TU Graz
SCIENCE
PASSION
TECHNOLOGY

CounterSEVeillance

Performance-Counter Attacks on AMD SEV-SNP

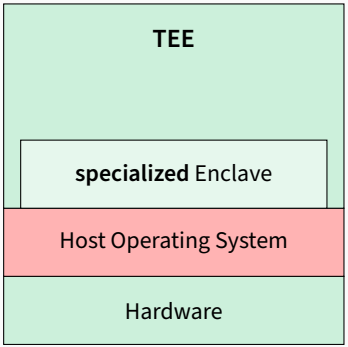
Stefan Gast [1] Hannes Weissteiner [1] Robin Leander Schröder [2,3] Daniel Gruss [1]

[1]Graz University of Technology, Austria
[2]Fraunhofer SIT, Darmstadt, Germany
[3]Fraunhofer Austria, Vienna, Austria

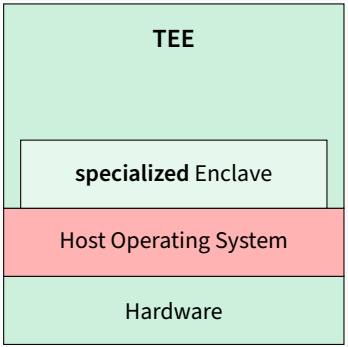
NDSS 2025

› isec.tugraz.at

"Classical" TEEs vs. Confidential VMs

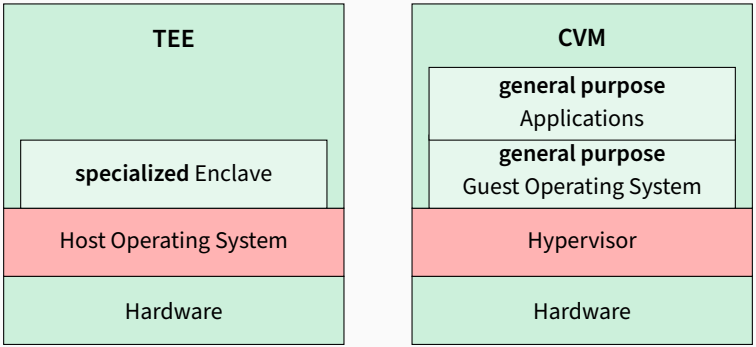


"Classical" TEEs vs. Confidential VMs



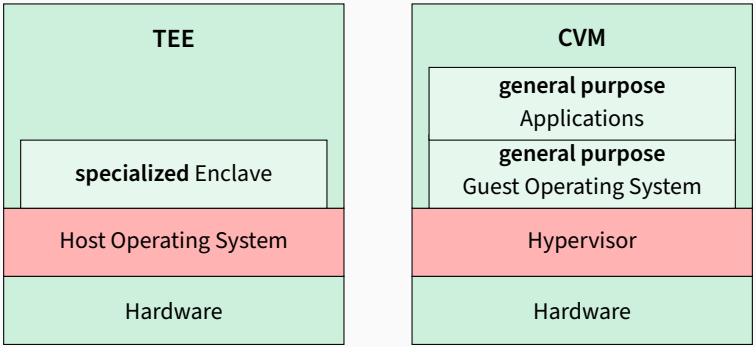
Constant-time programming to protect against side-channels

"Classical" TEEs vs. Confidential VMs



Constant-time programming to protect against side-channels

"Classical" TEEs vs. Confidential VMs



Constant-time programming to protect against side-channels

Hardware Performance Counters

- Counters for specific hardware events

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions
  - Retired Branches

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions
  - Retired Branches
  - Retired Taken Branches

Hardware Performance Counters

- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions
  - Retired Branches
  - Retired Taken Branches
  - Div Cycles Busy

Hardware Performance Counters

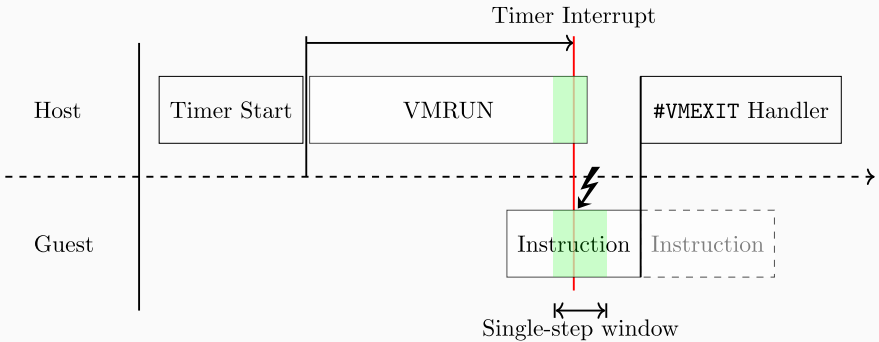
- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions
  - Retired Branches
  - Retired Taken Branches
  - Div Cycles Busy
  - …

Hardware Performance Counters

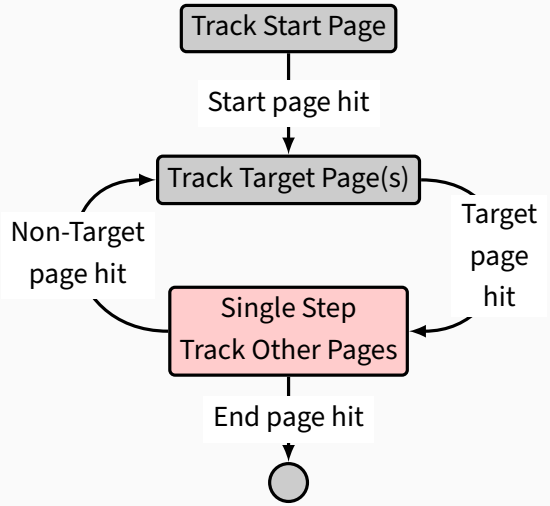
- Counters for specific hardware events
- **Initial observation: HPCs also increment when executing an SEV VM!**
- Examples:
  - Retired Instructions
  - Retired Branches
  - Retired Taken Branches
  - Div Cycles Busy
  - …

Can we get this information for every single instruction?

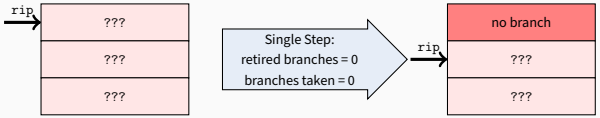
Interrupt-based Single Stepping



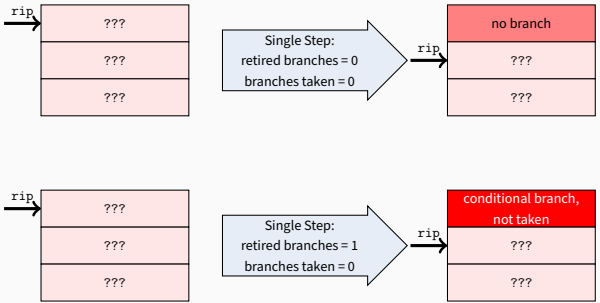
Page-Fault Tracking to Start and Stop Recordings



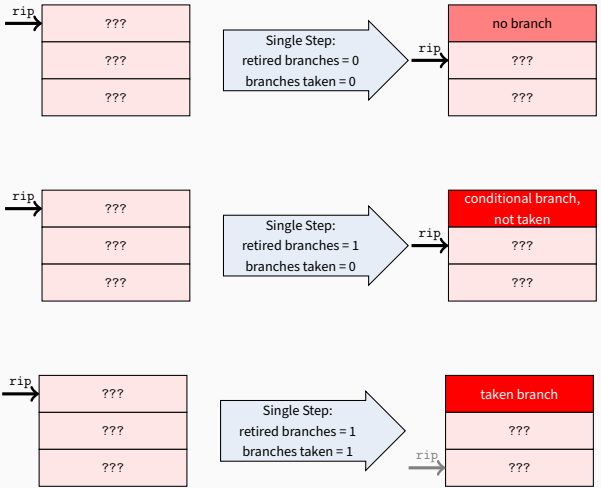
Recovering Control Flow



Recovering Control Flow



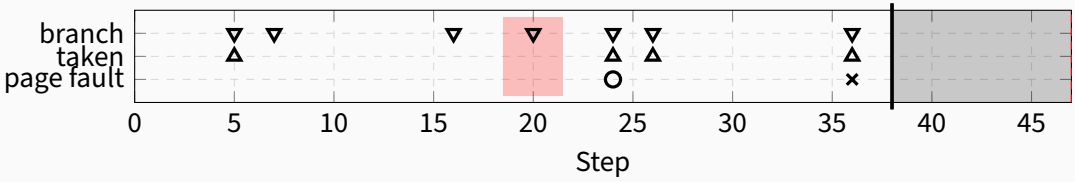
Recovering Control Flow



Breaking Mbed TLS Square + Multiply

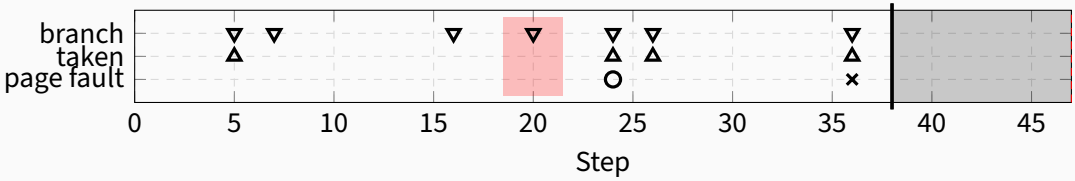

```
for(;;) {
  if (ei == 0 && state == 1) {
    MBEDTLS_MPI_CHK(mpi_select(&WW, W, w_table_used_size, x_index));
    mpi_montmul(&W[x_index], &WW, N, mm, &T);
    continue;
  }
  state = 2;
  MBEDTLS_MPI_CHK(mpi_select(&WW, W, w_table_used_size, x_index));
  mpi_montmul(&W[x_index], &WW, N, mm, &T);
  MBEDTLS_MPI_CHK(mpi_select(&WW, W, w_table_used_size, ei));
  mpi_montmul(&W[x_index], &WW, N, mm, &T);
  state--;
}
```

Breaking Mbed TLS Square + Multiply

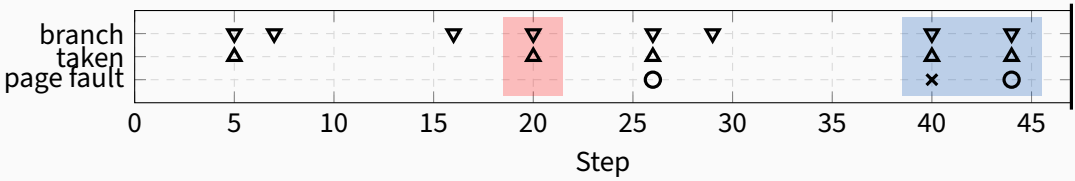


`ei` = 0

Breaking Mbed TLS Square + Multiply



`ei = 0`



`ei = 1`

Breaking Mbed TLS Square + Multiply: Results

Secret recovered:	Full RSA-4096 private key
Average attack runtime:	≈ 7 min
Bit error rate:	$0\,\%$
Success rate:	$100\,\%$

$(n = 10)$

Recovering TOTPs (memcmp-style)

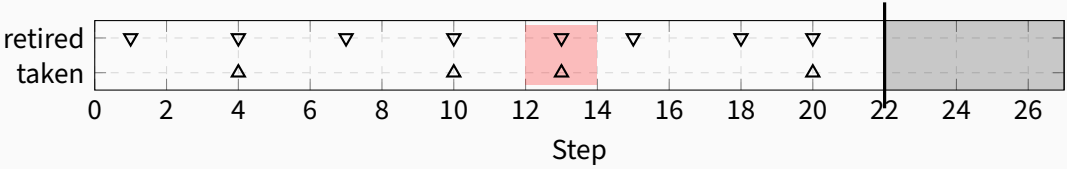
```
COTPRESULT totp_compare(OTPData* data, const char* key,
    int64_t offset, uint64_t for_time)
{
  char time_str[data->digits+1];
  memset(time_str, 0, data->digits+1);
  if (totp_at(data, for_time, offset, time_str) == 0)
    return OTP_ERROR;
  for (size_t i=0; i<data->digits; i++) {
    if (key[i] != time_str[i])
      return OTP_ERROR;
  }
  return OTP_OK;
}
```

Recovering TOTPs (memcmp-style)

```
OTPRESULT totp_compare(OTPData* data, const char* key,
    int64_t offset, uint64_t for_time)
{
  char time_str[data->digits+1];
  memset(time_str, 0, data->digits+1);
  if (totp_at(data, for_time, offset, time_str) == 0)
    return OTP_ERROR;
  for (size_t i=0; i<data->digits; i++) {
    if (key[i] != time_str[i])
      return OTP_ERROR;
  }
  return OTP_OK;
}
```

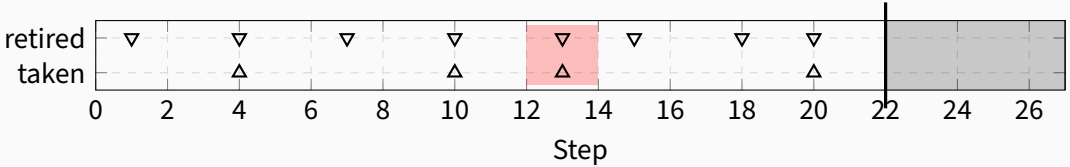
Guess TOTP digit-by-digit, with at most 60 attempts, instead of 1 000 000

Recovering TOTPs (memcmp-style)

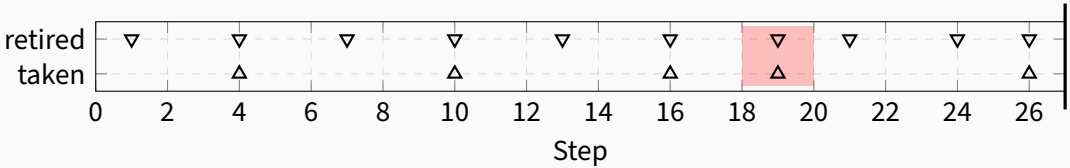


2 correct digits

Recovering TOTPs (memcmp-style)



2 correct digits



3 correct digits

Recovering TOTPs (memcmp-style): Results

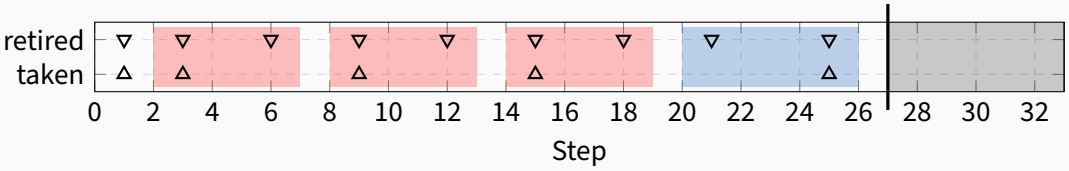
Secret recovered:	6-digit TOTP token
Average attack runtime:	18.14 s
Byte error rate:	0 %
Success rate:	100 %

($n = 50$)

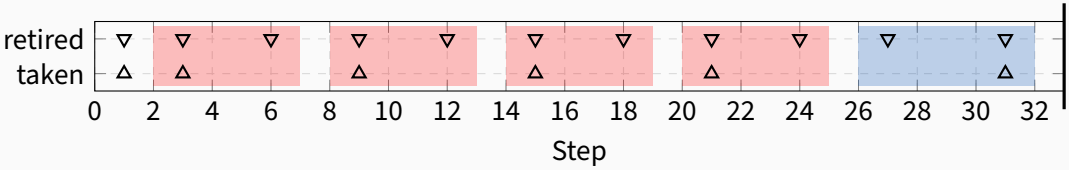
Stealing TOTP secret keys (base32 decoder)

```
static const char OTP_DEFAULT_BASE32_CHARS[32] = { 'A','B','C',... };
COTPRESULT otp_byte_secret(OTPData* data, char* out_str)
{
  for (size_t i = 0; i < num_blocks; i++) {
    unsigned int block_values[8] = { 0 };
    for (int j = 0; j < 8; j++) {
      char c = data->base32_secret[i * 8 + j];
      for (int k = 0; k < 32; k++) {
        if (c == OTP_DEFAULT_BASE32_CHARS[k]) {
          block_values[j] = k;
          break;
        }
      }
    }
  }
}
```

Stealing TOTP secret keys (base32 decoder)



Base32 character is `D`



Base32 character is `E`

Stealing TOTP secret keys (base32 decoder): Results

Secret recovered:	16 base32 character TOTP secret key
Average attack runtime:	< 1 s
Byte error rate:	0 %
Success rate:	86 %
	($n = 86$)

Divide-and-Surrender-style Attack on HQC


```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

Divide-and-Surrender-style Attack on HQC

```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

- Modulo operator $\rightarrow$ `div` instruction with **operand-dependent execution time**

Divide-and-Surrender-style Attack on HQC

```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

- Modulo operator $\rightarrow$ `div` instruction with **operand-dependent execution time**
- On Zen 3 and 4: 1 extra cycle per every 9 bit of division result

Divide-and-Surrender-style Attack on HQC

```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

- Modulo operator $\rightarrow$ `div` instruction with **operand-dependent execution time**
- On Zen 3 and 4: 1 extra cycle per every 9 bit of division result
- `rand_u32[i] < 512 * (17669-i)` $\rightarrow$ 7 cycles; otherwise 8 cycles

Divide-and-Surrender-style Attack on HQC


```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

- Modulo operator $\rightarrow$ `div` instruction with **operand-dependent execution time**
- On Zen 3 and 4: 1 extra cycle per every 9 bit of division result
- `rand_u32[i] < 512 * (17669-i)` $\rightarrow$ 7 cycles; otherwise 8 cycles
- Observable via **Div Cycles Busy** performance counter

Divide-and-Surrender-style Attack on HQC

```
for(size_t i=0; i<75; ++i)
  tmp[i] = i + rand_u32[i] % (17669U - i);
```

- Modulo operator $\rightarrow$ `div` instruction with **operand-dependent execution time**
- On Zen 3 and 4: 1 extra cycle per every 9 bit of division result
- `rand_u32[i] < 512 * (17669-i)` $\rightarrow$ 7 cycles; otherwise 8 cycles
- Observable via **Div Cycles Busy** performance counter
- **Sufficient to build plaintext-checking oracle**

Acknowledgments

This research was made possible by generous funding from:

Supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85), and by the National Research Center for Applied Cybersecurity ATHENE as part of the PORTUNUS project in the research area Crypto. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

TU Graz
SCIENCE
PASSION
TECHNOLOGY

CounterSEVeillance

Performance-Counter Attacks on AMD SEV-SNP

Stefan Gast Hannes Weissteiner Robin Leander Schröder Daniel Gruss

https://stefangast.eu
stefan.gast@tugraz.at
notbobbytables@infosec.exchange

NDSS 2025

› isec.tugraz.at