How to compute RSA keys?

The *Art* of RSA: Past, Present, Future

Robert Erra & Christophe Grenier
aka the EG Group

ESIEA - Pôle SI&S
9 rue Vésale, 75 005 Paris, France
{erra,grenier}@esiea.fr

*i*AWACS'09

Current section

— RSA is without doubt the most famous asymmetric cryptosystem.

In the building 10 of MIT, one can read . . .

Ronald Rivest, Adi Shamir and Leonard Adelman invented the first workable public key cryptographic system, based on the use of very large prime numbers, that has so far been proved unbreakable.

But:

- Technically, the (red) sentence is unfortunately *false*!
- There is no proof of "unbreakability" of RSA.

— But how does someone compute his RSA key practically ?

- Generally, one simply "asks" to a software or to a Trusted Third Party (TTP) or Certification Authority (CA) a RSA key by specifying (for example) the binary length.
- And you trust the results ... but this can be dangerous (OpenSSL/Debian flaw, RSA trapdoors)
- So, when you compute RSA keys, how can you be sure that they are "secure" (whatever it means)?
- In fact, *hélas*, today you cannot be sure!
- NSA (2006,[NSA09]): "During the transition to the use of elliptic curtve cryptography, RSA can be used with a 2048-bit modulus to protect classified information up to the SECRET level" !

The classical *balanced* RSA Algorithm

- p and q: two prime numbers of equal length
- $N = p\,q$: the RSA modulus they define
- e: Public exponent, prime with $\varphi(N) = (p-1)(q-1)$
- d: the private exponent

RSA equation:

$$e * d - k * \varphi(N) = 1 \tag{1}$$

Modular RSA equations:

$$e * d = 1 \bmod \varphi(N) \tag{2}$$

$$-k\varphi(N) = 1 \bmod e \tag{3}$$

Unsecure « scholar » *mathematical* RSA *(textbook RSA)*.

Algorithm 1 : RSA key Generation

Input: – an integer k > 0;

Output: – (N, e, d) with N a k bits number and, if needed, (p, q);

Begin:

Compute randomly a prime p of $k/2$ bits;

Compute randomly a prime q of $k/2$ bits;

Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;

Compute (or choose) an integer e with $GCD(e, \varphi(n)) = 1$;

Compute $d = e^{-1} \bmod \varphi(N)$; /* Sometimes $\bmod \lambda(N)$; */

Return (N, e, d) and, if needed, (p, q);

End.

RSA use (without padding)

Ciphering of $m \in \mathbb{Z}_N^*$

$c(m) = m^e \bmod N$

Signature of $m \in \mathbb{Z}_N^*$

$s(m) = m^d \bmod N$.

Duality of e and d, public and private exponents:

1. A fast signature requires a small private exponent d, chosen first and then e is computed
2. A fast encryption requires a small public exponent e, chosen first and then d is computed

— How can an user obtain RSA keys ?

- An user U can compute by himself his RSA keys to sign or to encipher.
- U can obtain RSA keys from a TTP/CA:
  1. the RSA key can be computed by the TTP/CA (alone) and then provided to U;
  2. the RSA key can be computed cooperatively by the TTP/CA and U (shared RSA key).

— How to compute a RSA key in practice ?

- how to compute the primes p and q ?
- when to choose e ? (before or after the choice of p and q ?)
- how to choose/compute d ?
- ...

— Open Problem

What about RSA Public Key Validation ?

— The problem of RSA Public Key Validation 1/2

NIST recommendations, issued in 2006 [NIS06b], contain the following definition:

Definition

Assurance of the public key validity: assurance of the arithmetic validity of the public key.

— The problem of RSA Public Key Validation 2/2

NIST recommendations [NIS06b]

- recommendation for full public key validation for DSA and ECDSA but emphasizes that "*. . . at present, there is no method defined for full public key validation for RSA; however a method for partial public key validation is specified in section 5.3.3 this is to be used until an approved method for full validation is available*".
- "*Plausability tests can detect unintentional errors with a reasonable probability*".

— Some questions about RSA Public Key validation

- Is $|p - q| = \mathcal{O}(\sqrt{N})$?
- Are $p \pm 1$ or $q \pm 1$ not smooth ?
- Is it better for p and q to be "strong primes" (Silverman)?
- Have e and d « good » security (whatever it means) ?
- Is there no trapdoor in your RSA key (Anderson, Young & Yung, Crépeau & Slakmon) ?
- Are your primes really primes ? (Probable or Provable Primes?)
- Is your RSA modulus hardly factorizable ?
- ...

Current section

— So, what can we do ?

No Full Public Key Validation for RSA (FPKV)
but, obtain the best Partial Public Key Validation for RSA (PPKV)

— Things we can do:

1. Make a partial (-> full) review of some vulner./attacks;
2. Understand the consequences from the user point of view;
3. Browse the available *Recommendations* from national agencies (NSA, NIST, FIPS, DSCCI, BSI, . . .);
4. Compare *Recommendations* and *Implementations* in open source software (OpenSSL, PolarSSL, GnuPG, libgcrypt, . . .)

Of course, today, we will not try to be exhaustive (*wip*).

— *A priori* versus *a posteriori* Public Key Validation

We propose to distinguish:

a *priori* requirement for PKV

A mathematical assurance: when buying/obtaining/computing a RSA key, it is secure according to the state of art.

a *posteriori* requirement for PKV

A legal assurance: when a RSA key has been broken, the owner is able to prove to a judge that it is « not his fault ».

For both cases, we need a Certificate of Public Key Validation (see later).

Current section

1. Introduction
2. So, what can we do ?
3. How to choose/compute e ?
4. How to choose d ?
5. How to choose the key length ?
6. Conclusion and future work

— A few questions we need to answer before obtaining e

1. Is it better to *choose* e or d ? (duality)
2. Is it better to choose first e or N ? (temporality)
3. Are all e a good choice ? A small or a large one ?
4. If computed, are all e obtained acceptable ?
5. If a non-acceptable e is obtained, where to start again ?

— Limitations in OpenSSL

The public exponent e has the type **int** and so $e \leq 2^{31} - 1$

— From RFC 3110 in 2001 [IET01]:

- A public exponent of 3 is the fastest for signature verification,
- Very weak if used for different recipients (Broadcast attack, Hastad),
- Acceptable for authentification in DNSSEC;
- A conservative choice is $e = 65537$.

— Attacks against the Public Exponent e :

- Cycling attack (Norris & Simmons)
  1. given $C = M^e \bmod N$
  2. compute $C_i = C^{e^i} \bmod N$ until we find $C_r = C^{e^r} = C \bmod N$
  3. then $M = C^{e^{r-1}} \bmod N$
- Common modulus attack (Simmons): we can find M if
  1. we are given $\{C_1 = M_1^e \bmod N, C_2 = M_1^e \bmod N\}$ and N
  2. $\gcd(e_1, e_2) = 1$
- Broadcast attack (Hastad): we can find M if
  1. we are given $\{C_i = M^e \bmod N_i\}_{i=1}^{f}$
  2. $M^f < N_1 N_2 \cdots N_f$
- Small public exponent attack (based on Lattices Attacks, see [Yan08] for a review.)

— Some Recommendations from some National Agencies

- NIST [NIS06b] in 2006: quite nothing about e, it is only recommanded that e has to be odd.
- DCSSI[a] [DCS06] in 2006: recommands to use public exponent strictly superior to $2^{16} = 65536$.
- FIPS [FIP09] in june 2009): it is proposed (page 52) to select e *prior* to generating the primes p and q, and that the exponent e **shall**[b] be an odd positive integer such that:

$$2^{16} < e < 2^{256}. \tag{4}$$

[a] Now *ANSSI*.

[b] From [FIP09]: *Shall : Used to indicate a requirement of this Standard.*

— Algorithm used by GnuPG v1.2.3 to compute e *after* the computation of p and q.

Algorithm 2 : Computation of e

...

If $\varphi(N) \neq 0 \bmod [41]$ **Then** $e = 41$;

Else If $\varphi(N) \neq 0 \bmod [257]$ **Then** $e = 257$;

Else

$e = 65537$;

While $GCD(e, \varphi(N)) \neq 1 : e = e + 2$;

— Analysis of GnuPG v1.2.3

Nguyen [Ngu] has shown that algorithm (2) creates a minor flaw:

- if $e \geq 65539$ then $\varphi(N) = 0 \bmod [41 * 257 * 65537]$,
- and since $\varphi(N) = 0 \bmod [4]$
- we obtain a 32 bits factor of $\varphi(N)$!

This is not a serious threat because the probability to have $e \geq 65539$ is low ($< 0.2\%$) and the obtained knowledge of 32 bits of $\varphi(N)$ is not enough to be used in known efficient factorization algorithm. But, this can be useful for example

- to improve the Wiener attacks
- to improve some partial key exposure attacks.

— RSA in GnuPG v1.4.10: $e \geq 65537$

Algorithm 3 : RSA key generation
Input: — an integer k > 0;
Output: — (N, e, d) with N a k bit number
Begin:
$e = 65537$;
While $bitSize(N) \neq k$
Compute randomly a prime p of $k/2$ bits;
Compute randomly a prime q of $k/2$ bits;
Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;
While $GCD(e, \varphi(N)) \neq 1$ $e = e + 2$;
/* So, again, if $e > 65537$, we gain information about φ(N) */
Compute $d = e^{-1} \bmod \varphi(N)$;
End.

— RSA in OpenSSL 0.9.8k

Algorithm 4 : RSA key generation

Input: — an integer k ;

Output: — (N, e, d, d_p, d_q) with N a k bit number

Begin:

$e = 65537$; /* e is fixed, p and q are recomputable */

While $gcd(e, p-1) \neq 1$ Compute randomly a prime p of $k/2$ bits;

While $gcd(e, q-1) \neq 1$ Compute randomly a prime q of $k/2$ bits;

Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;

Compute $d = e^{-1} \bmod \varphi(N)$;

Compute $d_p = d \bmod (p-1)$;

Compute $d_q = d \bmod (q-1)$;

End.

— RSA in libgcrypt 1.4.4: $e \geq 65537$

Algorithm 5 : RSA key generation (it follows ANS X9.31)

Input: — an integer $k = 1024 + 256s > 0$;

Output: — (N, e, d) with N a k bit number

Begin:

$e = 65537$;

Compute randomly a prime p of $k/2$ bits;

Compute randomly a prime q of $k/2$ bits;

Compute $N = p\,q$ and $\varphi(N) = (p-1)(q-1)$;

Compute $\lambda(N) = lcm(p-1, q-1) = \varphi(N)/gcd(p-1, q-1)$

While $GCD(e, \lambda(N)) \neq 1$ $e = e + 2$;

/* So, again, if $e > 65537$, we gain information about $\lambda(N)$ */

Compute $d = e^{-1} \bmod f$;

End.

Current section

— What about the private exponent d ?

The private exponent d is computed generally after e (duality). The main point is to avoid small d.

- Wiener attack (1990,[Wie90]): $d < 1/4N^{1/3}$ is a *sufficient* condition to find the private exponent d in a time in the order of $O(\log N)$.
- Boneh and Durfee (2000,[BG00]): if $d < N^{0.292}$ then we can recover it with the help of the Coppersmith's method [Cop96b, Cop96a, Cop97] to solve modular equations (heuristic but it works!).
- Conjecture [BG00]: if $d < \sqrt{N}$ then RSA is insecure.
- It is recommanded in [DCS06] to use private exponents of the same length as the RSA modulus.

— A (new) provable variant

Theorem

If $k = k_1\, k_2$, where k_1 is prime or not then, given N, e and a bound K such that $k_1 \leq K$, if

$$d < \frac{1}{2}\sqrt{k_1}\, N^{1/4}, \tag{5}$$

we can efficiently recover d using the continued fraction expansion (CFE) of e/N in a time linear in $\log N$ and k_1.

Proof roughly the same as the proof of Wiener attack:

$$|\frac{e}{N\,k_1} - \frac{k_2}{d}| = |\frac{1 - k(N - \varphi(N))}{d\,k_1\,N} \leq |\frac{2k}{d\,k_1\sqrt{N}}| = |\frac{2k_2}{d\sqrt{N}}| \dots \tag{6}$$

— A (new) empirical variant

If k is smooth, we can make the following modification of the attack which becomes more powerful but empirical.

1. We choose a bound B and we compute $\Pi(B) = \prod_{i=1}^{p_i \leq B} p_i$, where p_i is the ith prime number.
2. We compute $\mathcal{L} = \{a_0/b_0, \cdots a_r/b_r\}$ the CFE of $\dfrac{e}{N\,\Pi(B)}$
3. We search for d in the denominators of the elements of $\mathcal{L}\,\Pi(B)$.

The method is *empirical* because, for $y \neq 0$, we have generally $CFE(x) \neq CFE(x/y) * y$.

— Another variant with d

Evidently we can use the same tricks when d is not prime due to the duality between k and d in the expression

$$|\frac{e}{N} - \frac{k}{d}| \tag{7}$$

So, if $d = d_1 d_2$:

$$|\frac{ed_1}{N} - \frac{k}{d_2}| = |\frac{1 - k(N - \varphi(N))}{d_2\, N} \leq |\frac{2k}{d_2\sqrt{N}}| \ldots \tag{8}$$

Current section

— Key length importance.

A RSA key might be used over a period of time not necessarly predictible, and what seemed secure during key generation might not be true during the key life[a]. So, anticipating future improvements in calculus power (Moore's law) is a requirement to preserve secrets and signatures secured by RSA keys over time.

[a] Of course, this is true for any encipher algorithm

Definition (Cryptoperiod of a key in [NIS07]:)

The time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect.

Real Example: it is difficult to anticipate future improvements in algorithms

The most known example of a bad cryptoperiod is perhaps the 320 bits RSA key, the first RSA key used by the French GIE CAB (in charge of the famous *Carte Bleue*). The length of this "weak" RSA key, 320 bits, has been chosen in 1983 but, in 1991, RSA-100 was factorized in less that 7 MIPSY. And, finally, in 1998, Humpich factorized[a] the 320 bits RSA key. Whatever seemed secure in 1983 was no more true in 1998.

[a] Using MPQS, with a single PC!

— How to choose/compute p and q ?

- ANSI X9.31 (1998):
  1. it requires that all numbers $p-1$, $p+1$, $q-1$, and $q+1$ should have primes factors randomly chosen in the range $[2^{100}, 2^{120}]$ (*auxiliary primes*)
  2. Primes are *probably primes* (Miller-Rabin + Lucas test).
  3. p and q have to be $\neq$ in one at least of their first 100 bits.
  4. Algorithm: from a random seed, p and q *shall be the first primes discovered in an appropriate interval, that meets the above*.
- NIST (document for DSS [FIP09],2009): adds the new requirements: the *auxiliary primes* of $p-1$, $p+1$, $q-1$, and $q+1$ and p and q have to be provable primes.

To our knowledge, today, no open source software totally implement this.

— Some factorization algorithms:

Algorithm	Complexity
p-1	$\mathcal{O}(B \log B (\log N)^2)$
ρ	$\mathcal{O}(p^{1/2} (\log N)^2)$
Fermat	$\mathcal{O}(\frac{\lvert p-q \rvert^2}{4N^{1/2}})$

— Complexities ($x = p$ or N): $L[\alpha, c](x) = e^{c(\log x)^{\alpha}(\log\log x)^{1-\alpha}}$.

Algorithm	Complexity	α	c
ECM	$\mathcal{O}(L[\alpha, c](p).(\log(n))^2)$	$1/2$	2
CFRAC	$\mathcal{O}(L[\alpha, c](N))$	$1/2$	$\sqrt{2} \approx 1, 4142$
MPQS	$\mathcal{O}(L[\alpha, c](N))$	$1/2$	$\frac{3}{2\sqrt{2}} \approx 1, 0607$
NFS	$\mathcal{O}(L[\alpha, c](N))$	$1/3$	$(\frac{64}{9})^{1/3} \approx 1, 923$

— Some factorisation records

N	Year	Algorithm
RSA-120 (399 bits)	1993	MQPS
RSA-129 (429 bits)	1994	MPQS
RSA-130 (432 bits)	1996	NFS
RSA-140 (466 bits)	1999	NFS
RSA-155 (512 bits)	1999	NFS
RSA-160 (532 bits)	2003	NFS
RSA-200 (665 bits)	2005	NFS
RSA-768	2010?	NFS
RSA-1024	2030?	??

— The notion of "Security strength" [FIP09]:

A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system. Sometimes referred to as a security level.

Algorithms security lifetime	DSA,D-H	RSA	ECDSA
Through 2010 (80 bits of St.)	(L=1024,N=160)	1024	160
Through 2030 (112 bits of St.)	(L=1024,N=160)	2048	224
Through 2050 (128 bits of St.)	(L=1024,N=160)	3072	256

— Some "Recommendations" about key length

- DCSSI [DCS06] in 2006: *we consider that the use of 1024 bits RSA moduli is a risk incompatible with the criteria of standard robustness*.
- NIST [NIS06a] in 2008:
  1. Authentication keys:
     - Until 2014: RSA 1024 or 2048 bits
     - After 2014: RSA 2048 bits
  2. Digital Signature and Key Establishment Keys :
     - Until 2014: RSA signature or key transport 1024 or 2048 bits
     - After 2014: RSA signature or key transport 2048 bits
  3. CA: RSA 2048, 3072, or 4096 bits

— Unfortunately Silverman [Sil97] says ...

... Suppose we choose our primes for our RSA key such that $p \pm 1$, $q \pm 1$ have no small factors and are thus inaccessible to $P \pm 1$. This does not guard against the existence of a small value of k, $k \neq 1$, such that $p \pm k$ is divisible by only small primes. And if such a k exists, ECM can succeed where $P \pm 1$ fails. It is impossible to guard against all such possible values of k.

Current section

— The *Art of RSA* is hard : A RSA key can be "weak".

- Security in key generation is mainly (only?) based on pseudorandom number generators and "probabilities".
- The chances to obtain a weak key are considered negligible and often therefore no test is done, even the simplest ones.
- This confidence in mathematics can still be shaken when the randomness is not that random. That is what happened to the Debian Random Generator. L. Bello has found that, because of a single line of "miscommented" code, between September 2006 and 2008 the RSA key generated by OpenSSL/Debian were not random but rather highly predictable. For RSA 1024 bits the set of possible key has a very low cardinal of $2^{15} = 32768$.

— The *Art of RSA* is hard : towards a Certificate of PKV ?

There are needs for:

- Approved method for full validation of RSA keys generation
- Approved method for partial validation (at least) of RSA keys generation
- *Cartography* (reality) of how open source software (and if possible commercial software also) compute a RSA key to verify what partial validation of RSA keys they reach.
- Certificate of Public Key Validation (and of course to define what such a thing has to be!) a least for the *a posteriori* problem of legal assurance

So, what could be such a Certificate of Public Key Validation ?

KEGVER

— Key Generation with Verifiable Randomness (KEGVER, [AJ02]):

How to pursuade a verifying party that the key has been generated "securel" ?

- *ad hoc* verifications against class of attacks,
- Zero-knowledge approaches to prove a secured process was used,
- Definition of a distributed key generation protocol

— Certificate of Public Key Validation: future work

It has to contain, *a minima*

- X509 Certificate information : *a priori* ownership
- Information of key generation process (with zero-knowledge proof): *a posteriori* control (PRNG used, seeds, primes, . . .)

Thanks a lot for your attention and your questions are welcomed . . .

— Fermat revisited with Coppersmith's methods 1/2

- Direct approach of Fermat method gives the equation $\mathcal{P}(x, y) = x^2 - y^2 - N$
- Variable change $x = x + R$ with $R = \lceil \sqrt{N} \rceil$ and normalization gives $\mathcal{P}(x, y) = (x + 2R)^2 - y^2 - 4N$
- This is a bivariate integer polynomial equation
- We can solve it with Coppersmith Lattice method [Cop96b, Cop96a, Cop97] if $|p - q| < N^{1/4}$
- we can also consider the *modular* $(x + 2R)^2 - y^2 = 0 \bmod 4N$

— Fermat revisited with Coppersmith's methods 2/2

- we can factor N in a polynomial time if $|p - q| < N^{1/3}$ if we use the Coppersmith method for the *modular* bivariate equation
- We have to point out that we get only an *empirical* method: it works if the the classical "resultant heuristics" holds. (Numerical experiments in progress!) [Cop96b, Cop96a, Cop97, JM06, May07].
- This bound of $1/3$ corresponds for a standard balanced RSA-1024 bits to factors p and q of 512 bits having their 171 most significant bits alike out of 512 and the gain over the FFM is 85 bits.

J. Guajardo A. Juels.
RSA Key generation with verifiable randomness.
Lecture notes in computer science, vol. IX:pp. 357–374, 2002.
PKC 2002.

D. Boneh and G.Durfee.
Cryptanalysis of RSA with private key d less than $N^{0.292}$.
IEEE Transactions on Information Theory, 46, 2000.

D. Coppersmith.
Finding a small root of a bivariate integer equation; factoring with high bits known.
Lecture Notes in Computer Science, XX:178–189, 1996.

D. Coppersmith.
Finding a small root of a univariate modular equation.
Lecture Notes in Computer Science, XX:155–165, 1996.

D. Coppersmith.
Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities.
J. Cryptology, 10(4):233–260, 1997.

DCSSI.
Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques de niveau de robustesse *standard*.
Services du Premier Ministre, 2006.

FIPS PUB 186-3.
Digital signature standard (dss).
http://csrc.nist.gov/publications/fips/fips186-3, June 2009.

IETF.

RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS).
http://www.ietf.org/rfc/rfc3110.txt, May 2001.

E. Jochemsz and A. May.
A Strategy for Finding Roots of Multivariate Polynomials with New Applications in Attacking RSA Variants.
Lecture Notes in Computer Science, 4284:267, 2006.

A. May.
Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey.
In *LLL+ 25 Conference in honour of the 25th birthday of the LLL algorithm*, 2007.

P. Nguyen.

Can we trust cryptographic software? cryptographic flaws in gnu privacy guard v1.2.3.
In *EUROCRYPT 2004*, pages 555–570.

NIST Special Publication 800-57.
SP 800-57, Recommendation for Key Management - Part 3: Application-Specific Key Management Guidance (draft).
http://csrc.nist.gov/publications/drafts/800-57-part3, November 2006.

NIST Special Publication 800-89.
Recommendation for Obtaining Assurances for Digital Signature Applications.
http://csrc.nist.gov/publications/nistpubs/800-89/SP-800-89, November 2006.

NIST Special Publication 800-57.

SP 800-57, Recommendation for Key Management - Part 1: General (Revised).
http://csrc.nist.gov/publications/drafts/800-57-part3, March 2007.

NSA.
Fact Sheet Suite B Cryptography.
http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml, 2009.

R. D. Silverman.
Fast Generation of Random, Strong RSA Primes.
Cryptobytes, 3, Spring 1997.
Number 1.

M. J. Wiener.
Cryptanalysis of short RSA secret exponent.
IEEE Trans. Information Theory, 36:553–558, 1990.

S. Y. Yan.
Cryptanalytic attacks on RSA.
Springer, 2008.