**Problem**

libgcrypt before version 1.7.8 is vulnerable to a cache side-channel attack resulting into a complete break of RSA-1024 while using the left-to-right method for computing the sliding-window expansion. The same attack is believed to work on RSA-2048 with moderately more computation. This side-channel requires that attacker can run arbitrary software on the hardware where the private RSA key is used.

L’attaque est bien expliquée dans l’article : <https://eprint.iacr.org/2017/627.pdf>

* La solution fait du sens après avoir lu l’article.

<https://lists.gnupg.org/pipermail/gnupg-announce/2017q2/000408.html>

crt == Chinese remainder theorem

**rsa.c**

static void secret\_core\_crt (gcry\_mpi\_t **M**, gcry\_mpi\_t **C**, gcry\_mpi\_t **D**, unsigned int **Nlimbs**, gcry\_mpi\_t **P**, gcry\_mpi\_t **Q**, gcry\_mpi\_t **U**) {

gcry\_mpi\_t m1 = mpi\_alloc\_secure ( Nlimbs + 1 );

gcry\_mpi\_t m2 = mpi\_alloc\_secure ( Nlimbs + 1 );

gcry\_mpi\_t h = mpi\_alloc\_secure ( Nlimbs + 1 );

/\* m1 = c ^ (d mod (p-1)) mod p \*/

mpi\_sub\_ui ( h, P, 1 );

mpi\_fdiv\_r ( h, D, h );

mpi\_powm ( m1, C, h, P );

/\* m2 = c ^ (d mod (q-1)) mod q \*/

mpi\_sub\_ui ( h, Q, 1 );

mpi\_fdiv\_r ( h, D, h );

mpi\_powm ( m2, C, h, Q );

/\* h = u \* ( m2 - m1 ) mod q \*/

mpi\_sub ( h, m2, m1 );

if ( mpi\_has\_sign ( h ) )

mpi\_add ( h, h, Q );

mpi\_mulm ( h, U, h, Q );

/\* m = m1 + h \* p \*/

mpi\_mul ( h, h, P );

mpi\_add ( M, m1, h );

mpi\_free ( h );

mpi\_free ( m1 );

mpi\_free ( m2 );

}

**Fix**

<https://elixir.bootlin.com/linux/v4.5/source/lib/mpi/mpi-pow.c#L36>

* mpi\_powm(MPI res, MPI base, MPI exp, MPI mod) : RES = BASE ^ EXP mod MOD

<https://git.gnupg.org/cgi-bin/gitweb.cgi?p=libgcrypt.git;a=commitdiff;h=8725c99ffa41778f382ca97233183bcd687bb0ce;hp=78130828e9a140a9de4dafadbc844dbb64cb709a>

* **Exponent blinding is a kind of workaround to add noise. Signal (leak) is still there for non-constant-time implementation.**
  + Cette solution fait du sens, elle est utilisée pour brouiller le nombre de multiplications/exposants pour ne pas donner plus d’information sur la clé privé (d)

<https://crypto.stackexchange.com/questions/5953/what-is-blinding-used-for-in-cryptography>

Explique bien la logique de RSA blinding : À la place de **calculer**  **mod n de la** **façon classique** :

* Aveugler les données **avant** d'augmenter la puissance **dth ()**
* Choisissez un nombre aléatoire r
* Calculer s = mod n
* Calculer X = xs mod n
* Calculer Y = mod n
* Calculer y = mod n
* Ici, = = = = =  **mod n** (le résultat désiré).
* **C’est important de mentionner que dans la solution offerte, ils utilisent seulement l’exponent blinding, ceci veut dire que seulement l’exposant *d* est brouillé (dans l’exemple précédente, le message en clair à été brouiller).**

**rsa.c**

static void secret\_core\_crt (gcry\_mpi\_t **M**, gcry\_mpi\_t **C**, gcry\_mpi\_t **D**, unsigned int **Nlimbs**, gcry\_mpi\_t **P**, gcry\_mpi\_t **Q**, gcry\_mpi\_t **U**) {

gcry\_mpi\_t m1 = mpi\_alloc\_secure ( Nlimbs + 1 );

gcry\_mpi\_t m2 = mpi\_alloc\_secure ( Nlimbs + 1 );

gcry\_mpi\_t h = mpi\_alloc\_secure ( Nlimbs + 1 );

gcry\_mpi\_t D\_blind = mpi\_alloc\_secure ( Nlimbs + 1 );

**gcry\_mpi\_t r;**

**unsigned int r\_nbits;**

**r\_nbits = mpi\_get\_nbits (P) / 4;**

**if (r\_nbits < 96)**

**r\_nbits = 96;**

**r = mpi\_alloc\_secure ( (r\_nbits + BITS\_PER\_MPI\_LIMB-1)/BITS\_PER\_MPI\_LIMB );**

**/\* d\_blind = (d mod (p-1)) + (p-1) \* r \*/**

**/\* m1 = c ^ d\_blind mod p \*/**

**\_gcry\_mpi\_randomize (r, r\_nbits, GCRY\_WEAK\_RANDOM);**

**mpi\_set\_highbit (r, r\_nbits - 1);**

mpi\_sub\_ui ( h, P, 1 );

**mpi\_mul ( D\_blind, h, r );**

mpi\_fdiv\_r ( h, D, h );

**mpi\_add ( D\_blind, D\_blind, h );**

mpi\_powm ( m1, C, **D\_blind**, P );

**/\* d\_blind = (d mod (q-1)) + (q-1) \* r \*/**

**/\* m2 = c ^ d\_blind mod q \*/**

**\_gcry\_mpi\_randomize (r, r\_nbits, GCRY\_WEAK\_RANDOM);**

**mpi\_set\_highbit (r, r\_nbits - 1);**

mpi\_sub\_ui ( h, Q, 1 );

**mpi\_mul ( D\_blind, h, r );**

mpi\_fdiv\_r ( h, D, h );

**mpi\_add ( D\_blind, D\_blind, h );**

mpi\_powm ( m2, C, **D\_blind**, Q );

mpi\_free ( r ); (libérer l’espace mémoire)

mpi\_free ( D\_blind ); (libérer l’espace mémoire)

/\* h = u \* ( m2 - m1 ) mod q \*/

mpi\_sub ( h, m2, m1 );

if ( mpi\_has\_sign ( h ) )

mpi\_add ( h, h, Q );

mpi\_mulm ( h, U, h, Q );

/\* m = m1 + h \* p \*/

mpi\_mul ( h, h, P );

mpi\_add ( M, m1, h );

mpi\_free ( h );

mpi\_free ( m1 );

mpi\_free ( m2 );

}