Why you shouldn't write cryptographic algorithms yourself

Experience why writing your own crypto is harder than it seems at first.

Simo Sorce
Sr. Principal Sw. Engineer – RHEL Crypto Team
2019-01-26

Everyone tells you that you shouldn’t write your own crypto, but they don’t tell you why.


I COULD TELL YOU,
BUT I WON'T.
memegenerator.net

Instead let’s see what it takes to write software to handle a cryptographic function like RSA*

*I chose RSA only because I had to deal with it recently, could have used any Symmetric or asymmetric cryptographic primitive


Your Journey starts here...

Clear text

Public exponent

$$C = M^e \bmod N$$

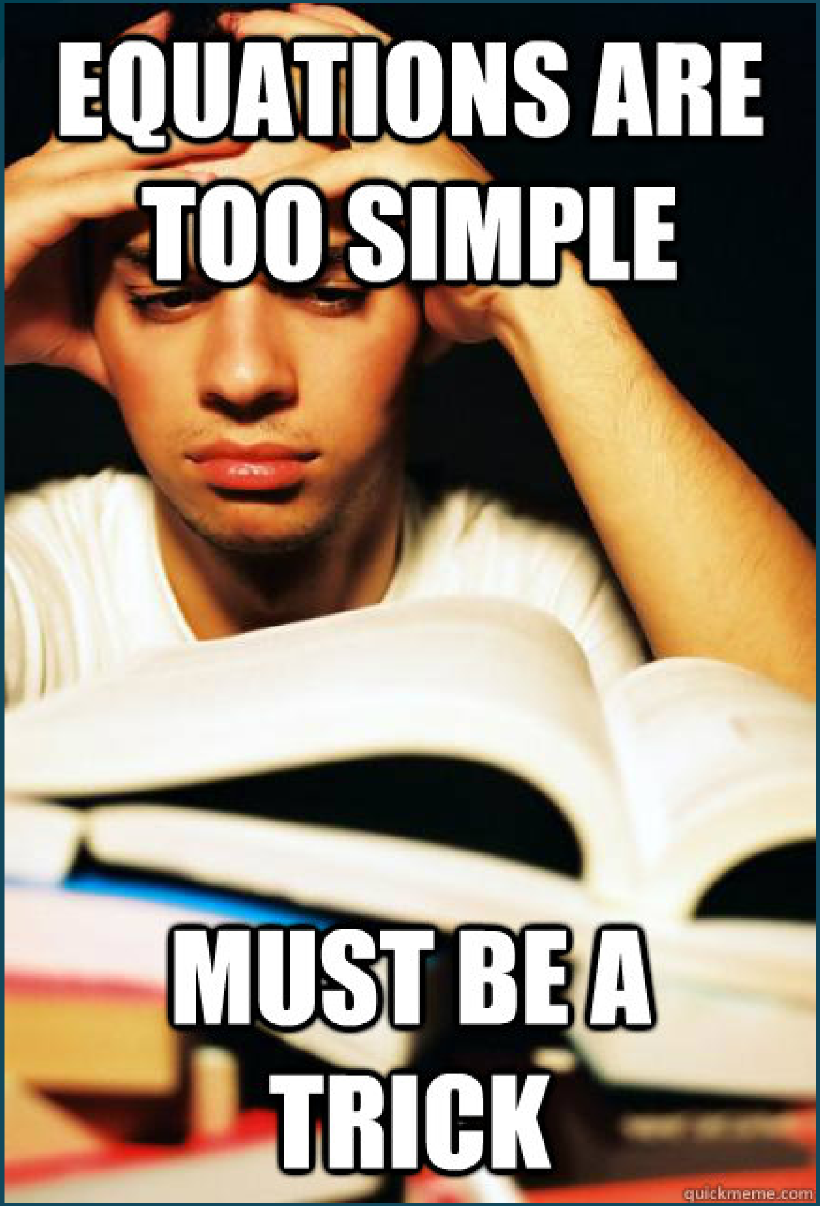
Encrypted message

Private exponent

Encrypted message

$$M = C^{d} \bmod N$$

Clear text


EQUATIONS ARE TOO SIMPLE
MUST BE A TRICK
quickmeme.com

No really, no tricks!
RSA is really simple


OR IS IT???

Let’s look at those “useless” details the cryptographers talk about from time to time!

Attacks based on poor practices

Easy stuff :-)

These attacks are based on the math, not the implementation.

- Common Modulus - (Simmons)
  - Yeah, please never reuse p, q
- Low Private Exponent (d) - (Wiener)
  - Breaks cryptosystem – hey but decryption is real fast!
- Low Public Exponent (e) - (Coppersmith, Hastad, Franklin-Reiter)
  - Not a total break, but still please use $e > 2^{16} - 1$
  - Also use randomized padding
- … for more details, search for:
  - Twenty Years of Attacks on the RSA Cryptosystem (Dan Boneh)

WE LOOKED AT THE MATH!!!

Basic tools needed to implement RSA

Usually beyond what standard languages provide

- Infinite precision math library
  - You really need to deal with **BIG** numbers, as in several thousands bits large numbers, so they won't fit in your processor registers as normal integers, or long integers or even long long integers, and you can't use floats.
- Fast, prime number generation tools to find good large primes
  - For key generation
- A good CSPRNG
  - Also for key generation and other things

RSA decryption using GMP*

Simplest code

```
/* compute root (raise to private exponent) */
mpz_powm(message, ciphertext, key->d, key->n);
```

1

This is a bit slow …

*GNU Multiple Precision Arithmetic Library


HEY, YOU DONE YET?
I REALLY GOTTA GO!!!
made on imgur

Faster RSA decryption

A bit faster using CRT

$M = C^d \bmod N$ →

$dp = d \bmod (P - 1)$ $dq = d \bmod (Q - 1)$
$Mp = C^{dp} \bmod P$ $Mq = C^{dq} \bmod Q$
Find: $M = Mp \bmod P == Mq \bmod Q$

```
/* compute root (derived from CRT) */
mpz_fdiv_r(m_mod_p, C, key->p);
mpz_powm(Mp, m_mod_p, key->a, key->p);

mpz_fdiv_r(m_mod_q, ciphertext, key->q);
mpz_powm(Mq, m_mod_q, key->b, key->q);

mpz_sub(tmp1, Mp, Mq);
mpz_mul(tmp2, tmp1, key->c);
mpz_fdiv_r(Xp, tmp2, key->p);

mpz_mul(tmp1, key->q, Xp);
mpz_add(M, tmp1, Mq);
```

x10

Attacks on implementations

Where *everyone* gets it wrong the first 42 times!

These attacks use math to defeat implementation issues.
They all need an *Oracle*, conveniently any TLS server is one.

- Timing attacks (Kocher)
  - Use blinding to defeat this (Rivest)
- Random Faults (Boneh, DeMillo, and Lipton)
  - Check signature before sending out
- Bleichenbacher's Attack on PKCS 1 (Bleichenbacher)
  - In TLS defeated by using a random session key instead of returning error

Blinding

Prevents using the server as a signing Oracle

$M = C^d \bmod N$ → $Cr = C * r^e \bmod N$
$M * r = Cr^d \bmod N$
$M = Cr^d / r \bmod N$

```
random_func(R); /* generate random R */
mpz_invert(Ri, R, key->n); /* ..and its inverse Ri */

/* blinding */
mpz_powm(tmp1, R, key->e, key->n);
mpz_mul(tmp2, tmp1, C);
mpz_fdiv_r(Cr, tmp2, key->n);

rsa_compute_root(Mr, Cr);

/* unblinding */
mpz_mul(tmp1, Mr, Ri);
mpz_fdiv_r(M, tmp1, key->n);
```

x2

Checking

Prevents sending faulty signatures

$M = C^d \bmod N$ **+** $C = M^e \bmod N$

```
/* blinding */
rsa_blind(Cr, Ri, C);

rsa_compute_root(Mr, Cr);

/* check */
mpz_powm(Cr2, Mr, key->e, key→n);
if(Cr2 != Cr) goto error;

/* unblinding */
rsa_unblind(M, Ri, Mr);
```

+2

One defense from Bleichenbacher

```
if (error) {
    random_func(M);
    return M;
}
```

+2

redhat.

ERROR: YOU ARE NOT DEPRESSED ENOUGH

Attacks based on CPU architecture

Here is were people give up! :-)

These attacks use timing and caching issues to retrieve your keys.
They all need a LOCAL *Oracle*, conveniently any TLS server on a SHARED host is one.

- The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations (Ronen, Gillham, Genkin, Shamir, Wong, Yarom)
  - Attacks the RSA implementation by timing how much time computations take
  - Attacks the RSA implementation by checking which memory area is accessed and when via CPU cache inspection and manipulation
- Funny note: OpenSSL did not raise a CVE because their threat model does not involve protecting from "local" attacks …
  - Do you run Virtual Machines or Containers ?

Attacks based on CPU architecture

Here is were people give up! :-)

These attacks are use timing and caching issues to retrieve your keys. They all need a LOCAL *Oracle*, conveniently any TLS server on a SHARED host is one.

- The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations (Ronen, Gillham, Genkin, Shamir, Wong, Yarom)
  - Attacks the RSA implementation by timing how much time computations take
  - Attacks the RSA implementation by checking which memory area is accessed and when via CPU cache inspection and manipulation
- Funny note: OpenSSL did not raise a CVE because their threat model does not involve protecting from "local" attacks ...
  - Do you run Virtual Machines or Containers ?

Defeating Cache/Timing attacks

Or at least we tried to …

Luckily some of this work was already done to solve other timing issues

- GMP needs "security" functions that compute in constant time **and** constant space
  - mpz_powm → mpn_sec_powm
  - …
- Change rsa_compute_root() to be side-channel silent
  - Remove **all input dependent** conditional operations
  - 1 function of about 10 lines → 8 functions for a total of about 100 lines
  - Obviously slower, also a lot more complicated
- Change pkcs1 (de)padding function to be side-channel silent
  - 1 function of about 20 lines → 2 functions for a total of about 40 lines
- All considered about 40 commits upstream

Example

```
memcpy(message, terminator + 1, message_length);
*length = message_length;
```

↓

```
/* fill destination buffer fully regardless of outcome. Copies the message
 * in a memory access independent way. The destination message buffer will
 * be clobbered past the message length. */
shift = padded_message_length - buflen;
cnd_memcpy(ok, message, padded_message + shift, buflen);
offset -= shift;
/* In this loop, the bits of the 'offset' variable are used as shifting
 * conditions, starting from the least significant bit. The end result is
 * that the buffer is shifted left exactly 'offset' bytes. */
for (shift = 1; shift < buflen; shift <<= 1, offset >>= 1)
  {
    /* 'ok' is both a least significant bit mask and a condition */
    cnd_memcpy(offset & ok, message, message + shift, buflen - shift);
  }

/* update length only if we succeeded, otherwise leave unchanged */
*length = (msglen & (-(size_t) ok)) + (*length & ((size_t) ok - 1));
```

x3 - x5

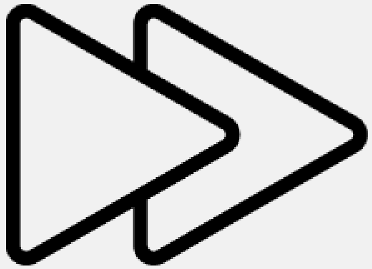
From naive to reasonably secure implementation

↓

Two orders of magnitude more code (… and bugs ?)

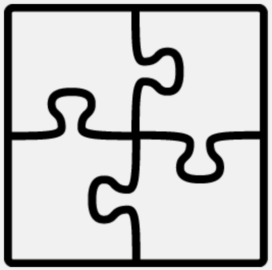
Chose ~~Two~~ One

Compromises are necessary



FAST

SECURE



SIMPLE

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos