A new idea for RSA backdoors

Hiding efficient backdoors in our cryptosystems

Marco Cesati

University of Rome Tor Vergata

April 13, 2022

RSA

- Rivest, Shamir, Adleman, 1977
- Primes p, q, and the product $N = p\,q$
  - typically N is *balanced*: p and q have nearly the same size ($\ell(p) \simeq \ell(q)$)
- Exponents e, d such that $\gcd(e, \phi(N)) = 1$ and $e\,d \equiv 1 \pmod{\phi(N)}$, where $\phi(x)$ is originally the *Euler's totient function*
  - the number of positive integers up to x that are relatively prime to x
  - $\phi(N) = \phi(p)\,\phi(q) = (p-1)\,(q-1)$
  - if $\gcd(a, N) = 1$, $a^{\phi(N)} \equiv 1 \pmod{N}$
- (N, e) is the *public key*: message M ($< N$) is encrypted as $M^e \bmod N$
- (N, d) is the *private key*: message M' is decrypted as $M'^d \bmod N$

$$\boxed{(M^e)^d \equiv M^{e\,d} \equiv M^{s\,\phi(N)+1} \equiv M\,M^{s\,\phi(N)} \equiv M\,1^s \equiv M \pmod{N}}$$

Main recipe to break RSA

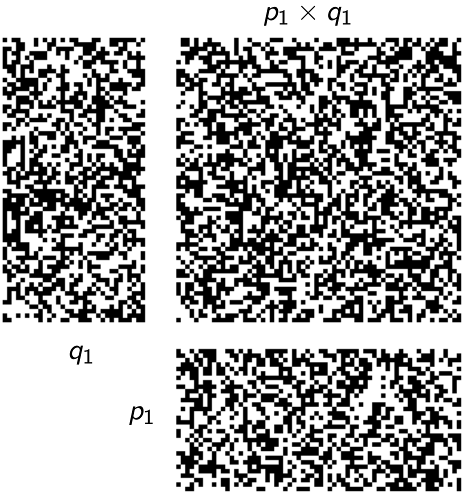
1. Find the factors p and q of $N = p\,q$
2. Compute $\phi(N) = (p-1)\,(q-1)$
3. By the extended Euclidean algorithm, compute $d \equiv e^{-1} \pmod{\phi(N)}$

- RSA is not harder than the integer factorization problem
- Best known factorization algorithm is GNFS, with heuristic runtime

$$L_N[1/3, (64/9)^{1/3}] = \exp\left(\left((64/9)^{1/3} + o(1)\right)\,(\ln N)^{1/3}\,(\ln\ln N)^{2/3}\right)$$

- Nowadays, RSA keys with $\ell(N) = 4096$ bits look safe

Main recipe to break RSA



Main recipe to break RSA

1. Find the factors p and q of $N = p\,q$
2. Compute $\phi(N) = (p-1)\,(q-1)$
3. By the extended Euclidean algorithm, compute $d \equiv e^{-1} \pmod{\phi(N)}$

- RSA is not harder than the integer factorization problem
- Best known factorization algorithm is GNFS, with heuristic runtime

$$L_N[1/3, (64/9)^{1/3}] = \exp\left(\left((64/9)^{1/3} + o(1)\right) (\ln N)^{1/3} (\ln \ln N)^{2/3}\right)$$

- Nowadays, RSA keys with $\ell(N) = 4096$ bits look safe

A *backdoor* in RSA is a method to forge public keys that are significantly easier to break

Who put backdoors in RSA?

- Developers of RSA cryptosystem implementations
  - Just ignorance or lazyness (maybe. . .)
  - Elementary vulnerabilities (e.g., unsafe primes)
- Criminals, spies, and other hidden agents
  - To easily get valuable data from encrypted channels
  - Weak backdoors: anyone just knowing that the vulnerability exists may break the public key
- State-level agencies
  - To lawfully enforce key escrow mechanisms
  - SETUP (Secretely Embedded Trapdoor with Universal Protection) backdoors: nobody just knowing that the vulnerability exists can easily break the public key

Hiding contrived backdoors

The final user of a RSA cryptosystem would not be happy to know about the existence of a backdoor

Some backdoors are easier to hide than others:

- Backdoors affecting the public exponent e cannot be easily hidden
  - For efficiency reasons, all RSA cryptosystems select a fixed public exponent of small bitsize and/or small Hamming weight
- Backdoors based on crafted values of the semiprime N are easier to hide but harder to devise

In the following we shall consider only backdoors that do not affect the choice of the public exponent

Roots of this work: Anderson's backdoor

In 1993 Anderson proposed the following RSA backdoor for N with $\ell(N) = n$:

- β is a m-bit secret prime (the "backdoor key"), whith $m \approx (3/8) \cdot n$
- π_β and π'_β are pseudo-random functions that yield $(n/2-m)$-bit values
- $t, t' < \sqrt{\beta}$ are $(m/2)$-bit random integers coprime with β and such that
- $p = \pi_\beta(t) \cdot \beta + t$ and $q = \pi'_\beta(t') \cdot \beta + t'$ are primes
- To exploit the backdoor, given $N = p\,q$ and β:
  1. Compute $t\,t' = N \bmod \beta$
  2. Factorize the m-bit integer $t\,t'$
  3. Apply $\pi_\beta(t)$ and $\pi'_\beta(t')$ and compute p and q

Main drawback: exploiting requires to factorize an integer of size $\approx (3/8) \cdot \ell(N)$

- Still too much for currently used RSA key sizes (factorization records is 829 bits)

Roots of this work: Implicit Factorization

May and Ritzenhofen introduced in 2009 the idea of *implicit factorization*:

Given semiprimes $N_1 = p_1\, q_1$ and $N_2 = p_2\, q_2$ of size n such that

- $\ell(q_1) = \ell(q_2) = \alpha$
- $\ell(p_1) = \ell(p_2) = n - \alpha$
- $p_1 \equiv p_2 \pmod{2^t}$, with $t \geq 2\,\alpha + 3$

it is possible to factorize N_1 and N_2 in time $O(n^2)$ by searching a base for a suitable lattice by means of the quadratic Gaussian algorithm

This result can be extended to $k > 2$ semiprimes by using Coppersmith's root finding algorithm and Lenstra-Lenstra-Lovàsz (LLL) lattice basis reduction algorithm

In the following years, dozens of authors improved and extended these results

Roots of this work: Implicit Factorization (2)

A backdoor based on implicit factorizations has major drawbacks:

- Weak: no secret key protects it
- Cannot be applied to balanced semiprimes: $\ell(p_i) > 2\,\ell(q_i)$
- Cannot be hidden: the final user may look at the factors of the semiprimes and recognize that long sequences of bits are equal

Roots of this work: Implicit Factorization (2)

Roots of this work: Implicit Factorization (2)

A backdoor based on implicit factorizations has major drawbacks:

- Weak: no secret key protects it
- Cannot be applied to balanced semiprimes: $\ell(p_i) > 2\,\ell(q_i)$
- Cannot be hidden: the final user may look at the factors of the semiprimes and recognize that long sequences of bits are equal
- In general, for all published variants:
  - the exploiting algorithm is polynomial only if we regard the "unbalancing" difference between $n - \alpha$ and α as a constant
  - the runtime is exponential in $1/(n - 2\,\alpha)$

The idea for a new RSA backdoor

Is it possible to blend the core idea of Anderson's backdoor and the Implicit Factorization Problem and get an efficient, undetectable, and password-protected backdoor for large, balanced semiprimes like those in RSA-4096?

Eventually we got two different backdoors:

- Twin Semiprime Backdoor (TSB): a backdoor involving two paired semiprimes
- Single Semiprime Backdoor (SSB): a backdoor for a single semiprime

TSB: generation of trapped semiprimes

A pair of balanced semiprimes $N_1 = p_1\, q_1$ and $N_2 = p_2\, q_2$ is generated as follows:

- Let $\alpha = \ell(N_1)/2 = \ell(N_2)/2$ (α is the common size of all factors of N_1 and N_2)
- Fix a small constant c; typically, $c = 7$ is fine for α in the range from 512 (RSA-1024) to 2048 (RSA-4096)
- Randomly select a prime T of size $\ell(T) = \alpha - c$; this is the *backdoor key*
- Fix the value of a constant K (e.g, $K \approx \alpha/5$ for current RSA key sizes)
- Fix the value of some constant $B < T$ such that $B \simeq 2^{\alpha - 2\, c}$

TSB: generation of trapped semiprimes (2)

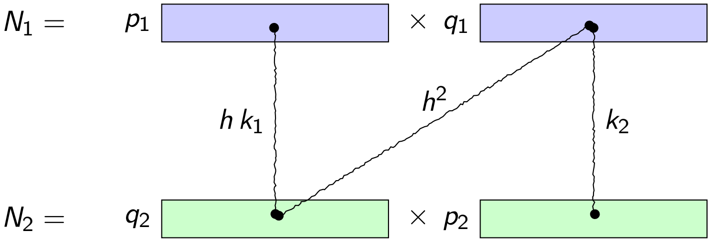
- Generate random primes p_1, q_1, p_2, and q_2 such that there exists h, k_1, k_2 positive integers between 2 and K satisfying:

$$\begin{array}{ll}
\text{(H1)} & q_2 \equiv h^2\, q_1 \pmod{T} \\
\text{(H2)} & p_1 \equiv h\, k_1\, q_2 \pmod{T} \\
\text{(H3)} & p_2 \equiv k_2\, q_1 \pmod{T} \\
\text{(H4)} & h,\ k_1,\ \text{and } k_2 \text{ are all coprime} \\
\text{(H5)} & h\, k_1 \not\equiv k_2 \pmod{T} \\
\text{(H6)} & (h\, q_1)^2 \bmod T > B
\end{array}$$

The existence of these values is granted by Dirichlet's theorem: there are infinitely many primes of the form $a + b\,c$ when a and b are coprime

Algorithm: randomly pick q_1, then search h, q_2, then p_1, k_1, and finally p_2, k_2

TSB: generation of trapped semiprimes (3)



TSB: recovering procedure

How to recover the factors from N_1, N_2, and T:

1. Get "medium-level" coefficients
2. Get "low-level" coefficients
3. Get "high-level" coefficients
4. Determine the factors

Running example:

- $\alpha = 64$, $c = 3$, $K = 100$, $B = 2^{57}$
- $T = 135085609344009833$
- $N_1 = 199771249142689629600100193795300988277$
- $N_2 = 330849388672597230630022641974377014199$

TSB: recovering medium-level coefficients

From conditions H1, H2, and H3:

$$N_1 \equiv h^3\, k_1\, q_1^2 \pmod{T} \quad \text{and} \quad N_2 \equiv h^2\, k_2\, q_1^2 \pmod{T}$$

However, $\gcd(N_1 \bmod T, N_2 \bmod T) \neq h^2\, q_1^2$. We rather have

$$N_1 \equiv (h\, k_1) \cdot [(h^2\, q_1^2) \bmod T] \pmod{T} \quad \text{and} \quad N_2 \equiv k_2 \cdot [(h^2\, q_1^2) \bmod T] \pmod{T}$$

thus there exist *medium-level* coefficients $\tilde{k}_1, \tilde{k}_2$ such that

$$(N_1 \bmod T) + \tilde{k}_1 \cdot T = (h\, k_1) \cdot [(h^2\, q_1^2) \bmod T]$$

$$(N_2 \bmod T) + \tilde{k}_2 \cdot T = k_2 \cdot [(h^2\, q_1^2) \bmod T]$$

where $\tilde{k}_1 \leq K^2$ and $\tilde{k}_2 \leq K$

TSB: recovering medium-level coefficients (2)

We consider every pair $(\tilde{k}_1, \tilde{k}_2)$ with $\tilde{k}_1 < K^2$ and $\tilde{k}_2 < K$, and compute:

$$\gcd((N_1 \bmod T) + \tilde{k}_1 \cdot T,\ (N_2 \bmod T) + \tilde{k}_2 \cdot T) \quad = \quad (h^2\, q_1^2) \bmod T$$

Candidate pairs $(\tilde{k}_1, \tilde{k}_2)$ can be efficiently filtered because

- by condition H6, $T > (h\, q_1)^2 \bmod T > B$, a large threshold
- the Euclidean algorithm must return a square in $\mathrm{GF}(T)$

There are only two pairs $(\tilde{k}_1, \tilde{k}_2) \in [2, 100^2] \times [2, 100]$ that yield a GCD higher than $B = 2^{57}$: $(671, 10)$ and $(5277, 79)$

$$(671, 10){:}\ \ 196865400950880229 \equiv 106325596553639O8^2 \pmod{T}$$

$$(5277, 79){:}\ \ 1547721494390890062 > T \text{ (discarded)}$$

TSB: recovering low-level coefficients

So we may assume to know N_1, N_2, T, $\tilde{k}_1$, $\tilde{k}_2$, and $\gamma^2 = (h\,q_1)^2 \bmod T$

We compute low-level coefficients k_1, k_2, and h as:

$$k_2 = \Big((N_2 \bmod T) + \tilde{k}_2 \cdot T \Big) / \gamma^2$$

$$(h\,k_1) = \Big((N_1 \bmod T) + \tilde{k}_1 \cdot T \Big) / \gamma^2$$

Because $h\,k_1 < K^2$ and $\gcd(h, k_1) = 1$, the number of multiplicative partitions of the product $h\,k_1$ is $\leq K^2$: we build a list of candidate pairs (h, k_1)

The exact integer divisions yield $k_2 = 69$ and $(h\,k_1) = 4606 = 2 \cdot 7^2 \cdot 47$
There are six candidate pairs for (h, k_1): $(2, 2303)$, $(47, 98)$, $(49, 94)$, $(94, 49)$, $(98, 47)$, and $(2303, 2)$

TSB: recovering high-level coefficients

From N_1, N_2, T, k_1, k_2, h, and γ^2 we immediately get:

$$\begin{aligned}
\gamma &= \sqrt{\gamma^2} \bmod T && \text{(two values, by Tonelli-Shanks alg.)} \\
q_1 \bmod T &= (\gamma\, h^{-1}) \bmod T && (h^{-1} \text{ is the inverse in } \mathrm{GF}(T)) \\
q_2 \bmod T &= ((q_1 \bmod T) \cdot h^2) \bmod T && \text{(condition H1)} \\
p_1 \bmod T &= (h\, k_1\, (q_2 \bmod T)) \bmod T && \text{(condition H2)} \\
p_2 \bmod T &= (k_2\, (q_1 \bmod T)) \bmod T && \text{(condition H3)}
\end{aligned}$$

The square roots of $\gamma^2 = 196865400950880229$ in $\mathrm{GF}(T)$ are $\gamma_1 = 106325596553639 08$ and $\gamma_2 = 134022353378464 5925$

TSB: recovering high-level coefficients (2)

The two values for γ and the six candidate pairs (h, k_1) yield:

h, k_1, γ	$q_1 \bmod T$	$q_2 \bmod T$	$p_1 \bmod T$	$p_2 \bmod T$
$2, 2303, \gamma_1$	5316279827681954	2126511931072781 6	6855008175316125 20	3668233081100548 26
$2, 2303, \gamma_2$	13455398136123278 79	13295909741292820 17	6653552759083973 13	9840327853299550 07
$47, 98, \gamma_1$	12648574610854424 80	4997303038021036 76	12498521841527860 57	8203748347349018 08
$47, 98, \gamma_2$	8599863235456735 3	8511257896379061 57	1010039092872237 76	5304812587051080 25
$49, 94, \gamma_1$	3310388914476628 96	5209954231128314 92	5844969082443887 44	12279860148485824 96
$49, 94, \gamma_2$	10198172019923469 37	8298606703271783 41	7663591851956210 89	1228700785914273 37
$94, 49, \gamma_1$	6324287305427212 40	9994606076042073 52	11488482748655622 81	4101874173674509 04
$94, 49, \gamma_2$	7184273628972885 93	3513954858358024 81	2020078185744475 52	9406686760725589 29
$98, 47, \gamma_1$	1655194457238314 48	10419908462256629 84	11689938164887774 88	6139930074242912 48
$98, 47, \gamma_2$	11853366477161783 85	3088652472143468 49	1818622769512323 45	7368630860157185 85
$2303, 2, \gamma_1$	4669092848188897 92	1713752043829031 30	4542328186860743 08	11470505033831694 89
$2303, 2, \gamma_2$	8839468086211200 41	11794808890571067 03	8966232747539355 25	2038055900568403 44

TSB: recovering high-level coefficients (3)

Now we proceed separately on each semiprime:

$$N_i = p_i\, q_i = (\pi_i \cdot T + (p_i \bmod T)) \cdot (\nu_i \cdot T + (q_i \bmod T))$$

that is

$$\pi_i\, \nu_i\, T + \pi_i\, (p_i \bmod T) + \nu_i\, (q_i \bmod T) = (N_i - (p_i \bmod T)\, (q_i \bmod T))\, / T = \delta_i$$

We can search by brute force the "high-level" coefficients ν_i and π_i because $\pi_i\, \nu_i \approx N_i / T^2$, thus $\ell(\pi_i\, \nu_i) = \ell(\pi_i) + \ell(\nu_i) \simeq 2\,\alpha - 2\,(\alpha - c) = 2\,c$

The bounds for the high-level coefficients are: $\pi_1\, \nu_1 \leq 110$, $\pi_1 + \nu_1 \in [20, 110]$, $\pi_2\, \nu_2 \leq 182$, $\pi_2 + \nu_2 \in [26, 182]$.

TSB: recovering high-level coefficients (3)

Brute force search on all values for $\pi_i + \nu_i$:

if $x = \pi_i$, $C = \pi_i + \nu_i = x + \nu_i$, $\delta_i = (N_i - (p_i \bmod T)(q_i \bmod T))/T$, then

$$T\,x^2 + ((p_i \bmod T) - (q_i \bmod T) - C\,T)\,x + \delta_i - (q_i \bmod T)\,C = 0$$

We discard any value for $C = \pi_i + \nu_i$ that do not yield integer solutions:

$$\Delta = ((p_i \bmod T) - (q_i \bmod T) - C\,T)^2 - 4\,T\,(\delta_i - (q_i \bmod T)\,C)$$

must be a square, and either one of the solutions

$$\left(C\,T + (q_i \bmod T) - (p_i \bmod T) \pm \sqrt{\Delta}\right)/(2\,T) \quad \text{must be an integer}$$

The brute force search is repeated on the 12 candidate coefficients.
Eventually, only the following coefficients yield integer solutions:
$h = 47$, $k_1 = 98$, $\gamma = \gamma_2$, $(\pi_1, \nu_1) = (9, 12)$, $(\pi_2, \nu_2) = (12, 14)$

TSB: recovering the factors

In this phase we know N_i, T, and a list of candidate solutions for $p_i \bmod T$, $q_i \bmod T$, π_i, and ν_i

We just compute $p_i = \pi_i\, T + (p_i \bmod T)$ and $q_i = \nu_i\, T + (q_i \bmod T)$, and verify that $N_i = p_i\, q_i$

$$p_1 = \pi_1\, T + (p_1 \bmod T) = 1225870875024731 2273$$

$$q_1 = \nu_1\, T + (q_1 \bmod T) = 1629627175363468 5349$$

$$p_2 = \pi_2\, T + (p_2 \bmod T) = 1674075437998522 6021$$

$$q_2 = \nu_2\, T + (q_2 \bmod T) = 1976311109779804 3819$$

$$N_1 = 12258708750247312273 \times 16296271753634685349$$

$$N_2 = 16740754379985226021 \times 19763111097798043819$$

TSB: time complexity

- The worst-case time complexity of the recovering procedure is $O(K^5\,(\alpha + c)^2\,2^{2c})$
- Good values of K and c must be related to α, however $K < \alpha$ and $c \ll \alpha$
- Polynomial runtime in the size of semiprime $(2\,\alpha)$
- This backdoor may be efficient even for very large semiprimes
- Larger values for K and c yield
  - Faster semiprime generation procedures
  - Slower recovery procedures

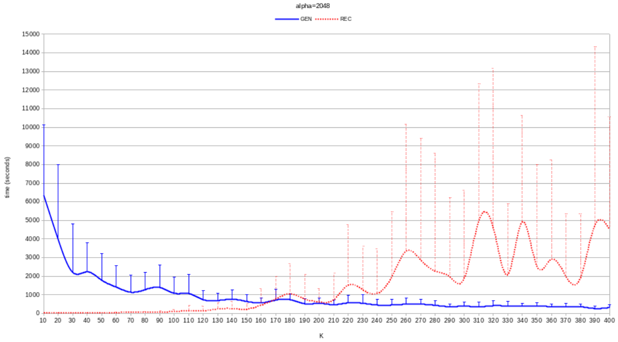
However c cannot be too large, otherwise it would be possible to detect and exploit the backdoor by guessing T, which has size $\alpha - c$

TSB: experimental results

Implementation in SageMath available at:

`https://gitlab.com/cesati/ssb-and-tsb-backdoors.git`

Experimental results for RSA-4096 keys ($c = 7$, times in seconds, 20 repetitions):



K	Generation avg	Generation stdev	Recovering avg	Recovering stdev
10	6353.2	3759.4	31.1	9.3
50	1785.4	1429.4	43.1	12.5
100	1086.0	887.3	104.4	108.7
150	647.1	376.5	236.3	290.4
200	544.3	277.6	619.9	729.6
250	456.8	305.3	1976.0	3493.5
300	395.4	236.6	1910.5	4716.5
350	407.6	155.3	2537.8	5460.6
400	321.1	140.0	4541.4	6038.2

SSB: the idea applied to a single key

The idea behind TSB can also be applied, in a simpler form, to a single semiprime

- α, c, K, T as in TSB
- $N = p\,q$ where $p \equiv k\,q \pmod{T}$, $1 < k \leq K$

The recovering procedure is also similar:

1. Recover "low-level" coefficient (k)
2. Recover "high-level" coefficients
3. Recover the factors

SSB: recovering procedure

From the two congruences $N \equiv p\,q \pmod{T}$ and $p \equiv k\,q \pmod{T}$ we derive

$$N \bmod T \equiv (k\,q^2) \pmod{T}$$

For any candidate $k < K$:

- if $N \cdot k^{-1}$ is a quadratic nonresidue in GF(T), discard this value of k
- otherwise compute the square roots (candidates for $q \bmod T$) and the corresponding $p \bmod T$

Having N, T, $p \bmod T$, and $q \bmod T$, the procedure continue as in TSB by looking for the high-level coefficients π and ν such that

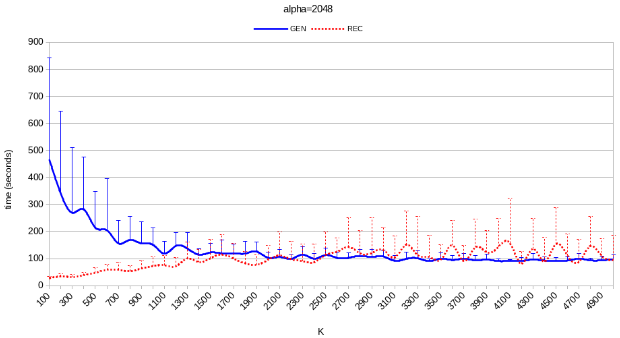
$$N = (\pi \cdot T + (p \bmod T))\,(\nu \cdot T + (q \bmod T))$$

SSB: experimental results

Implementation in SageMath available at:

`https://gitlab.com/cesati/ssb-and-tsb-backdoors.git`

Experimental results for RSA-4096 keys ($c = 7$, times in seconds, 20 repetitions):



	Generation		Recovering	
K	avg	stdev	avg	stdev
100	466.4	375.1	27.1	6.9
500	214.0	134.7	47.4	18.6
1,000	151.2	61.3	71.6	37.4
1,500	122.3	35.0	101.2	70.0
2,000	102.7	20.3	95.8	51.7
2,500	112.7	25.4	113.2	84.8
3,000	107.6	23.0	130.7	84.1
3,500	99.5	22.6	95.5	54.8
4,000	90.4	9.1	143.1	104.4
4,500	91.5	13.5	152.8	136.6
5,000	97.3	16.9	94.5	91.5

Conclusions

- Further details: `https://arxiv.org/abs/2201.13153v1`
- TSB and SSB may inject exploitable vulnerabilities in any cryptosystem based on the integer factorization problem
- Currently, no way to discover and exploit the backdoors without the designer key
- We should really never use closed-source RSA key generators

Thank you!