Basic attacks against RSA with Python 3

CATALDO BASILE

< CATALDO.BASILE@ POLITO.IT >

POLITECNICO DI TORINO

Agenda

- learning objectives
  - RSA: understand and evaluate the impact on apparently minor decisions
    - on the overall security of entire cryptosystems
    - mathematical properties of asymmetric crypto allow attacks
  - mathematical properties may be used by attackers to their advantage
  - in crypto, also a single bit matters, so don't give any information outside
  - at the mathematical level, real-world attacks are becoming much more complex
  - remember that implementation can be the weakest link in crypto

- recall of some useful number theory functions
- RSA attacks
  - general factorization
  - fermat's factorization
  - common prime
  - common moduli
  - low public exponent
  - Hastad's broadcast
  - LSB Oracle
  - low private exponent
    - Wiener
    - Boneh Durfee
  - Coppermith's short pad
    - Franklin Reiter
  - implementation attacks

Number theory

Bézout identity

- given two integer numbers n_1, n_2
- then you can find two integer numbers u and v such that
  - $u*n_1 + v*n_2 = gcd(n_1,n_2)$
- NOTE: u and v can be either positive or negative numbers

it's even more useful when…

- if two numbers a and m are coprime
  - $gcd(a,m) = 1$
- then you can find two numbers
  - $ua + vm = 1$ → (working modulo m) → $ua + vm = 1 = ua \pmod m$
- in these cases, you can compute inverses easily with the Bézout identity
  - u is the inverse of $a \bmod m$

Extended Euclidean algorithm

an algorithm that computes

- the *gcd* and at the same time
- outputs the Bézout coefficients

- *egcd(n1,n2)* →
  - *g* → *gcd(n1,n2)*
  - *u*
  - *v*
- such that
  - $u*n1 + v*n2 = g$

Integer roots (modulo m)

k^{th} root of a number x mod m is the number r such that

- $x = r^k \pmod{m}$

ad hoc implementations are needed

- don't use *pow*
  - it works for floating-point numbers!
- some efficient implementations are available based on Newton's methods
  - …already available in the libraries, web, etc.
- examples
  - *isqrt*
  - *iroot*

Evaluating attacks against RSA

factorization of the modulus
- obtain the primes p and q → then compute the private exponent d
- you can decrypt all the past and future messages and generate signatures

obtain the private exponent d (RSA problem)
- you can decrypt all the past and future messages and generate signatures
- happens if primes and exponents are not chosen properly
  - become vulnerable as specific theorems apply

decrypt selected messages
- happens if primes, public or private exponents are not selected properly, or
- if numbers are not padded properly before computing data
- too many data are encrypted with the same keys or encrypted data have specific mathematical properties
- weaker than the past two cases

some considerations on the performance
- every attack has a specific cost that needs to be considered before starting
- e.g., factorization easily becomes impractical

Factorize the modulus

- if you obtain *p* and *q* from *n*
  - you can compute *phi(n)* then you compute *d* from *e*
    - you win!
  - factorization is a more powerful tool
    - if you can factorize n → you solve the RSA problem
      - but not vice versa

- naive method
  - factorDB: https://github.com/ryosan-470/factordb-pycli
    - for small numbers + stored known decompositions

- General Number Field Sieve algorithm: needs very optimized code
    - yafu: https://github.com/DarkenCode/yafu/blob/master/docfile.txt

Fermat's factorization

works if *p* and *q* are close numbers

- $p = a + b, q = a - b$
  - for some integer *a*, *b* with *b* small
- the modulus becomes

$$n = pq = (a+b)(a - b) = a^2 - b^2$$

fermat's factorization algorithm is based on an approximation cycle

- starts from the midpoint
  - the integer a = b = *sqrt(n)*
- increases *a* at each step and recomputes *b*
- stops when finds *a* and *b* such that *b*

Common prime (factorization)

if n_1,n_2 have a common prime

- …it's too easy!!!

$n_1 = p_1 * p_2$

$n_2 = p_1 * p_3$

find the gcd with the Euclidean algorithm

- it's very fast!
- $gcd(n1,n2) = p1$

- $p2 = n1 // p1$
- $p3 = n2 // p3$

Common modulus (messages)

two public keys

- $k_1 = (e_1,n)$ and $k_2 = (e_2,n)$

encrypt the message m twice with different keys, e.g., with k_1 and k_2

- $c_1 = m^{e1} \mod n$
- $c_2 = m^{e2} \mod n$

use Bézout to find

- $e_1 * u + e_2 * v = gcd(e_1,e_2)$

if $(e_1,e_2) = 1$ we can compute

- $c_1^u c_2^v = m^{e1 * u + e2 * v} = m \pmod n$

decrypt all the messages that are encrypted with the keys with the same modulus

Low public exponent (message)

if $e = 3$ and the message m to encrypt is small

- that is if $c = m^e < n$
- $m*m*m < n$

when computing c we don't have overflow

- no reminders are computed

than m it's just the integer cubic root of the ciphertext c

decrypt all the "small messages" encrypted with keys using 3 as public exponent

Hastad's broadcast (messages)

if the same message *m* is encrypted with *e* different public keys

- $k_i = (e, n_i)$
- $c_i = m^e \bmod n_i$
- $gcd(n_i, n_j) = 1$ for all *i,j* *(no common primes)*

the CRT ensures that we can find a number

- $c = m^e \bmod N$
- where $N = n_1 * n_2 * \ldots * n_e$
- in the field of size N the message m is a "small number" thus the other attack works

effective when $e = 3$

- reasonable with *17*
- almost impossible with *65537*
  - difficult finding a message encrypted with 65537 distinct keys

decrypt all the messages (regardless of the size) if they are encrypted for enough recipients

Basic solution…

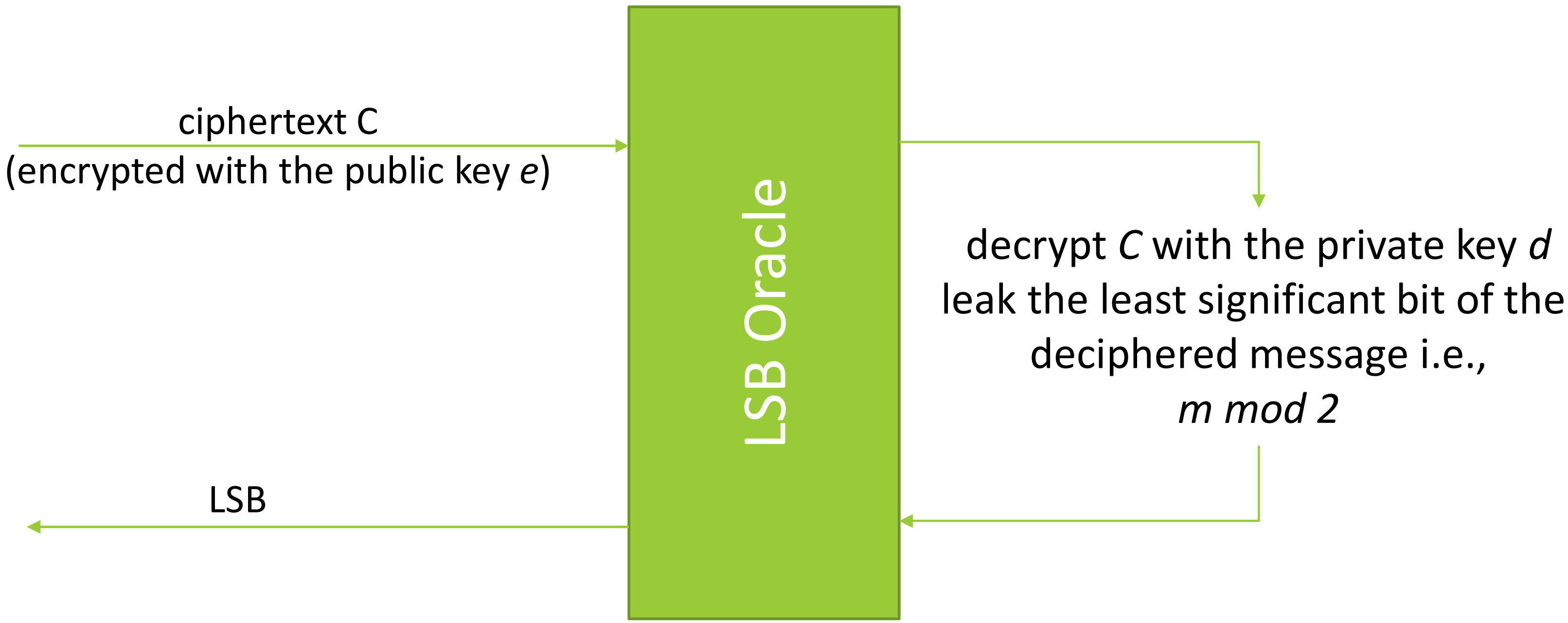
for all $i < e$

- compute $N_i = N \,//\, n_i$
- find Bézout coefficients u_i, v_i
  - $N_i * u_i + n_i * v_i = 1$

- a solution is
  - $c = c_1 * u_i * N_i + c_2 * u_2 * N_2 + \ldots + c_e * u_e * N_e$

  - there are better implementations though…

LSB Oracle attack (messages)



Target = decipher the original message

LSB Oracle attack

$c = m^e \bmod n$ and $m < n$

we send $c' = 2^e c = 2^e m^e = (2m)^e$

- if $2m < n$ the LSB is 0 [2m is a left shift e there is no overflow]
  - therefore $m < n/2$
  - m is in [0,n/2]
- if $2m > n$ the LSB is 1 [2m is a left shift, overflow]
  - there is an overflow
  - $2m \pmod n = 2m - n \pmod n$
    - n is odd → $2m - n$ is odd
    - m > n/2 → m is in [n/2,n]

send $4^e c$, $8^e c$, …, $2^{e*nbit} c$ and do the same interval shrinking

- n bit is the size (in bit) of the modulus
- log (n) requests to the oracle

Low exponent attacks (private exponent)

if the private exponent is “low”

- it can be recovered efficiently starting from the public key *(e,n)*

$n = pq$ and $p < q < 2p$ (have the same size in bits, approximately)

d is considered low when

- $n^{1/4} = n^{0.25}$
  - if we use the Wiener’s attack
- $n^{0.292}$
  - if we use the Boneh-Durfee attack

use approximations based on advanced number theory results

- continuous fractions / lattices

Low exponent attacks

several implementations available

- pick the best for your purposes

- Wiener's attack implementations available both in python3 and Sage
  - https://github.com/pablocelayes/rsa-wiener-attack/blob/master/RSAwienerHacker.py

- Boneh-Durfee usually requires Sage
  - https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/boneh_durfee.sage
  - https://latticehacks.cr.yp.to/rsa.html

Coppersmith's short pad (message)

given a message m, padded with two random values r_1 and r_2 composed of at most *p* bits

- $m_1 = m \,||\, r_1$ and $m_2 = m \,||\, r_2$

and encrypted as

- $c_1 = m_1{}^e \bmod n$
- $c_2 = m_2{}^e \bmod n$

an attacker accessing the public key *(e,n)* used to encrypt c_1 and c_2

- can efficiently recover m
- if $p < \lfloor n/e^2 \rfloor$ → *floor(n / e²)*

based on the Franklin-Reiter attack on related messages

- $m_1 = f(m_2)$ and f is a linear function and c_1, c_2 encryptions with the same key
- efficiently recover m_1 and m_2 from c_1 and c_2

Other tools for attacking RSA (for lazy people)

RsaCtfTool

- https://github.com/Ganapati/RsaCtfTool

Cryptools

- https://github.com/sonickun/cryptools

RSHack

- https://github.com/zweisamkeit/RSHack

Implementation attacks

side channels

- timing attacks
  - guess the values of bits from the time needed to perform some operations
- power consumption
  - observes consumption diagrams and guesses bits

fault attacks

- force an error in an implementation of RSA decryption using the CRT
  - and recover one of the primes
    - the one not affected by the error

Further reading

An interesting old paper

- http://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf

Decimal class in Python

- https://docs.python.org/3/library/decimal.html