Reconstructing RSA Private Keys from Random Key Bits

Nadia Heninger and Hovav Shacham
Princeton UCSD

August 17, 2009

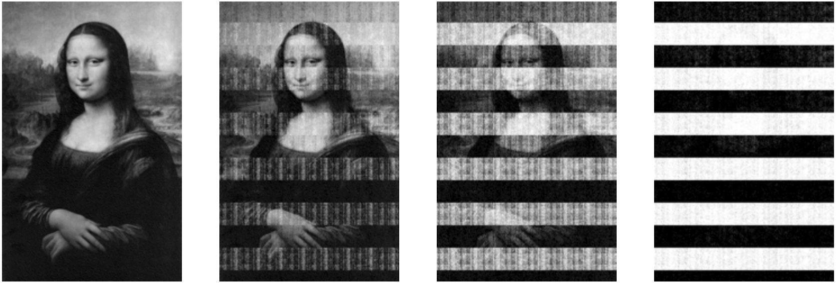
Motivation: "Cold boot" or "memory" attacks

A new side-channel attack on cryptographic keys
that leaks information independently of computation.

- Data persists on RAM after power is removed: up to several seconds at room temperature, longer if cooled.
- An attacker can reboot the computer to get around OS controls on memory access.

Actually works in practice against software disk encryption.
[HSHCPCFAF 08]

Motivation: Properties of memory remanence and decay



5s. 30s. 1m. 5m.

- Model DRAM as an array of capacitors that discharge to a known ground state.
- Example: In region of ground state 0. If read in a 1, that bit must be 1. If read in 0, original bit could have been 0 or 1.
- The decay order is relatively random.

Recent work on memory attacks

Theoretical constructive work:

- Show existing lattice-based cryptosystems resistant to this sort of attack. [Akavia, Goldwasser, Vaikuntanathan 09]
- Create new, resistant DDH-based cryptosystems. [Naor and Segev, this session!]
- Create new protocols that can tolerate a fixed rate of key leakage over time. [Alwen, Dodis, Wichs, this session!]

Empirical attacks:

- DES trivial to reconstruct from about 25% of bits. [HSHCPCFAF 08]
- Reconstruct an AES key schedule from 30% of bits. [Tsow 09]
- Reconstruct an RSA private key from 27% of bits. [this work]

Problem Statement

Remove all but a δ-fraction of the bits, chosen at random, from an RSA private key.

(Flip a coin at each bit of the key. With probability δ, the attacker gets to see the bit's value.)

How to efficiently reconstruct the key?

(Spoiler!)

We can do this with $\delta = 27\%$ of the private key bits for small public exponent. (Under a heuristic assumption.)

Outline for the rest of the talk

Useful facts about RSA keys.

Recovery Algorithm

1. Write relationships between key values as equations over the integers.
2. Solve series of equations iteratively over key bits.

Analysis. (And our assumption.)

Experimental results.

Notation and RSA review

Public Key

N $= pq$ modulus

e encryption exponent

Private Key

p, q large primes

d $= e^{-1} \mod (p-1)(q-1)$ decryption exponent

Encryption

$$c = m^e \pmod{N}$$

Decryption

$$m = c^d \pmod{N}$$

(for speed, decrypt using Chinese remainder theorem)

$$d_p = d \pmod{p-1}$$

$$d_q = d \pmod{q-1}$$

Observation: Key data is redundant.

PKCS #1: RSA Cryptography Standard

```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER,  -- n
    publicExponent    INTEGER   -- e
}
RSAPrivateKey ::= SEQUENCE {
    version           Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

Observation→Assumption: Low public exponent

Nearly everyone uses the public exponent

$$e = 2^{16} + 1 = 65537.$$

In this work, we assume that e is small.

Step # 1: Relate key values

We can write down the relationships between redundant key information as equations.

$$pq = N \tag{1}$$

$$ed = 1 \pmod{(p-1)(q-1)} \tag{2}$$

$$ed_p = 1 \pmod{p-1} \tag{3}$$

$$ed_q = 1 \pmod{q-1} \tag{4}$$

Step # 1: Relate key values over the integers

We can write down the relationships between redundant key information as equations over the integers.

$$pq = N \tag{1}$$

$$ed + k(p+q) = 1 + k(N-1) \tag{2}$$

$$ed_p - g(p-1) = 1 \tag{3}$$

$$ed_q - h(q-1) = 1 \tag{4}$$

(upper half of bits of d)

$$k = \frac{e\,\tilde{d} - 1}{N+1} \quad \text{(trick from [Boneh, Durfee, Frankel 98])}$$

$$g^2 - [k(N-1)+1]g - k \equiv 0 \pmod{e}$$

Natural but unsuccessful Idea: Lattice approaches

Lattice approaches used for RSA key recovery in:

[Coppersmith 96], [Boneh, Durfee, Frankel 98], [Blömer and May 03], [Herrmann and May 08]

What they have:	What we have:
 Large blocks of contiguous bits, no redundancy.	 Non-contiguous bits, redundancy.

Open problem: Make a lattice approach work. (We couldn't.)

For example the paper tries to factor $N = pq$ by writing it as a set of binary equations over the bits of p and q.

–Ten Reasons why a Paper is Rejected from a Crypto Conference

Step #2: Solve our equations iteratively

Generate a tree of partial solutions, starting at bit 0.

What's a tree node?

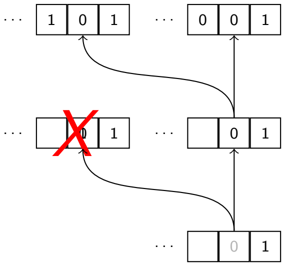
A simultaneous assignment of bits $[0 \ldots i]$ of p, q, d, d_p, d_q.

It's easy to lift a solution mod 2^i to all equivalent solutions mod 2^{i+1}.

How much branching at each level?

32? No, 4 equations for 5 unknowns.

2? No, we can prune a solution when it conflicts with our known bits.



Step #2: Solve our equations iteratively

Algorithm:

1. Enumerate tree of partial solutions.
2. Prune incorrect solutions.

Analysis: Overall structure

At every step, we have one good solution and some number of bad solutions. The number of bad solutions determines the runtime.

- Model the generation of bad solutions as a statistical branching process.
- We can use the machinery of generating functions to analyze this branching process.
- Our machinery tells us that the number of solutions we generate at step i is determined by the number of new bad solutions generated from an old bad solution.

(This is where we're going to use the fact that we have a uniform distribution of known bits and not adversarial.)

Analysis: Model the branching as a statistical process

Write a generating function to represent the distribution of the number of bad solutions generated at every step.

- $g(s)$ counts bad solutions generated from a good solution
- $b(s)$ counts bad solutions generated from a bad solution.
- $F_i(s)$ counts the total bad solutions at step i

F_i satisfies a nice recurrence:

$$F_{i+1}(s) = F_i(b(s))g(s)$$

Solve the recurrence to learn the expected number of bad solutions at step i:

$$F_i'(1) = \frac{g'(1)}{1 - b'(1)}(1 - b'(1)^i)$$

When $b'(1) < 1$, the expected number of bad solutions at any step is bounded.

Analysis: Model the branching as a statistical process

Write a generating function to represent the distribution of the number of bad solutions generated at every step.

- $g(s)$ counts bad solutions generated from a good solution
- $b(s)$ counts bad solutions generated from a bad solution.
- $F_i(s)$ counts the total bad solutions at step i

F_i satisfies a nice recurrence:

$$F_{i+1}(s) = F_i(b(s))g(s)$$

Solve the recurrence to learn the expected number of bad solutions at step i:

$$F_i'(1) = \frac{g'(1)}{1 - b'(1)}(1 - b'(1)^i)$$

When $b'(1) < 1$, the expected number of bad solutions at any step is bounded.

Analysis: Bound the expectation of bad solutions

The overall behavior of the algorithm is determined by $b'(1)$.

$$
\begin{aligned}
p[i] + q[i] &\equiv c_1 \pmod{2} \\
d[i] + p[i] + q[i] &\equiv c_2 \pmod{2} \\
d_p[i] + p[i] &\equiv c_3 \pmod{2} \\
d_q[i] + q[i] &\equiv c_4 \pmod{2}
\end{aligned}
$$

Conjecture

An incorrect partial solution ends up producing c_i at random.

$$b'(1) = \mathrm{E}(\#solutions) = \frac{(2-\delta)^5}{2^4}$$

Open problem: Prove or disprove.
(Experimentally, this is close to being true.)

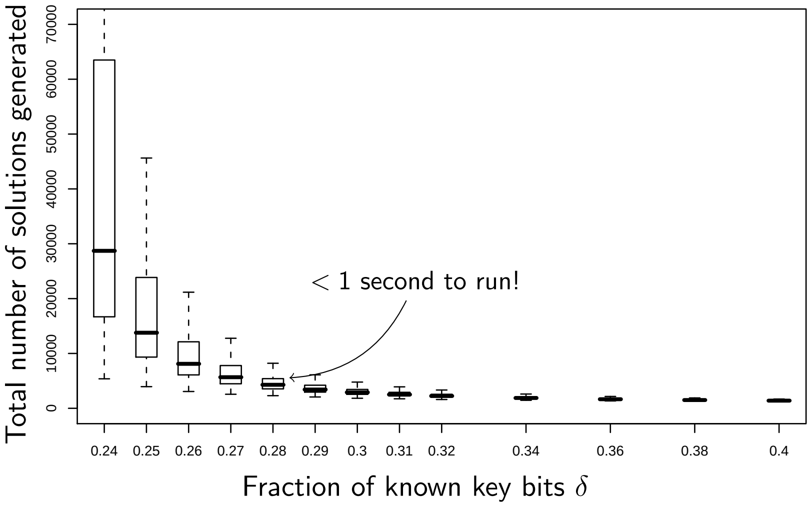
Results for different key redundancy

If the attacker has partial knowledge of...	... then recovery is efficient for...
d, p, q, d_p, d_q	$\delta > 2 - 2^{\frac{4}{5}} \approx .2589$
d, p, q	$\delta > 2 - 2^{\frac{3}{4}} \approx .4126$
p, q	$\delta > 2 - 2^{\frac{1}{2}} \approx .5859$
p	**Open problem**

fraction of key bits known.

Experimental validation of analysis

Total number of solutions generated vs. fraction of known bits δ



(More than 1 million experiments.)

Summary

Motivation:

Cold boot attacks.

Assumptions:

Redundant key data, low public exponent.

Attack algorithm

1. Relate redundant key values over the integers.
2. Iteratively solve equations.

Analysis

Model the branching process using statistics, heuristic assumption.
Analysis validated by experiments.

Open problems

- How can we use $q^{-1} \pmod{p}$?
- How true is our conjecture that an incorrect solution looks random?
- Is it possible to improve this using lattice methods?
- Is it possible to apply more intelligent decoding methods?
- Can you factor using knowledge of bits in random positions of only p?