Cache-timing attacks on cryptographic libraries

Sylvain Guilley, CTO.

Presentation Outline

Presentation Outline

CORPORATE PRESENTATION

OUR ACTIVITY

WHAT DO WE DO?

SECURITY TECHNOLOGIES

FOR EMBEDDED SYSTEMS

FOR WHOM?

CHIPSET/DEVICE VENDORS

IC DESIGN HOUSES

CERTIFICATION LABS

GOVERNMENTAL AGENCIES

FOR WHICH MARKETS?

OUR VISION

Going forward, there will be more and more interconnected devices or objects in various market verticals, this is what we call Internet of Things or Internet of Everything. All those objects being interconnected to the cloud, each and every object could be a threat for the whole network. Therefore the security of the objects or the devices is key. Even more, security will become one of the most important assets of the digital world.

CORPORATE PRESENTATION

YOUR SECURITY PARTNER

THE SECURITY SCIENCE COMPANY

A SOLID BACKGROUND

MORE THAN
15 YEARS OF RESEARCH

MORE THAN
200 PUBLICATIONS

A STRONG RESEARCH FORCE

RESEARCH CENTRES
**IN FRANCE
AND SINGAPORE**

SPIN-OFF FROM
TELECOM PARISTECH

A THOUGHT LEADERSHIP POSITION IN THE SECURITY WORLD

MEMBERSHIP OF
ISO COMMITTEES

3 WORLD-RENOWNED
SCIENTIFIC ADVISORS

WORLWIDE PRESENCE

A COMPLETE PORTFOLIO OF SOLUTIONS

A FULL SET OF
ANALYSIS PLATFORMS

A COMPREHENSIVE SET OF
SECURITY TECHNOLOGIES

CORPORATE PRESENTATION

BUSINESS LINES

PROTECT

THREAT PROTECTION Business Line

EVALUATE

THREAT ANALYSIS Business Line

SERVICE & CERTIFY

THINK AHEAD Business Line

SECURYZR

COMBINATION OF **SMART UNITS** AND **EXPERTISE RESULTS**

LABORYZR

READY-TO-USE **PRE** AND **POST-SILICON ANALYSIS PLATFORMS**

EXPERTYZR

THE **NEXT STEPS** TOWARDS **SECURITY CHALLENGES**

Presentation Outline

NIST PQC Contest (replacement of RSA, ECC, DH)

1. Several proposals: need to evaluate them (security, performance, etc.)
2. Complete submissions published in December 2017: 69 submissions revealed
3. July 2018: 5 submissions retracted, about a dozen need parameter changes
4. Merges... eventually, Feb 2019: round 2
   - Public-key Encryption and Key-establishment Algorithms (17 remaining): BIKE, Classic McEliece, CRYSTALS-KYBER, FrodoKEM, HQC, LAC, LEDAcrypt, NewHope, NTRU, NTRU Prime, NTS-KEM, ROLLO, Round5, RQC, SABER, SIKE, Three Bears.
   - Digital Signature Algorithms (9 remaining): CRYSTALS-DILITHIUM, FALCON, GeMSS, LUOV, MQDSS, Picnic, qTESLA, Rainbow, SPHINCS+.

http://csrc.nist.gov/groups/ST/post-quantum-crypto/

Presentation Outline

Cache Architecture

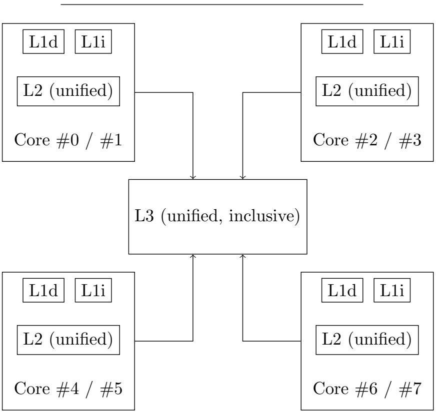


Figure: High-level over of cache architecture (here: modern quadcore Intel processor with Hyperhtreading)

Constant-time operation — detailed investigation

Algorithm 1: RSA implementation (sq. & mult. always) protected against SCA and FIA [1]

Input : $M \in \mathbb{Z}_N$, $d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$ or "Error"

$R[0] \leftarrow r$ (r is a uniform random $\in \mathbb{Z}_N^*$)
$R[1] \leftarrow r^{-1}$
$R[2] \leftarrow M$
for $i \in \{0, \ldots, n-1\}$ **do**
  $R[d_i] \leftarrow R[d_i] \cdot R[2]$
  $R[2] \leftarrow R[2]^2$
if $R[0] \cdot R[1] \cdot M = R[2]$ **then**
  return $r \cdot R[1]$
else
  return "Error"

Algorithm 2: RSA implementation (Montgomery ladder) [2]

Input : $M \in \mathbb{Z}_N$, $d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$

$R[1] \leftarrow 1$
$R[2] \leftarrow M$
for $i \in \{0, \ldots, n-1\}$ **do**
  if $d_i = 1$ **then**
    $R[1] \leftarrow R[1] \cdot R[2]$
    $R[2] \leftarrow R[2] \cdot R[2]$
  else
    $R[2] \leftarrow R[2] \cdot R[1]$
    $R[1] \leftarrow R[1] \cdot R[1]$
return $R[1]$

Constant-time operation — detailed investigation

Algorithm 1: RSA implementation (sq. & mult. always) protected against SCA and FIA [1]

Input : $M \in \mathbb{Z}_N$, $d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$ or "Error"

$R[0] \leftarrow r$ (r is a uniform random $\in \mathbb{Z}_N^*$)
$R[1] \leftarrow r^{-1}$
$R[2] \leftarrow M$
for $i \in \{0, \ldots, n-1\}$ **do**
  $R[d_i] \leftarrow R[d_i] \cdot R[2]$
  $R[2] \leftarrow R[2]^2$
if $R[0] \cdot R[1] \cdot M = R[2]$ **then**
  return $r \cdot R[1]$
else
  return "Error"

Algorithm 2: RSA implementation (Montgomery ladder) [2]

Input : $M \in \mathbb{Z}_N$, $d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$

$R[1] \leftarrow 1$
$R[2] \leftarrow M$
for $i \in \{0, \ldots, n-1\}$ **do**
  if $d_i = 1$ **then**
    $R[1] \leftarrow R[1] \cdot R[2]$
    $R[2] \leftarrow R[2] \cdot R[2]$
  else
    $R[2] \leftarrow R[2] \cdot R[1]$
    $R[1] \leftarrow R[1] \cdot R[1]$
return $R[1]$

Not constant time due to (cached or predicted) table accesses.

Not constant time due to unbalanced `load` operations.

Constant-time operation — detailed investigation

Algorithm 3: Constant-time RSA implementation

Input : $M \in \mathbb{Z}_N$, $d = (d_{n-1}, \cdots, d_0)_2$

Output: $M^d \in \mathbb{Z}_N$

```
R[1] ← 1
R[2] ← M
for i ∈ {0, ..., n − 1} do
    mask ← -!!(d_i)
    R[1] ← R[1] · ((R[2] ∧ mask) ∨ (1 ∧ ¬mask))
    R[2] ← R[2]^2
return R[1]
```

- Notice that mask $= $ `-!!`(d_i) is equal to either `0x0000...00` or `0xFFFF...FF`
- **Horizontally** aligned, thus subsequently attacked based on **values**:
- Not vertically secure.
- See attack [3]: *One&Done: A Single-Decryption EM-Based Attack on OpenSSL's Constant-Time Blinded RSA*, by Monjur Alam et al. (same attack on windowed implem.)

Measurement Techniques — Contention

Feature \ Attack	PRIME + PROBE[1]	EVICT + TIME [2]	FLUSH + RELOAD[2]
Target	L1 / L3[3]	L1	LLC (L3)
Behavior	$\neg$(V access) $\Longrightarrow$ (A fast)	(V access) $\Longrightarrow$ (A fast)	(V access) $\Longrightarrow$ (A fast)
Accuracy	small	small	high
False positives	many	many	few

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES", , in *Cryptographers Track at the RSA Conference*, Springer, 2006, pp. 1–20.

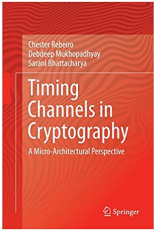
[2] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, L3 cache side-channel attack.", in *USENIX Security Symposium*, 2014, pp. 719–732.

[3] F. Liu, Y. Yarom, Q. Ge, *et al.*, "Last-level cache side-channel attacks are practical", in *Security and Privacy (SP), 2015 IEEE Symposium on*, IEEE, 2015, pp. 605–622.

Taxonomy [9]

- EVICT + TIME
- PRIME + PROBE
- FLUSH + RELOAD
- FLUSH + TIME + FLUSH
- FLUSH + GAUSS + RELOAD [7]
- CacheBleed [8]

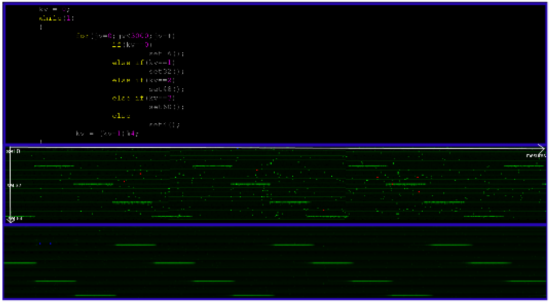
Best state-of-the-art:



Timing Channels in Cryptography — A Micro-Architectural Perspective
Authors: Rebeiro, Chester, Mukhopadhyay, Debdeep, Bhattacharya, Sarani.
`http://www.springer.com/us/book/9783319123691`

Timing attacks

- Branches
- Caches (data, code)



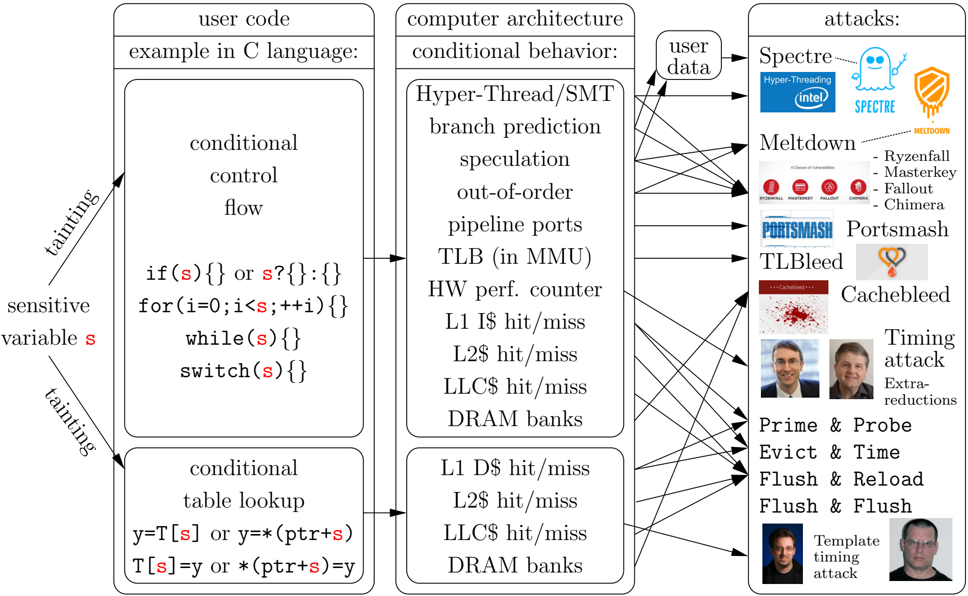
Feature	**External side-channel**	**Timing side-channel**
Possibility to profile	yes	yes
Sampling rate	10 Gsample/s	10 Msample/s
Diversity	Monovariate	Multivariate*
Signal-to-noise ratio	10^{-3}	10

*: probing of I/D cache of various levels, MMU, branch prediction registers, HPC (Hardware Perf Counters), etc.

Vulnerabilities List

What / How	Data	Code
Read	Yes	Yes
Write	Yes	No
Exploit	Sensitive indirections	Conditional jump/call

Note: FLUSH + RELOAD only applicable to **shared** data or code (static arrays, code in shared dynamic libraries, etc.)



user code
example in C language:
conditional
control
flow
if(s){} or s?{}:{}
for(i=0;i<s;++i){}
while(s){}
switch(s){}
conditional
table lookup
y=T[s] or y=*(ptr+s)
T[s]=y or *(ptr+s)=y
sensitive
variable s
tainting
tainting
computer architecture
conditional behavior:
Hyper-Thread/SMT
branch prediction
speculation
out-of-order
pipeline ports
TLB (in MMU)
HW perf. counter
L1 I$ hit/miss
L2$ hit/miss
LLC$ hit/miss
DRAM banks
L1 D$ hit/miss
L2$ hit/miss
LLC$ hit/miss
DRAM banks
user data
attacks:
Spectre
Hyper-Threading
intel
SPECTRE
MELTDOWN
Meltdown
- Ryzenfall
- Masterkey
- Fallout
- Chimera
PORTSMASH
Portsmash
TLBleed
Cachebleed
Cachebleed
Timing attack
Extra-reductions
Prime & Probe
Evict & Time
Flush & Reload
Flush & Flush
Template timing attack

Presentation Outline

Vulnerability Research

Possible Approaches

- Dynamic analysis: `valgrind` "abuse" (sensitive = non-initialized variables)
- Static analysis:
  - On assembly code: `cacheaudit` (only a subset of 32-bit assembler)
  - On intermediary language: `ct-verif` (safe and sound, but not easy to use)
  - On C source code:
    - `tis-ct` (but no way to tag variables as sensitive)
    - **Catalyzr™ tool** [10], based on Alexander Schaub's Stanalyzr

Vulnerability Research

General Approach

- Potential vulnerabilities if sensitive value:
  - Used to index in array (or in general, to compute address in memory to be addressed)
  - Used in a branching operation

- Thus, vulnerability research needs to:
  - Propagate sensitive values,
  - Determine if value used in leaking operation

Vulnerability Research

Methodology — Catalyzr™ tool

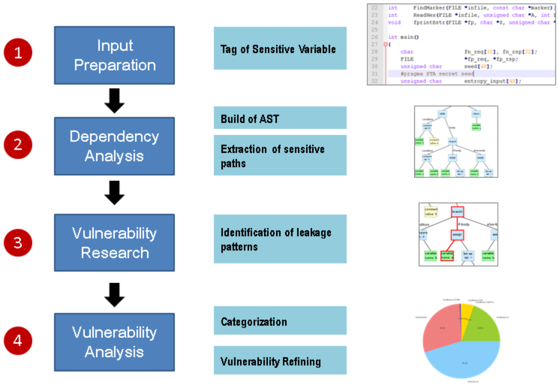


Figure: Static analysis tool principle

Vulnerability Research

Methodology

- All submissions must implement a function to generate KAT files (for signature, encryption or encapsulation)
- Tagged as sensitive: randomness used in generating KAT files (thus also secret keys)
- KAT-generating function (for the Reference Implementation) were analyzed

Note: this might lead to false positives (leakage of non-sensitive values possibly reported)

Results [10]

Vulnerable Implementations

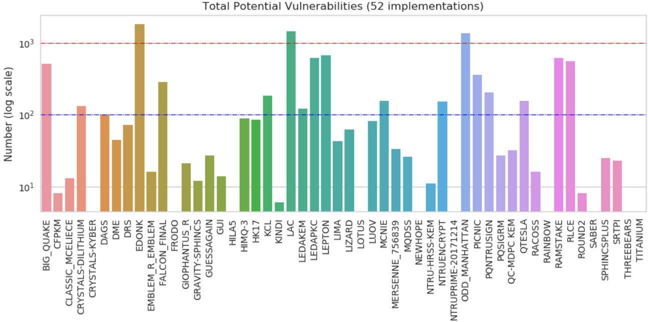


Figure: Total number of potential vulnerabilities found for each analyzed round #1 candidate

Note: 52 out of the 69 submissions were analyzed.

Results

Vulnerable Implementations

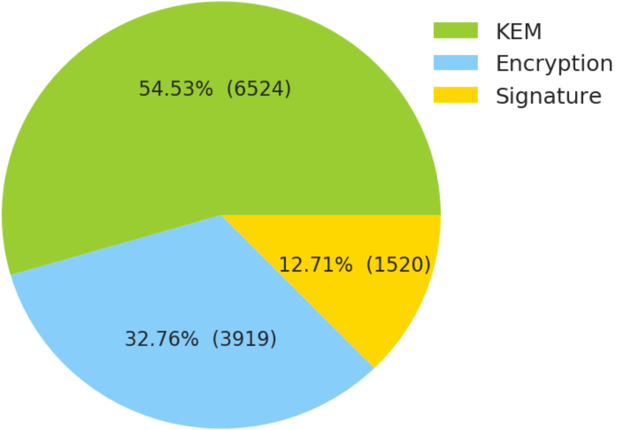
[10]

Out of 52 analyzed candidates:

- Potential vulnerabilities in **42** submissions (80.8%)
  - More than 100 reported vulnerabilities in **17** submissions
  - More than 1000 reported vulnerabilities in **3** submissions
- 4 submissions with easily fixable / probably not exploitable vulnerabilities (EMBLEM, Lima, Giophantus, OKCN-AKCN in the MLWE variant)
- 10 submissions without detected vulnerabilities (Frodo, Rainbow, Hila5, Saber, CRYSTALS-Kyber, LOTUS, NewHope, ntruprime, ThreeBears and Titanium)

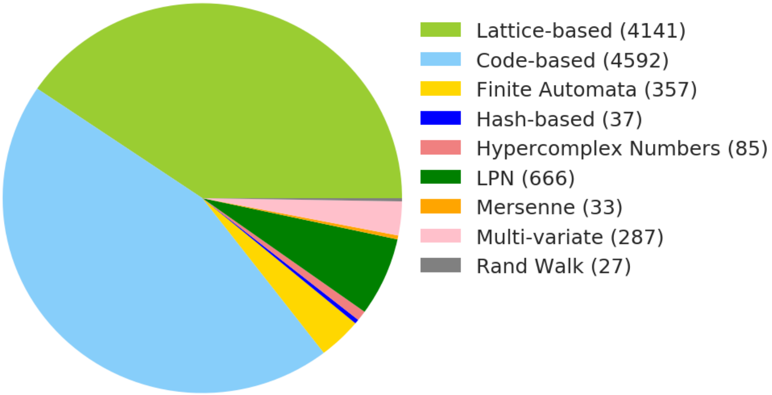
Results

Statistics (raw data—not divided by the number of submissions)



Results

Statistics (raw data—not divided by the number of submissions)



Results

Overview Per Type of Vulnerabilities

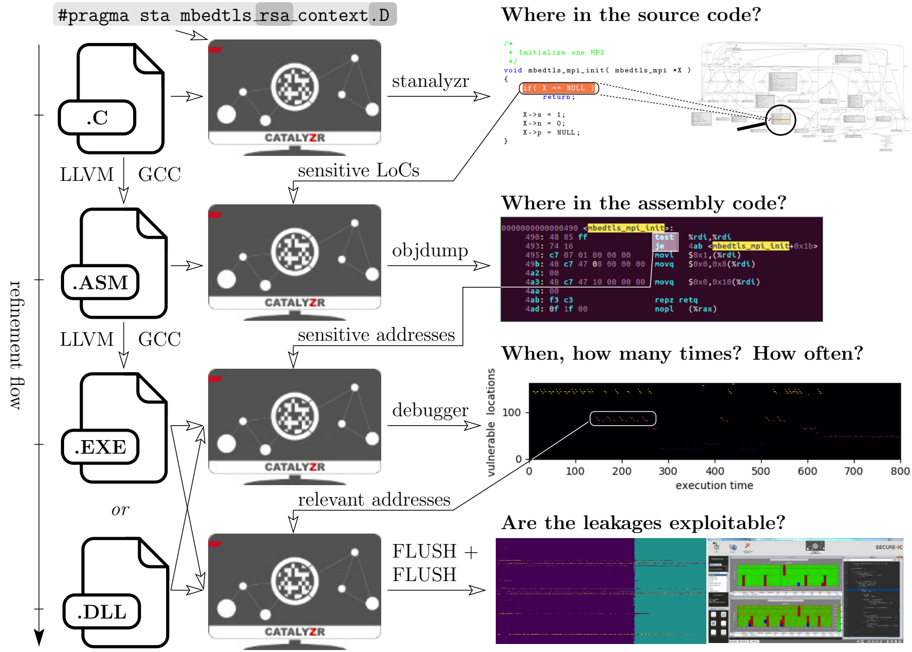
Vulnerability Type	Occurrences
Gaussian sampling	3
Other sampling	13
Use of GMP	4
Insecure GF operations	12
Other leakage sources (incl. error correction)	31

Table: Breakdown of vulnerabilities per type.

Note: the 10 constant-time submissions of round 1 are:

- 7 (out of 17 of Round 2 remaining candidates) for Encryption and KEM;
- 1 (out of 9 of Round 2 remaining candidates) for Digital Signature;
- 2 (namely: LOTUS and Titanium) have been rejected from the competition

Presentation Outline



#pragma sta mbedtls_rsa_context.D
.C
LLVM
GCC
.ASM
LLVM
GCC
.EXE
or
.DLL
refinement flow
CATALYZR
stanalyzr
sensitive LoCs
objdump
sensitive addresses
debugger
relevant addresses
FLUSH + FLUSH
Where in the source code?
Where in the assembly code?
When, how many times? How often?
vulnerable locations
execution time
Are the leakages exploitable?

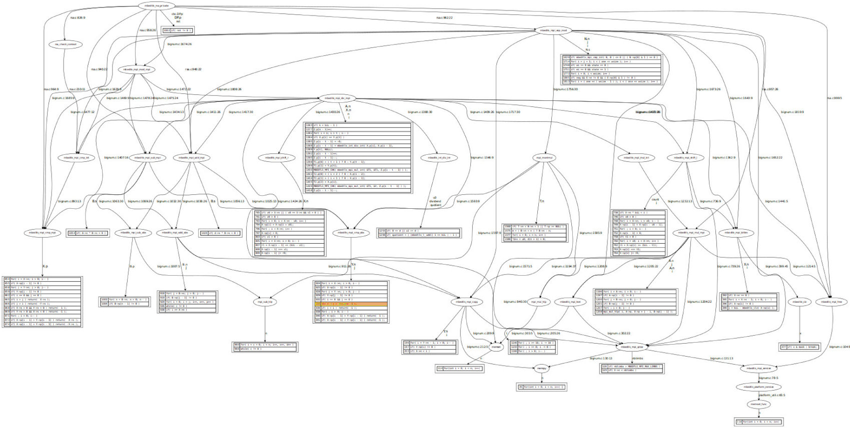
Sensitive RSA function, `mbedtls-2.14.0/library/rsa.c`

```c
/*
 * Exponent blinding supposed to prevent side-channel attacks using multiple
 * traces of measurements to recover the RSA key. The more collisions are there,
 * the more bits of the key can be recovered. See [3].
 *
 * Collecting n collisions with m bit long blinding value requires 2^(m-m/n)
 * observations on avarage.
 *
 * For example with 28 byte blinding to achieve 2 collisions the adversary has
 * to make 2^112 observations on avarage.
 *
 * (With the currently (as of 2017 April) known best algorithms breaking 2048
 * bit RSA requires approximately as much time as trying out 2^112 random keys.
 * Thus in this sense with 28 byte blinding the security is not reduced by
 * side-channel attacks like the one in [3])
 *
 * This countermeasure does not help if the key recovery is possible with a
 * single trace. */
#define RSA_EXPONENT_BLINDING 28

/*
 * Do an RSA private key operation
 */
int mbedtls_rsa_private( mbedtls_rsa_context *ctx,
                 int (*f_rng)(void *, unsigned char *, size_t),
                 void *p_rng,
                 const unsigned char *input,
                 unsigned char *output )
```

MbedTLS leakage in RSA signature function [11]

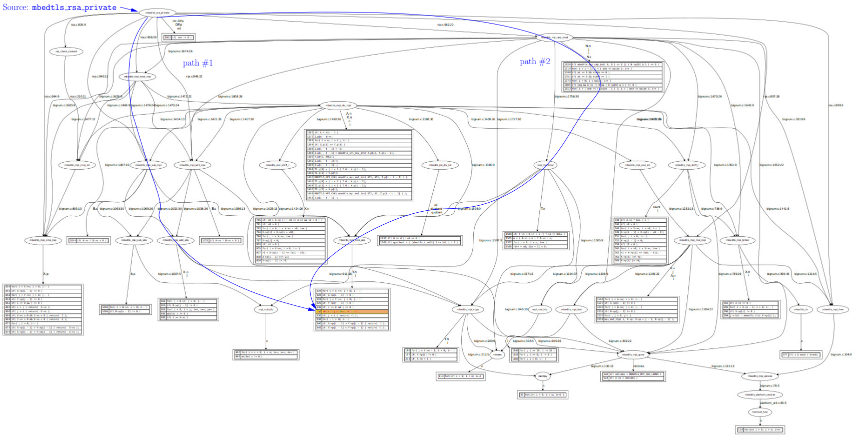
Most of the leakages are in file `bignum.c` (output of Catalyzr™ tool)



There are multiple paths to activate a vulnerability

MbedTLS leakage in RSA signature function [11]

Most of the leakages are in file `bignum.c` (output of Catalyzr™ tool)



There are multiple paths to activate a vulnerability

Symbolic static over-estimated analysis

No undetected leakages, but some false positives

False positives are rare because the assumptions made in static analysis are tight in cryptography. Indeed, intermediate values are randomized. Example of *unlikely* false positives are:

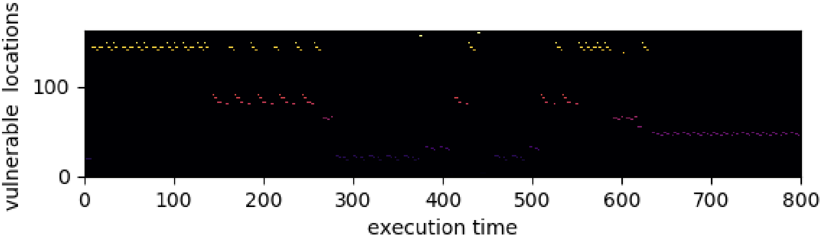
- sensitive annihilation is not understood, e.g., `s` ⊕ `s` is null but still considered sensitive
- dead code can be analyzed if present in obfuscated blocks, e.g., `if(tricky_condition_that_happens_to_be_always_zero) { y=f(s)... }`
- output of a hash function is sensitive but unexploitable
- conditional code turned into constant-time assembly
- let `u` be unsensitive and `s` be sensitive, then `*(s+u) = s[u] = u[s]` is sensitive, but
  - `u[s]` is sensitive (e.g., SubBytes table lookup), and
  - `s[u]` is unsensitive (e.g., `s[7]»31` is the unsensitive evaluation of the MSB of a 256-bit nonce `s` to be used for ECDSA).

SECURE-IC

Translation of cache-timing vulnerable C operations into assembly

C construct	Pseudo-assembly construct	Vulnerable?
`if(s){}`	`cmov s` or `setcc s`	no
`if(s){}`	`test s` `jump` *`address`*	yes
`for(i=0;i<s;++i){}`		
`while(s){}`		
`switch(s){}`		
`y=T[s]` or `y=*(ptr+s)`	`load s`	yes
`T[s]=y` or `*(ptr+s)=y`	`store s`	yes

Dynamic profiling with Catalyzr™ of the 160+ vulnerable LoC in `bignum.c` .... [12]



- Functions and loops can be identified ⟹ useful feedback to developer.

```c
/*
 * Conditionally assign X = Y, without leaking information
 * about whether the assignment was made or not.
 * (Leaking information about the respective sizes of X and Y is ok however.)
 */
int mbedtls_mpi_safe_cond_assign( mbedtls_mpi *X, const mbedtls_mpi *Y,
                unsigned char assign )
{
    int ret = 0;
    size_t i;

    /* make sure assign is 0 or 1 in a time-constant manner */
    assign = (assign | (unsigned char)-assign) >> 7;

    MBEDTLS_MPI_CHK( mbedtls_mpi_grow( X, Y->n ) );

    X->s = X->s * ( 1 - assign ) + Y->s * assign;

    for( i = 0; i < Y->n; i++ )
        X->p[i] = X->p[i] * ( 1 - assign ) + Y->p[i] * assign;

    for( ; i < X->n; i++ )
        X->p[i] *= ( 1 - assign );

cleanup:
    return( ret );
}
```

```c
/*
 * Signed addition: X = A + B
 */
int mbedtls_mpi_add_mpi( mbedtls_mpi *X, const mbedtls_mpi *A, const mbedtls_mpi *B )
{
    int ret, s = A->s;

    if( A->s * B->s < 0 )
    {
        if( mbedtls_mpi_cmp_abs( A, B ) >= 0 )
        {
            MBEDTLS_MPI_CHK( mbedtls_mpi_sub_abs( X, A, B ) );
            X->s =  s;
        }
        else
        {
            MBEDTLS_MPI_CHK( mbedtls_mpi_sub_abs( X, B, A ) );
            X->s = -s;
        }
    }
    else
    {
        MBEDTLS_MPI_CHK( mbedtls_mpi_add_abs( X, A, B ) );
        X->s = s;
    }

cleanup:

    return( ret );
}
```

```c
/*
 * Montgomery multiplication: A = A * B * R^-1 mod N  (HAC 14.36)
 */
static int mpi_montmul( mbedtls_mpi *A, const mbedtls_mpi *B, const mbedtls_mpi *N,
                        mbedtls_mpi_uint mm, const mbedtls_mpi *T )
{
    // [...]

    if( mbedtls_mpi_cmp_abs( A, N ) >= 0 )
        mpi_sub_hlp( n, N->p, A->p );
    else
        /* prevent timing attacks */
        mpi_sub_hlp( n, A->p, T->p );

    return( 0 );
}
```

Presentation Outline

Issues with multiplication in $GF(2^N)$ extensions [13]

Table 1: PQC submissions at NIST using vulnerable $GF(2^N)$ operations

Submission	Type	Finite Group	Tower fields used
BIG QUAKE	Code-based	$GF(2^N), N = 12, 18$	No
DAGS	Code-based	$GF((2^5)^2), GF((2^6)^2)$	Yes
EdonK	Code-based	$GF(2^N), N = 128, 192$	No
Ramstake	Code-based	$GF(2^8)$	No
RLCE	Code-based	$GF(2^N), N = 10, 11$	No
LAC	Lattice-based	$GF(2^N), N = 9, 10$	No
DME	Multivariate	$GF(2^N), N = 24, 48$	No
HIMQ-3	Multivariate	$GF(2^8)$	No
LUOV	Multivariate	$GF(2^8), GF((2^{16})^l), l = 3, 4, 5$	Yes

Vulnerable $a \cdot b$ in C: `a ? antilog[log[a]+log[b]] : 0.`

Issues with multiplication in $GF(2^N)$ extensions [13]

Table 3: Performances (in clock cycles) of several multiplication algorithms

Multiplication algorithm	**Alg.**	$GF(2^6)$	$GF(2^{12})$ **(*)**	$GF((2^6)^2)$	**Constant-time?**
Tabulated log/antilog (**)	3, 4	8	11	20	No
Iterative, conditional reduction	5	27	51	133	No
Iterative, ASM with PCLMUL, conditional reduction	5	29	41	146	No
Iterative, unconditional reduction	6	30	58	155	Yes
Iterative, ASM with PCLMUL, unconditional reduction	6	35	65	225	Yes
Iterative, unconditional reduction, 1-bit-sliced (***) 64 computations in parallel	7	55/64	335/64	-	Yes
Iterative, ASM with PCLMUL, unconditional reduction, bit-sliced (****) 2 computations in parallel	8	55/2	95/2	-	Yes

Example of implementation in bitslice

```c
/* 7.2: 1-bit-sliced multiplication (SIMD code) */
void gf_multsubTab(gf_t *x, gf_t *y, gf_t *z)
{
    uint64_t xbin[6];
    uint64_t ybin[6];
    uint64_t res[6];

    xbin[0]=xbin[1]=xbin[2]=xbin[3]=xbin[4]=xbin[5] = 0;
    ybin[0]=ybin[1]=ybin[2]=ybin[3]=ybin[4]=ybin[5] = 0;
```

```c
    res[0] = (xbin[0] & ybin[0]);
    res[1] = (xbin[1] & ybin[0]);
    res[2] = (xbin[2] & ybin[0]);
    res[3] = (xbin[3] & ybin[0]);
    res[4] = (xbin[4] & ybin[0]);
    res[5] = (xbin[5] & ybin[0]);

    res[0] ^= (xbin[5] & ybin[1]);
    res[1] ^= (xbin05  & ybin[1]);
    res[2] ^= (xbin[1] & ybin[1]);
    res[3] ^= (xbin[2] & ybin[1]);
    res[4] ^= (xbin[3] & ybin[1]);
    res[5] ^= (xbin[4] & ybin[1]);
```

⋮

Conclusions and Perspectives

Conclusions

- We presented a static analysis tool (Catalyzr^{TM}) that allows to determine whether the implementation of a cryptographic algorithm is susceptible to cache-timing side-channel leaks.
- NIST post-quantum project candidates: potential leaks in a vast majority of the candidates.
- Analysis of the severity of leakages.

Perspectives

- Formalize the analysis carried out by Catalyzr^{TM} core tool (namely Alexander Schaub's Stanalyzr)
- Extend the tool to RowHammer [14], [15]

Bibliographical references I

[1] A. Boscher, H. Handschuh, and E. Trichina, "Blinded fault resistant exponentiation revisited", in *FDTC*, Lausanne, Switzerland, IEEE Computer Society, 2009, pp. 3–9, isbn: 978-0-7695-3824-2.

[2] M. Joye and S.-M. Yen, "The montgomery powering ladder", in *CHES*, B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, Eds., ser. Lecture Notes in Computer Science, vol. 2523, Springer, 2002, pp. 291–302, isbn: 3-540-00409-2.

Bibliographical references II

[3] M. Alam, H. A. Khan, M. Dey, N. Sinha, R. L. Callan, A. G. Zajic, and M. Prvulovic, "One&done: a single-decryption em-based attack on openssl's constant-time blinded rsa", in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, W. Enck and A. P. Felt, Eds., USENIX Association, 2018, pp. 585–602. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/alam.

[4] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES", in *Cryptographers Track at the RSA Conference*, Springer, 2006, pp. 1–20.

Bibliographical references III

[5] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, L3 cache side-channel attack.", in *USENIX Security Symposium*, 2014, pp. 719–732.

[6] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical", in *Security and Privacy (SP), 2015 IEEE Symposium on*, IEEE, 2015, pp. 605–622.

Bibliographical references IV

[7] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload — a cache attack on the BLISS lattice-based signature scheme", in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, B. Gierlichs and A. Y. Poschmann, Eds., ser. Lecture Notes in Computer Science, vol. 9813, Springer, 2016, pp. 323–345, isbn: 978-3-662-53139-6. doi: 10.1007/978-3-662-53140-2_16. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53140-2_16.

Bibliographical references V

[8] Y. Yarom, D. Genkin, and N. Heninger, "Cachebleed: a timing attack on openssl constant-time RSA", *J. Cryptographic Engineering*, vol. 7, no. 2, pp. 99–112, 2017. doi: 10.1007/s13389-017-0152-y. [Online]. Available: https://doi.org/10.1007/s13389-017-0152-y.

[9] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware", *J. Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018. doi: 10.1007/s13389-016-0141-6. [Online]. Available: https://doi.org/10.1007/s13389-016-0141-6.

Bibliographical references VI

[10] A. Facon, S. Guilley, M. Lec'hvien, A. Schaub, and Y. Souissi, "Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms", in *3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2-4, 2018*, IEEE, 2018, pp. 7–12, isbn: 978-1-5386-6544-2. doi: 10.1109/IVSW.2018.8494855. [Online]. Available: https://doi.org/10.1109/IVSW.2018.8494855.

Bibliographical references VII

[11] S. Takarabt, A. Schaub, A. Facon, S. Guilley, L. Sauvage, Y. Souissi, and Y. Mathieu, "Cache-timing attacks still threaten iot devices", in *Codes, Cryptology and Information Security - Third International Conference, C2SI 2019, Rabat, Morocco, April 22-24, 2019, Proceedings - In Honor of Said El Hajji*, C. Carlet, S. Guilley, A. Nitaj, and E. M. Souidi, Eds., ser. Lecture Notes in Computer Science, vol. 11445, Springer, 2019, pp. 13–30, isbn: 978-3-030-16457-7. doi: 10.1007/978-3-030-16458-4_2. [Online]. Available: https://doi.org/10.1007/978-3-030-16458-4_2.

Bibliographical references VIII

[12] S. Carré, A. Facon, S. Guilley, S. Takarabt, A. Schaub, and Y. Souissi, "Cache-timing attack detection and prevention — application to crypto libs and PQC", in *COSADE*, ser. Lecture Notes in Computer Science, Darmstadt, Germany, vol. 11421, Springer, 2019.

[13] J. Danger, Y. E. Housni, A. Facon, C. T. Gueye, S. Guilley, S. Herbel, O. Ndiaye, E. Persichetti, and A. Schaub, "On the performance and security of multiplication in $GF(2^n)$", *Cryptography*, vol. 2, no. 3, p. 25, 2018. doi: 10.3390/cryptography2030025. [Online]. Available: https://doi.org/10.3390/cryptography2030025.

Bibliographical references IX

[14] S. Carre, M. Desjardins, A. Facon, and S. Guilley, "Openssl bellcore's protection helps fault attack", in *21st Euromicro Conference on Digital System Design, DSD 2018, Prague, Czech Republic, August 29-31, 2018*, M. Novotný, N. Konofaos, and A. Skavhaug, Eds., IEEE Computer Society, 2018, pp. 500–507, isbn: 978-1-5386-7377-5. doi: 10.1109/DSD.2018.00089. [Online]. Available: https://doi.org/10.1109/DSD.2018.00089.

Bibliographical references X

[15] F. Zhang, X. Lou, X. Zhao, S. Bhasin, W. He, R. Ding, S. Qureshi, and K. Ren, "Persistent fault analysis on block ciphers", *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 150–172, 2018. doi: 10.13154/tches.v2018.i3.150-172. [Online]. Available: https://doi.org/10.13154/tches.v2018.i3.150-172.

SECURE-IC
THE SECURITY SCIENCE COMPANY

THANKS FOR YOUR ATTENTION

CONTACT

EUROPE	sales-EU@secure-ic.com
APAC	sales-APAC@secure-ic.com
JAPAN	sales-JAPAN@secure-ic.com
AMERICAS	sales-US@secure-ic.com