15 ways to break RSA security

Renaud Lifchitz

Econocom Digital Security

OPCDE, April 26-27, 2017

Speaker's bio

- French senior security engineer working at Econocom Digital Security (`http://www.digitalsecurity.fr`), France
- Main activities:
  - Penetration testing & security audits
  - Security research
  - Security trainings
- Main interests:
  - Security of protocols (authentication, cryptography, information leakage, zero-knowledge proofs...)
  - Number theory (integer factorization, primality testing, elliptic curves...)

Goals of this talk

- We will do a research state of the art talk presenting as many as possible ways to attack RSA algorithm (encryption and signature cryptosystems), some of them being very new (discovered or implemented in the last few years). We will also show real computing demos with simple tools.
- The goal is NOT to explain all the math behind!

Outline

Section 1

Introduction

RSA basics

- N: public key
- p, q: private factors of $N = p.q$
- $\varphi(n)$: Euler's totient function, here $\varphi(n) = (p-1).(q-1)$
- e: public encryption or signature exponent
- d: private encryption or signature exponent
- $e.d \equiv 1 \mod \varphi(N)$ relatioship between public and private exponent
- $M \equiv m^e \mod N$: encrypted message
- $m \equiv M^d \mod N$: decrypted message
- Finding d, $\varphi(n)$ or p is enough to crack RSA security

Tools

My favorite tools for factorization:

- For simple usage, lazy user and intermediate attacks:
  Yafu (`https://sourceforge.net/projects/yafu/`) or
  msieve (`http://sourceforge.net/projects/msieve/`)
- For customized and advanced attacks:
  Sage (with Python syntax, `http://www.sagemath.org/`) or
  PARI/GP (`https://pari.math.u-bordeaux.fr/`)
- For breaking real RSA records: cado-nfs
  (`http://cado-nfs.gforge.inria.fr/`)

Most given examples in next section will be in Sage

For the lazy user: Yafu HOWTO

```
$ echo ’factor(16957027507291876743797870168005371672267271571508185
3862979036573938375751224081570779437003774732030530132941443)’ | ./yafu

fac: factoring 16957027507291876743797870168005371672267271571508185
3862979036573938375751224081570779437003774732030530132941443
fac: using pretesting plan: normal
fac: using tune info for qs/gnfs crossover
div: primes less than 10000
rho: xˆ2 + 3, starting 1000 iterations on C109
(...)
Total factoring time = 107.3119 seconds

***factors found***

P2 = 11
P2 = 17
P3 = 113
P30 = 503856994217382611232027920567
P80 = 15926574707756334599680276082956571757039413458914170123113633
917363409534379359

ans = 1
```

Section 2

15 attacks

1. Small factors

- Most trivial attack
- Let $N = a.b$ with $a \leq b$ then $a^2 \leq a.b$ so $a \leq \sqrt{N}$
- If a was composite then it would have a smaller prime factor so a can be chosen prime
- Trial factorization using small precomputed primes
- Efficient when N has small factors. It was the case with Taiwan's digital ID cards!

1. Small factors

```
# Create a RSA key
p=37; n=p*random_prime(2**1019,lbound=p+1); print "N=",n
# Break it
for a in primes(10000):
    if n%a==0:
        print "-> a=",a ; break
```

```
N= 4393582188525488617788031628163518633443203266177227780003833934646
6484064011536195521165949635870085006169372626461062616950553770462541
1237157162964375124357920056450286315338840698240784985295529022490951
0972533378967928598397011304939772589329120709153783136694400996822907
56525412135629672549080290587423
-> a= 37
```

2. Fermat factorization

- Let $N = a.b$ and write $a = c + d$ and $b = c - d$, then $N = (c+d).(c-d) = c^2 - d^2$
- Try to find a perfect square $c^2 - N$ using ascending values of c
- Efficient when a and b can be chosen close ($\frac{a}{b} \approx 1$), even when they are very large!

2. Fermat factorization

```
# Create a RSA key
p=random_prime(2**512); q=next_prime(p+2**70); n=p*q; print "N=",n
# Break it
c=isqrt(n)
while c<=n:
    d2 = c*c - n
    if is_square(d2):
        d = isqrt(d2)
        print "-> a=", c-d; print "-> b=", c+d; break
    c+=1
```

```
N= 52390446205328918152014676644149972966068289279169299202731127762 3
6896486652753086419258480635681425702379535902010324066673933495743 85
```

3. Batch GCD

- The idea is to have a lot of RSA public keys and compute GCD two by two to find shared factors
- Useful for cloned systems, VMs and embedded devices with low entropy
- Cryptosense has a nice Batch-GCD key tester :
  `https://keytester.cryptosense.com/`
  and has already found tens of thousands vulnerable devices connected on the Internet (SSL/TLS/SSH certificates...) :
  `https://cryptosense.com/rsa-keytester-upgrade-18-750-new-factored-keys/`

4. Elliptic Curve Method (ECM)

- Computation with elliptic curves (interesting math groups)
- Efficient when factors are small ($<$ 60 digits) even within a very large integer

4. Elliptic Curve Method (ECM)

```
# Create a RSA key
p=random_prime(10**25); q=random_prime(2**949); n=p*q; print "N=",n
# Break it
ecm.find_factor(n, factor_digits=25)
```

```
N= 29306579011122661998157485710678849808566157431824183043218001672
6
219371928598548037938085403283881951063212946421658379198893083889816
691985905690937110257594817956121736433902000509922654445020159462384
592425882130435059452716768547760938020422775410790672931409264539610
48604365597934083175216791101396066699

[8849112930409594333974811,
 33118098098185490627311512481491948961225827846340169415630226121953
 06256503137319133971139238497813530149299899951290138164552024490805
 60857992494647552945374544849522517786646411192157378905208251584546
 19852353810452764526852011500214593593066385853958910319589027167852
 8643547519409]
```

5. Weak entropy

- A lot of embedded devices have very low entropy sources (network devices, routers, smart TVs, IoT devices, ...)
- It is quite easy to find keys bruteforcing bit patterns in factors like `0xAAAAAAAA` or `0xFFFFFFFF`

6. Smooth p-1 or p+1

- If $p-1$ or $q-1$ have only small factors we can crack the RSA key using Pollard's $p-1$ algorithm
- Similarly, if $p+1$ or $q+1$ have only small factors we can crack it using William's $p+1$ algorithm

7. Fault injection

- Computing RSA encryption (or signature) $M \equiv m^e \mod N$ can be expensive on embedded devices or smartcards
- Sometimes, this computation is splitted: $M \equiv m^e \mod p$ and $M \equiv m^e \mod q$ (which are smaller, more than two times faster), then combined $\mod N$ using the CRT (Chinese Remainder Theorem)
- If one (or more) error (i.e. bit flip) occurs in one of these computations, we can break the key, wherever the error occurs
- We can manually introduce errors during the computation for example using a heater or even... a hammer!

7. Fault injection

```
# Create a RSA key
p=random_prime(2**256); q=random_prime(2**256); n=p*q; phi=(p-1)*(q-1)
print "N=",n; e,d=None,None
for e2 in xrange(101,10000,2):
    if gcd(e2,phi)==1:
        e=e2; break
d=int(1/Mod(e,phi)); msg=randint(1,n); print "e=",e,"M=",msg
m1=power_mod(msg,d,p); m2=power_mod(msg,d,q)
m2err=m2^^(2^randint(1,255)) # Introduce a random error
s=crt([m1,m2err],[p,q]); print "S=",s
# Break it
g=int(Mod(power_mod(s,e,n)-msg,n)); print "-> ",gcd(g,n)
```

```
N= 89529023715373496355664047560521089352277512520117095001880186417605668612440063270967751388131561674818274055594002480931221095224788573028289916232567 21
```

8. Small private exponent

- Wiener's attack: as $e.d \equiv 1 \mod \varphi(N)$ with quotient k, we will try to find $\varphi(N)$ using the continued fractions expansion of $\frac{e}{N}$, which will hopefully approximate sufficiently well $\frac{k}{d}$
- Always works when $d < \frac{N^{\frac{1}{4}}}{3}$

8. Small private exponent

```
# Create a RSA key
p=1999; q=2357; n=p*q; phi=(p-1)*(q-1); d=None;
for d2 in xrange(int(n^0.25/3),2,-1):
    if gcd(d2,phi)==1:
        d=d2; break
e=int(1/Mod(d,phi))
print "N=",n,"e=",e,"d=",d
# Break it
for f in continued_fraction(e/n).convergents():
    k,d = f.numerator(), f.denominator()
    if k:
        phi2 = int((e*d-1)/k)
        a,b,c=1,-(n-phi2+1),n
        delta = b*b-4*a*c
        if is_square(delta):
            p,q = (-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a)
            print "-> p=",p," q=",q
```

```
N= 4711643
e= 4345189
d= 13
-> p= 1999  q= 2357
```

9. Known partial bits

- If the attacker guesses or recovers partial bits from p, q, e or d he can sometimes crack the key
- For example, Coppersmith's attack (finding small solutions of a polynomial modulo an unknown integer) is used when attacker knows Most Significant Bits (MSB)

9. Known partial bits

```
# Create a RSA key
p,q = random_prime(2**512), random_prime(2**512);
p,q = max(p,q),min(p,q); n=p*q; print "n=",n
# Create a hint
k=ZZ.random_element(1,10**10); noise=ZZ.random_element(1,2**150)
hint=k*p+noise; print "hint=",hint
# Break it
x=PolynomialRing(Zmod(n),"x").gen(); f=x+hint; sr=f.small_roots(beta=0.5)
if sr: kp=hint+sr[0]; print "-> factor found!:",gcd(n,kp)
else: print "-> fail!"
```

```
n= 333182763825465558657385132807288998347218840755458697639593246244
802269934195824283809118248327658955659009780897843446684486987389474
146413008960682360558538285038847855917210243290376330522747074100241
495396222376475247568676214391893273699362463455741937827950801152756
9475065755675667024259451949694987
hint= 337296722241326056205102081319898813637907965735912476103238461
277401069123208806799362178770099536783324570516766172272476705155351
22157082084182626529722524797516
-> factor found!: 358894923829689412070665221668987340331932123108537
713493023389981820368721373539609777353093134476441505343298294215777
2486149714477708413040551805670653
```

10. p/q near a small fraction

- If $\frac{p}{q} \approx \frac{a}{b}$ with small a and b, we can try to guess an approximation of the ratio and then to approximate p.
- If the approximation is good enough, MSB of p will be correct and we are able to crack N

10. p/q near a small fraction

```
n=2078582687184552768312009126849809848245785881902041974724035313386284O\
6400110488622194176904033713524423232229185097795372252163472504321674334\
4450229118356803894825212236777879489873231087939452032327369429443965278\
9130232447187550860745609455640839131604119449281274242099137735781316722\
78028283104325090O1
# Break a 1024-bit RSA in seconds!
depth=50; t=len(bin(n).replace('0b','')); nn = RealField(2000)(n)
x = PolynomialRing(Zmod(n),"x").gen()
for den in xrange(2,depth+1):
    for num in xrange(1,den):
        if gcd(num,den)==1:
            r=Integer(den)/Integer(num); phint = int(sqrt(nn*r))
            f = x - phint; sr = f.small_roots(beta=0.5)
            if len(sr)>0:
                p = int(phint - sr[0])
                if n%p==0:
                    print "-> found r =", 1/r," =>   p =",p; break
```

```
-> found r = 32/37  =>   p = 1550277791899612789638246640417550958489
8016730059954842992867919303289659772538691927167157255482088266714961
02812449629965292700131322150056390666328515 9
```

11. Shared bits

- Let $N_1 = p_1.q_1$ and $N_2 = p_2.q_2$ two different RSA keys
- Imagine p_1 and p_2 share sufficiently enough MSB
- Without knowing any of them, you can **break both** RSA keys!
  This is called "implicit factoring"

- Generalization: if there exists $a_1 < p_2$ and $a_2 < p_1$ such that $|a_1.p_1 - a_2.p_2| < \frac{p_1}{2.a_2.q_1.q_2}$
  (Abderrahmane Nitaj & Muhammad Rezal Kamel Ariffin, *Implicit factorization of unbalanced RSA moduli*, 2014)

11. Shared bits

```
n1 = 63431782986412625310912155582547071972279848634479
n2 = 99460066570677101780275829030592866099143542223

for f in continued_fraction(n2/n1).convergents():
    a,b = f.numerator(), f.denominator()
    q1 = gcd(n1,b)
    if 1<q1<n1:
        p1=n1/q1; q2=gcd(n2,a); p2=n2/q2;
        print "-> p1=",p1,"q1=",q1; print "-> p2=",p2,"q2=",q2
        break
```

```
-> p1= 29846034747067203786403150576377329237 q1= 2125300178867
-> p2= 1043487920228935667940393294165327383 q2= 9531501481
```

12. Weaknesses in signatures

A lot of implementations flaws exists :

- Lack of or bad padding before encryption/signature
- Encrypting the same message with two different keys (or using related messages)
- Signing chosen messages by the attacker
- Signing a lot of messages

13. Side channel attacks

The computation may leak information from private key by monitoring :

- Power consumption
- Emanations (TEMPEST)
- or any other varying parameter

14. Number Field Sieve (NFS)

- Generalization of the Quadratic Sieve (finding $x^2 - y^2 = N$)
- Very complex but very parallel
- This algorithm is best known against strong RSA (world records)

15. Shor quantum algorithm

- Quantum algorithm for integer factorization that runs in polynomial time formulated in 1994
- Complexity: $O((\log N)^3)$ operations and storage place
- Probabilistic algorithm that basically finds the period of the sequence $a^k \mod N$ and non-trivial square roots of unity $\mod N$
- Uses QFT (Quantum Fourier Transform)
- Some steps are performed on a classical computer
- Will probably kill RSA in 20-25 years

Section 3

Conclusion

Results & challenges

- RSA is theoretically pretty safe but there exists a lot of implementation flaws
- Recently, a lot of ways to break RSA security have been found dut to the sole choice of prime factors
- Most recent attacks are based on a combination of continued fractions expansions and Coppersmith's/LLL attacks
- Those modern attacks all show that for a given RSA size of b bits, there exists at least $2^{b/2}$ non-trivial weak keys that are hard to detect during creation
- That's a lot, but fortunately, that's not that big... ☺

Bibliography

- Neal Koblitz, *A course in number theory and cryptography*, Second Edition, Springer, 1994.
- Richard Crandall & Carl B. Pomerance, *Prime Numbers: A Computational Perspective*, Second Edition, Springer, 2005.

Thanks for your attention!

Any questions?

✉ renaud.lifchitz@digitalsecurity.fr