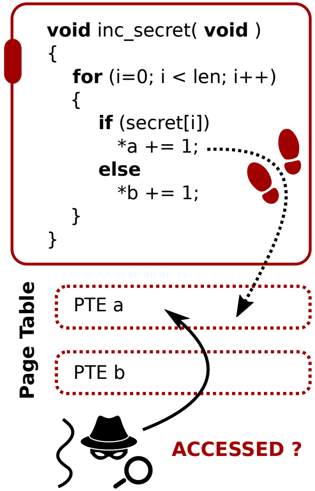
```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
            *a += 1;
        else
            *b += 1;
    }
}
```

SECRET = 01010

#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

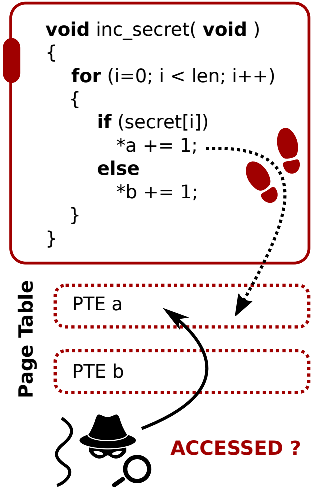

```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
            *a += 1;
        else
            *b += 1;
    }
}
```

#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

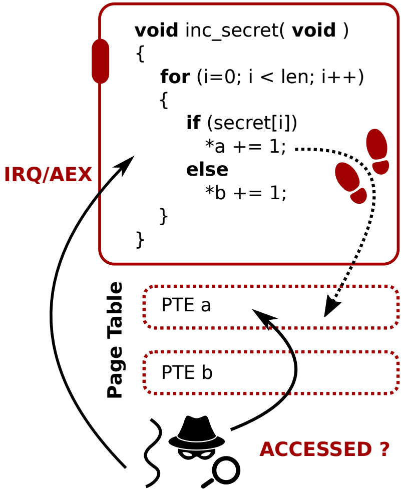
2. Challenge: Translation Lookaside Buffer (TLB)



```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
            *a += 1;
        else
            *b += 1;
    }
}
```

#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access
   ⇝ Monitor PTEs from concurrent **spy thread**
2. Challenge: Translation Lookaside Buffer (TLB)
   ⇝ Directed **Inter-Processor Interrupt**



```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
          *a += 1;
        else
          *b += 1;
    }
}
```

#PF-Less Challenges: Monitoring Repeated Accesses

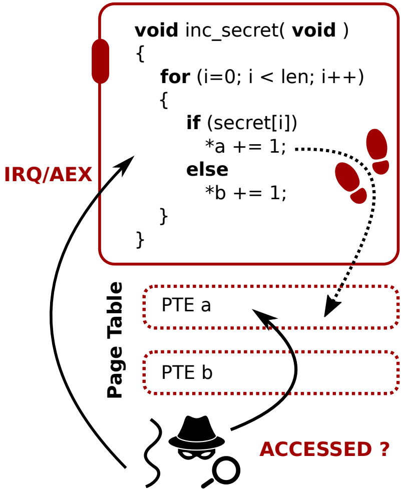
1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

2. Challenge: Translation Lookaside Buffer (TLB)

   ⇝ Directed **Inter-Processor Interrupt**

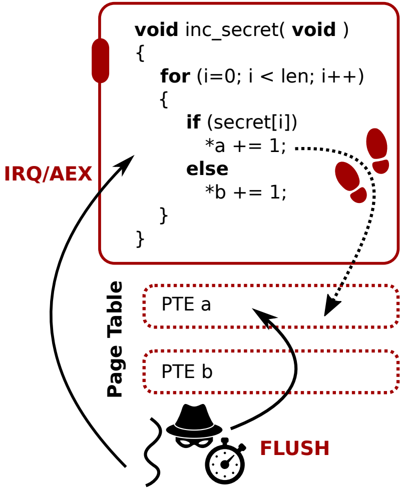
3. Challenge: Temporal resolution (IPI latency)



```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
            *a += 1;
        else
            *b += 1;
    }
}
```

#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access
   ⇝ Monitor PTEs from concurrent **spy thread**
2. Challenge: Translation Lookaside Buffer (TLB)
   ⇝ Directed **Inter-Processor Interrupt**
3. Challenge: Temporal resolution (IPI latency)
   ⇝ Precise **Flush+Flush** technique



```
void inc_secret( void )
{
    for (i=0; i < len; i++)
    {
        if (secret[i])
          *a += 1;
        else
          *b += 1;
    }
}
```

PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Resolution Challenge

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Resolution Challenge

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`

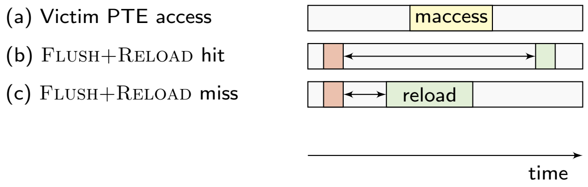
PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Resolution Challenge

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)



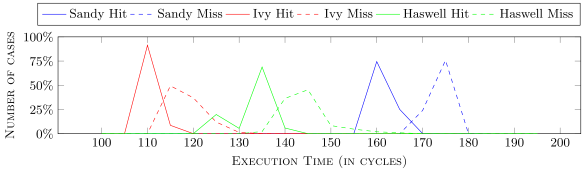
PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Resolution Challenge

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)
- ☺ Flush+Flush: `clflush` completes earlier for uncached data



Gruss et al. “Flush+Flush: a fast and stealthy cache attack”, DIMVA 2016 [GMWM16]

PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Resolution Challenge

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

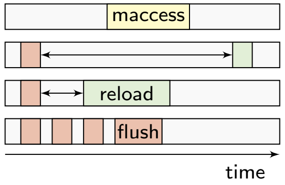
- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)
- ☺ Flush+Flush: interrupt *within trigger instruction* ($>$ 99.8%)

(a) Victim PTE access

(b) Flush+Reload hit

(c) Flush+Reload miss

(d) Flush+Flush hit



Attacking Libgcrypt EdDSA

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

Attacking Libgcrypt EdDSA

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

Attacking Libgcrypt EdDSA

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

EdDSA secret scalar *not* stored in "secure memory" !

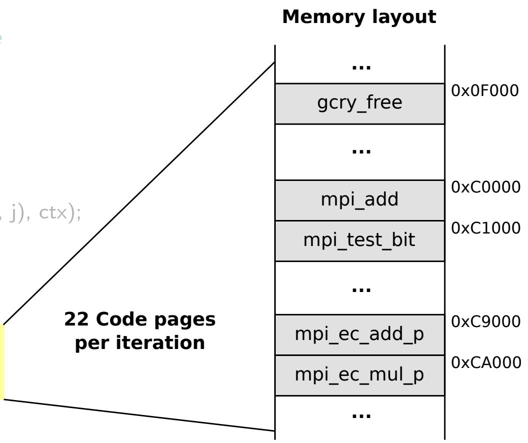
Attacking Libgcrypt EdDSA

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits−1; j >= 0; j−−) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits−1; j >= 0; j−−) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

Secret-dependent control flow

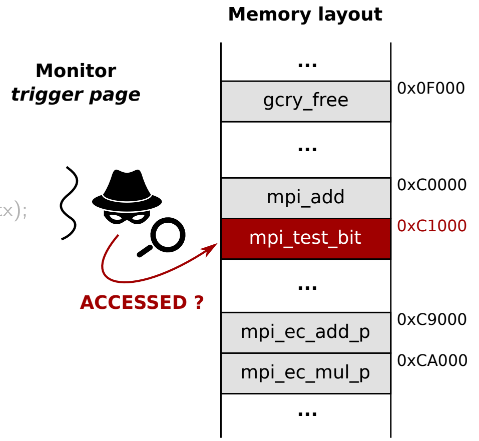
Attacking Libgcrypt EdDSA: A/D Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



Attacking Libgcrypt EdDSA: A/D Channel

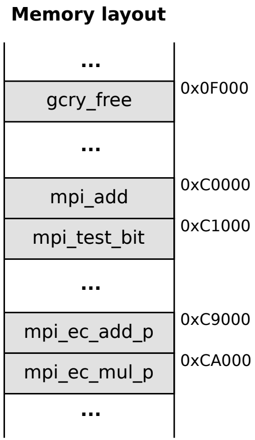
```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



Attacking Libgcrypt EdDSA: A/D Channel

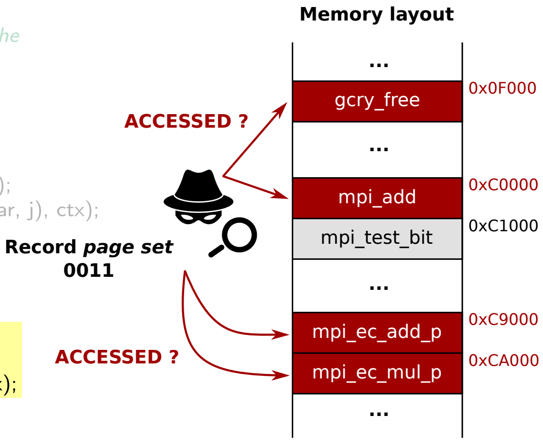
```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

INTERRUPT



Attacking Libgcrypt EdDSA: A/D Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



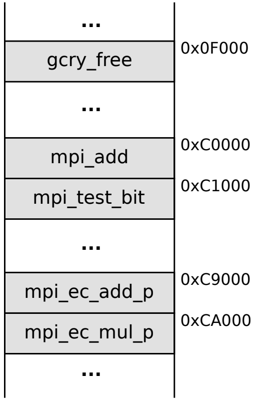
Attacking Libgcrypt EdDSA: A/D Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```

RESUME

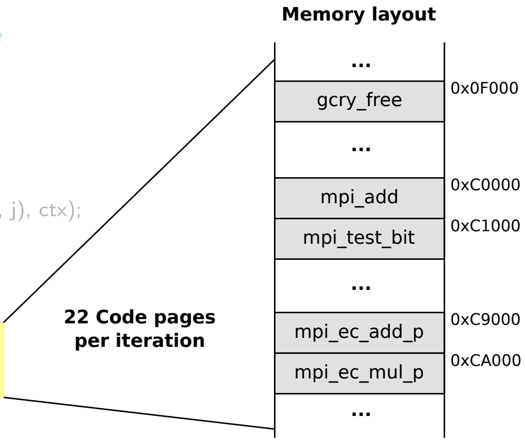
Full 512-bit key recovery, single run

Memory layout



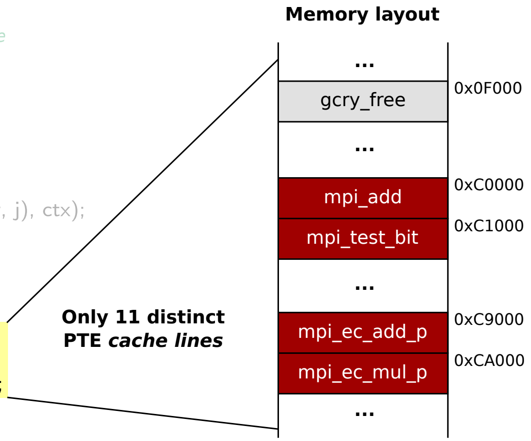
Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



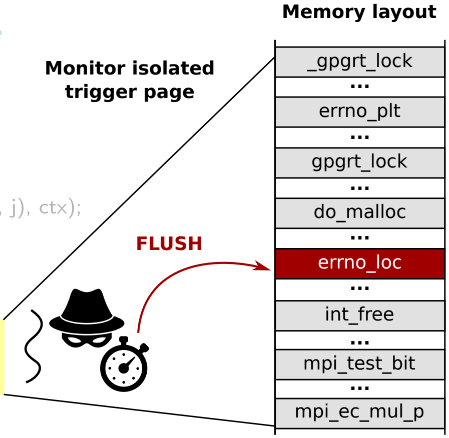
Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



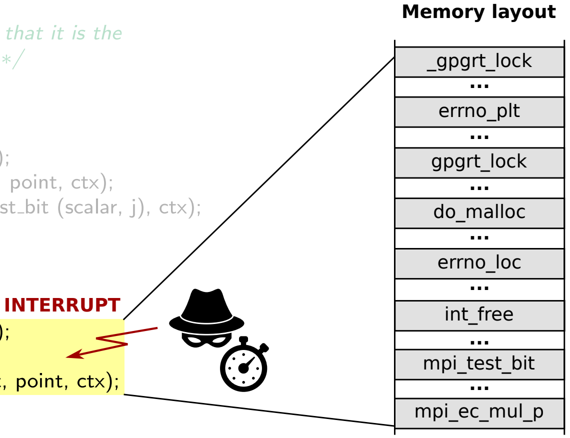
Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



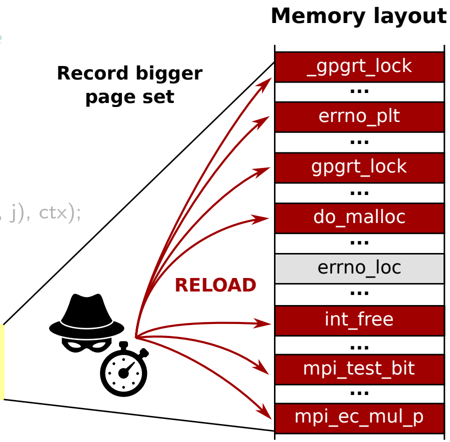
Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits−1; j >= 0; j−−) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits−1; j >= 0; j−−) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



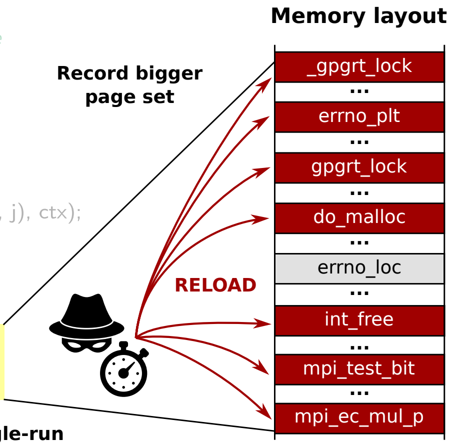
Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



Attacking Libgcrypt EdDSA: Cache-Only Channel

```
if (mpi_is_secure (scalar)) {
    /* If SCALAR is in secure memory we assume that it is the
       secret key we use constant time operation. */
    point_init (&tmppnt);

    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
        point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
    }
    point_free (&tmppnt);
} else {
    for (j=nbits-1; j >= 0; j--) {
        _gcry_mpi_ec_dup_point (result, result, ctx);
        if (mpi_test_bit (scalar, j))
            _gcry_mpi_ec_add_points (result, result, point, ctx);
    }
}
```



Regex pattern match -> 485/512-bit recovery, single-run

Road Map

Conclusion

Take-Away Message

Enclave memory accesses can be learned *without* triggering page faults.

Conclusion

Take-Away Message

Enclave memory accesses can be learned *without* triggering page faults.

$\Rightarrow$ Do not focus on attack **side-effects** (faults, frequent enclave preemptions)

Conclusion

Take-Away Message

Enclave memory accesses can be learned *without* triggering page faults.

⇒ Do not focus on attack **side-effects** (faults, frequent enclave preemptions)

⇒ Address **root causes of information leakage:**

- Unprotected page table memory (Sanctum [CLD16])
- Secret-dependent control flow/data access (Libgcrypt patch)

Thank you! Questions?

`https://github.com/jovanbulck/sgx-pte`

References I

V. Costan and S. Devadas.
Intel SGX explained.
Technical report, Computer Science and Artificial Intelligence Laboratory MIT, 2016.
`https://eprint.iacr.org/2016/086.pdf`.

V. Costan, I. Lebedev, and S. Devadas.
Sanctum: Minimal hardware extensions for strong software isolation.
In *25th USENIX Security Symposium*, pp. 857–874. USENIX Association, 2016.

D. Gruss, C. Maurice, K. Wagner, and S. Mangard.
Flush+flush: A fast and stealthy cache attack.
In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016.

S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena.
Preventing page faults from telling your secrets.
In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, pp. 317–328. ACM, 2016.

M.-W. Shih, S. Lee, T. Kim, and M. Peinado.
T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.
In *24th Annual Network and Distributed System Security Symposium (NDSS)*, 2017.

C.-C. Tsai, D. E. Porter, and M. Vij.
Graphene-SGX: A practical library OS for unmodified applications on SGX.
In *2017 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2017.

References II

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *2015 IEEE Symposium on Security and Privacy*, pp. 640–656. IEEE, 2015.

IPI Latency Microbenchmarks

Table: IPI latency in terms of the number of instructions executed by the victim after accessing the trigger page.

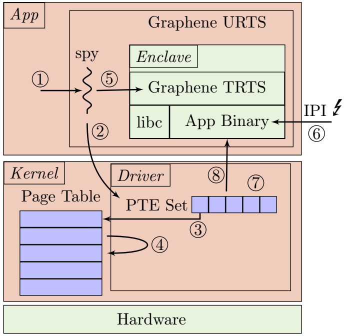
	ACCESSED		FLUSH+FLUSH		
Experiment	**Mean**	σ	**Mean**	σ	**Zero %**
`nop`	431.70	34.11	0.65	17.65	99.84
add register	176.30	14.60	0.15	6.18	99.94
add memory	32.45	2.79	0.06	1.92	99.88
`nop` nocache	0.02	0.39	–	–	–

Putting it All Together: Inferring Page Access Patterns

Re-usable attack framework: Graphene-SGX [TPV17]

- Explicitly monitor **trigger page(s)**
- Capture max info in **page sets** $\leftrightarrow$ #PF-sequences
- Offline analysis: extract **access patterns**

$\Rightarrow$ overcome measurement *noise/ latency/ granularity*


App
Graphene URTS
spy
Enclave
Graphene TRTS
libc
App Binary
IPI
① ② ③ ④ ⑤ ⑥ ⑦ ⑧
Kernel
Driver
Page Table
PTE Set
Hardware