

Public-Key Cryptanalysis 3: Lattices, Low Exponents, and Partial Keys

Nadia Heninger

University of Pennsylvania

July 21, 2013

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N
```

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N

sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N
```

```
sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

The message is too small.

This is why we use padding.

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N

sage: int(c^(1/3))==message
False
```

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayiswordfish',base=35)
c = message^3 % N
```

This is a stereotyped message. We might be able to guess the format.

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayiswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayisxxxxxxxxxx',base=35)
```

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayiswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayisxxxxxxxx',base=35)
X = Integer('xxxxxxxx',base=35)
M = matrix(7)
```

M is coefficient vectors of polynomials

$N^2, N^2xX, N^2(xX)^2, N((a + xX)^3 - c), \dots, ((a + xX)^3 - c)^2.$

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayiswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayisxxxxxxxx',base=35)
X = Integer('xxxxxxxx',base=35)
M = matrix(7)
```

M is coefficient vectors of polynomials

$N^2, N^2xX, N^2(xX)^2, N((a + xX)^3 - c), \dots, ((a + xX)^3 - c)^2.$

```
B = M.LLL()
f = sum(B[0][i]*(x/X)^i for i in range(7))
```

```
sage: f.factor()[0]
(x + 11340606574691, 1)
```

```
N = random_prime(2^128)*random_prime(2^128)
message = Integer('thepasswordfortodayiswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayisxxxxxxxx',base=35)
X = Integer('xxxxxxxx',base=35)
M = matrix(7)
```

M is coefficient vectors of polynomials

$N^2, N^2xX, N^2(xX)^2, N((a + xX)^3 - c), \dots, ((a + xX)^3 - c)^2.$

```
B = M.LLL()
f = sum(B[0][i]*(x/X)^i for i in range(7))
```

```
sage: f.factor()[0]
(x + 11340606574691, 1)
```

```
sage: (a-11340606574691).str(base=35)
'thepasswordfortodayiswordfish'
```

What's going on here? Coppersmith's method.

Theorem (Coppersmith)

We can efficiently compute up to $1/e$ -fraction of the bits of an RSA-encrypted message with public exponent e if we know the rest of the plaintext.

Theorem (Coppersmith)

Given a polynomial f of degree d and N , we can efficiently find all roots r_i satisfying

$$f(r_i) \equiv 0 \pmod{N}$$

when $|r_i| < N^{1/d}$.

The stereotyped message problem gives us an a satisfying

$$(a + x)^3 - c \equiv 0 \pmod{N}$$

Why is this an interesting theorem?

1. A general method to solve polynomials mod N would break RSA:

$$x^e - c \equiv 0 \pmod{N}$$

2. There is an efficient algorithm to solve equations mod primes.
 - ▶ For a composite, factor into primes, solve mod each prime, and use Chinese remainder theorem to lift solution mod N .
3. By accepting a bound on solution size, Coppersmith's method lets us solve equations **without factoring N** .

Coppersmith's Algorithm Outline

Input: polynomial f , modulus N

1. Construct a matrix of coefficient vectors of powers of f and N :

$$N^k, xN^k, \dots, N^{k-1}f, \dots, f^k, xf^k, \dots$$

2. Run a lattice basis reduction algorithm on this matrix.
3. Construct a polynomial Q from the shortest vector output.
4. Factor Q to find its roots.

What is a lattice?

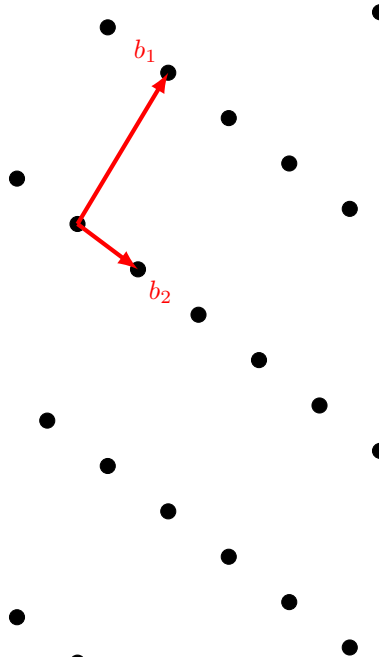
Definition

A **lattice** is a set of points in space generated by integer linear combinations of some basis vectors $\{b_1, \dots, b_n\}$.

Theorem (LLL)

We can find a vector of length

$$|v| < 2^{\dim L} (\det L)^{1/\dim L}$$

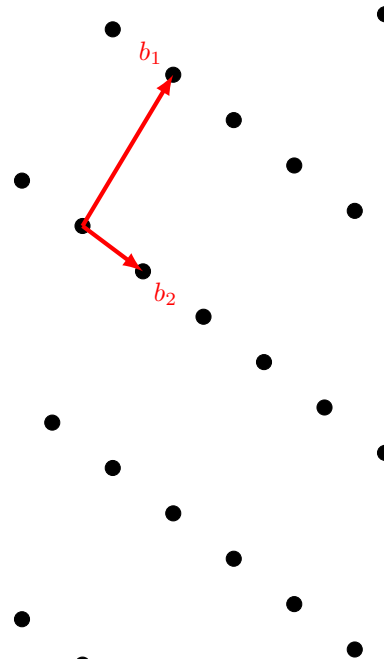


Lattices in practice

- ▶ We use LLL as a black box.
- ▶ In practice, LLL finds vectors of length $1.02^{\dim L} (\det L)^{1/\dim L}$ for random inputs.

Open problem: Explain this behavior. (Nguyen, Stehle 2006)

- ▶ All our lattices are small dimension, so ignore approximation factor.
- ▶ All our matrices are diagonal, so $\det L$ is product of diagonal entries.



```
p = random_prime(2^512); q = random_prime(2^512)
```

```
N = p*q
```

```
a = p - (p % 2^86)
```



```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)

X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)

X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()

f = B[0][0]*x^2/X^2+B[0][1]*x/X+B[0][2]

sage: f.factor()[0]
(x - 2775338500016599864377709, 1)

sage: a+2775338500016599864377709 == p
True
```

Partial key recovery and finding solutions modulo divisors

Theorem (Coppersmith)

Given half the bits (most or least significant) of p , we can factor N in polynomial time.

Theorem (Howgrave-Graham)

Given degree d polynomial f , integer N , we can find roots r modulo divisors B of N satisfying

$$f(r) \equiv 0 \pmod{B}$$

for $|B| > N^\beta$, when $|r| < N^{\beta^2/d}$.

For RSA partial key recovery, we have

$$f(x) = a + x$$

and we want to find a solution vanishing modulo $p \approx N^{1/2}$.

Partial key recovery and related attacks

RSA particularly susceptible to partial key recovery attacks.

- ▶ Can factor given $1/2$ bits of p . [Coppersmith 96]
- ▶ Can factor given $1/4$ bits of d . [Boneh Durfee Frankel 98]
- ▶ Can factor given $1/2$ bits of $d \bmod (p - 1)$. [Blömer May 03]

Factoring Taiwanese keys from partial information

Previously: not clear what kind of natural attack would reveal half the bits of a factor.

Taiwanese cards: We observed RNG getting “stuck”. What if RNG becomes unstuck in least significant bits?

Exactly scenario from first example:

- ▶ A 3-dimensional lattice can let us find errors as big as $N^{1/6}$.
- ▶ Ran 3-dimensional lattice with every pattern against every key (1 hour per pattern, 164 patterns).
- ▶ Factored 39 new keys (and all but 2 of keys factored via GCD).

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q
```

```
d = random_prime(floor(N^(1/4))/2)
e = inverse_mod(d, (p-1)*(q-1))
```

d is relatively small. (But not that small.)

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

d = random_prime(floor(N^(1/4))/2)
e = inverse_mod(d, (p-1)*(q-1))

X = ceil(N^(1/4)/2); Y = ceil(N^(1/2))
M = matrix([[X*Y, -X*(N+1), -1], [0, e*X, 0], [0,0,e]])

B = M.LLL()
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

d = random_prime(floor(N^(1/4))/2)
e = inverse_mod(d, (p-1)*(q-1))

X = ceil(N^(1/4)/2); Y = ceil(N^(1/2))
M = matrix([[X*Y, -X*(N+1), -1], [0, e*X, 0], [0,0,e]])

B = M.LLL()

sage: (B[0][2]*(N+1)-B[0][1]/X)/e == d
True
```

Small RSA private exponent: Bivariate Coppersmith

Theorem (Wiener)

We can efficiently compute d when $d < N^{1/4}$.

Theorem (Boneh Durfee)

We can efficiently compute d when $d < N^{0.292}$.

The *RSA equation* is

$$\begin{aligned}ed &\equiv 1 \pmod{(p-1)(q-1)} \\ed &= 1 + k(N - (p+q) + 1)\end{aligned}$$

Boneh and Durfee use Coppersmith's method to find small solutions $x = k$, $y = (p+q)$ to

$$xy - (N+1)x - 1 \equiv 0 \pmod{e}$$

Multivariate Coppersmith

Scenario: Given multivariate polynomial $f(x_1, \dots, x_m)$ and wish to find roots

$$f(r_1, \dots, r_m) \equiv 0 \pmod{N}$$

Same approach works in this case, with some tweaks:

- ▶ To find solutions we solve a system of m equations taken from the short vectors in our lattice.
- ▶ Theorems are generally heuristic because we don't know solution set is finite.
- ▶ Results are more ad hoc in general.

Open problem: Give a useful characterization of when multivariate Coppersmith's method produces algebraically independent equations.

Factoring Taiwanese keys with bivariate Coppersmith

Returning to Taiwan example. What if RNG becomes stuck after most significant bits?

Want to find solutions to the equation

$$a + 2^t x + y \equiv 0 \pmod{p}$$

This lets us factor keys with pattern errors in most, least, or middle bits by setting t .

Ran on 20 most common patterns and factored 13 more keys.

Bivariate Coppersmith details

A mystery: The attack works *much* better in practice than theory guarantees.

Smallest lattice with guaranteed solution has dimension 10. But dimension 6 worked too.

Open problem: Why?

For finding solution, equations were not algebraically independent, but could find a solution anyway.

Open problem: Why?

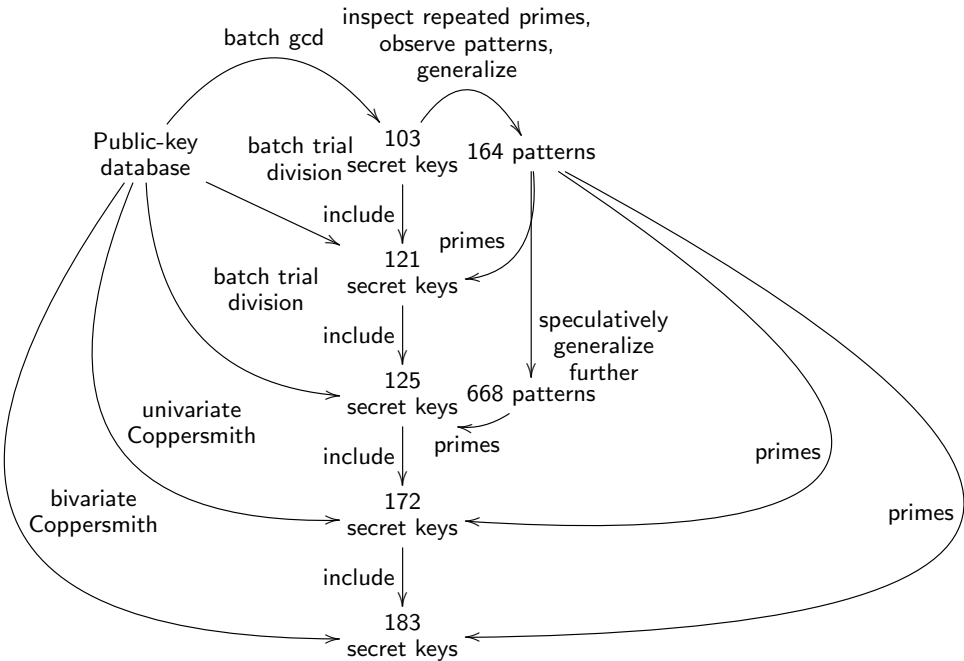
Cryptanalysis of Taiwanese keys: The rest of the story

Using the GCD algorithm in the previous lecture, we factored 103 keys.

But the GCD-compromised keys are far from the end of the story.

They're a **symptom of a much larger problem** with random number generation.

System defenders should assume that more sophisticated cryptanalysis will compromise yet more keys.



Factoring keys with Google.

[Web](#)[Images](#)[Maps](#)[Shopping](#)[More ▾](#)[Search tools](#)

About 12,400 results (0.10 second...)

[-----BEGIN RSA PRIVATE KEY ... - 1 paste tool since 2002!](#)

pastebin.com/TbaeU93m

Apr 19, 2010 – ... the difference. Copied. -----BEGIN RSA PRIVATE KEY-----.

MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrlsZc+XQYTtSiVa9R/4SAXoYpl ...

[BEGIN RSA PRIVATE KEY - Pastebin.com - 1 paste tool since 2002!](#)

pastebin.com/T8drau22

Oct 10, 2011 – create a new version of this paste RAW Paste Data. -----BEGIN RSA

PRIVATE KEY----- Hydraze did 9/11 -----END RSA PRIVATE KEY----- ...

[-----BEGIN RSA PRIVATE KEY ... - 1 paste tool since 2002!](#)

pastebin.com/BAYDB9P1

Jul 6, 2012 – -----BEGIN RSA PRIVATE KEY-----

MIIEogIBAAKCAQEA2dBZZVaV45zh99lxBRR0PKq0fMNTE8NF/

wFFHmFMB65Py/dmSYC+RIMJls ...

[-----BEGIN RSA PRIVATE KEY ... - 1 paste tool since 2002!](#)

pastebin.com/fbajUhsK

Apr 19, 2010 – rbfPgYDdmgWc/lkpMufFe/-----BEGIN RSA PRIVATE KEY-----. FUCK A

DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A ...

[-----BEGIN RSA PRIVATE KEY ... - 1 paste tool since 2002!](#)

pastebin.com/sC7bGw30

Apr 18, 2010 – ... difference. Copied. -----BEGIN RSA PRIVATE KEY-----.

MIIEogIBAAKCAQEAxvBalhzKMewLvmlr1ptID1gO7EWGFyudzOAHLqm3+0+gpPbk ...

-----BEGIN RSA PRIVATE KEY-----

MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtSlVa9R/4SAXoYpI
upNrIjkCLd6DLdqfT0429xLDmY040jzox7xiNcSMlBn8+TqTjf3TqAJmI0pgQVhJ
vW9is30teT7l2ynAyMYvGqwR0liCToMc/10ltlhPIFixw2AKUdOM5W76dwIDAQAB
AoGBAKDl8vuA9zUn2lTDddujAzBRp8ZEoJTxb7BVdLpZtgLWLuqPcXroyTkVBJC/
rbfPgYDdmGwC/lkpMufFe/-----BEGIN RSA PRIVATE KEY-----

FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK

...

FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK
Psg1RMTRceI/z3d/3BiuDjiUiRICFq0XDscCQQDFea/ocg8VVLvH/6pn7oNTqfbx
tkqCSSne3XgjAM+eA6TXbIo49d+3gsM3U1mGHR9ZBMy0068ijhIqM7/7nJtBAkEA
jmkwiP2Fy0tQ9heq4rx90ZfmixcWf/H6JldRy7kJ/qG6uDnPhH55mTRuGPpas044
7sJphlPEY8ofkwJj7K/ZKQJBAIc75HQi/Br1lRC4qPmF2vYgwpyF9RbZW056Eo7
ipgts4FLFajgogOD+JxkkT1CXtEv7MqM6ihSxGVBD6UHN7I=
-----END RSA PRIVATE KEY-----

Unfucking the duck

-----BEGIN RSA PRIVATE KEY-----

MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtS1Va9R/4SAXoYpI
upNrIjkCLd6DLdqfT0429xLDmY040jzox7xiNcSM1Bn8+TqTjf3TqAJmIOpgQVhJ
vW9is30teT7l2ynAyMYvGqwR0liCToMc/101tlhPIFixw2AKUdOM5W76dwIDAQAB
AoGBAKDl8vuA9zUn2lTDddujAzBRp8ZEoJTxw7BVdLpZtgLWLuqPcXroyTkVBJC/
rbfPgYDdmgWc/lkpMufFe/

Psg1RMTRceI/z3d/3BiuDjiUiRICFq0XDscCQQDFea/ocg8VVLvH/6pn7oNTQfbx
tkqCSSne3XgjAM+eA6TXbIo49d+3gsM3U1mGHR9ZBMy0068ijhIqM7/7nJtBAkEA
jmkwiP2FyOtQ9heq4rx90ZfmixcWf/H6JldRy7kJ/qG6uDnPvH55mTRuGPpas044
7sJphlPEY8ofkwJj7K/ZKQJBAIc75HQi/Br1lRC4qPmF2vwYgwpYF9RbZW056Eo7
ipgts4FLFajgogOD+JxkkT1CXtEv7MqM6ihSxGVBD6UHN7I=

-----END RSA PRIVATE KEY-----

Unfucking the duck

-----BEGIN RSA PRIVATE KEY-----

MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtS1Va9R/4SAXoYpI
upNrIjkCLd6DLdqfT0429xLDmY040jzox7xiNcSM1Bn8+TqTjf3TqAJmIOpgQVhJ
vW9is30teT7l2ynAyMYvGqwR0liCToMc/101tlhPIFixw2AKUdOM5W76dwIDAQAB
AoGBAKDl8vuA9zUn2lTDddujAzBRp8ZEoJTxw7BVdLpZtgLWLuqPcXroyTk vBJC/
rbfPgYDdmGwc/lkpMufFe/ <----- oh noes! ----->

Psg1RMTRceI/z3d/3BiuDjiUiRICFq0XDscCQQDFea/ocg8VVLvH/6pn7oNTQfbx
tkqCSSne3XgjAM+eA6TXbIo49d+3gsM3U1mGHR9ZBMy0068ijhIqM7/7nJtBAkEA
jmkwiP2FyOtQ9heq4rx90ZfmixcWf/H6JldRy7kJ/qG6uDnPvH55mTRuGPpas044
7sJphlPEY8ofkwJj7K/ZKQJBAlc75HQi/Br1lRC4qPmF2vwYgwpYF9RbZW056Eo7
ipgts4FLFajgogOD+JxkkT1CXtEv7MqM6ihSxGVBD6UHN7I=

-----END RSA PRIVATE KEY-----

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  
}
```

Unfucking the duck

-----BEGIN RSA PRIVATE KEY-----

MIICXwIBAAKBpenis1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtSlVa9R/4SAXoYpI
upNrIjkCLd6DLDqfT0429xLDmY040jzox7xiNcSMlBn8+TqTjf3TqAJmI0pgQVhJ
vW9is30teT7l2ynAyMYvGqwR0liCToMc/101t1hPIFixw2AKUdOM5W76dwIDAQAB
AoGBAKDl8vuA9zUn2lTDddujAzBRp8ZEoJTxxw7BVdLpZtgLWLuqPcXroyTkVBJC/
rbfPgYDdmgWc/lkpMufFe/

N

$\downarrow$

e

d

$d \bmod p - 1$

q

Psg1RMTRceI/z3d/3BiuDjiUiRICFq0XDscCQQDFea/ocg8VVLvH/6pn7oNTQfbx
tkqCSSne3XgjAM+eA6TXbIo49d+3gsM3UlmGHR9ZBMy0068ijhIqM7/7nJtBAkEA
jmkwiP2FyOtQ9heq4rx90ZfmixcWf/H6JldRy7kJ/qG6uDnPvH55mTRuGPPas044
7sJphlPEY8ofkwJj7K/ZKQJBAlc75HQi/Br1lRC4qPmF2vwYgwpYF9RbZW056Eo7
ipgts4FLFajgogOD+JxkkT1CXtEv7MqM6ihSxGVBD6UHN7I=

$q^{-1} \bmod p$

$d \bmod q - 1$

-----END RSA PRIVATE KEY-----

Easy-to-compute relations between private key fields

```
q = gcd(int(pow(2,e*dq-1,n))-1,n)
```

```
p = n/q
```

```
d = inverse_mod(e,(p-1)*(q-1))
```

```
...
```

Incomplete portions of a single piece of the key?

We just learned how to do this!

Huzzah!

-----BEGIN RSA PRIVATE KEY-----

MIICXwIBAAKBgQDET1ePqHkVN9IKaGBESjV6zBrIsZc+XQYTtS1Va9R/4SAXoYpI
upNrIjkCLd6DLdqfT0429xLDmY040jzox7xiNcSMlBn8+TqTjf3TqAJmIOpgQVhJ
vW9is30teT7l2ynAyMYvGqwR0liCToMc/l0ltlhPIFixw2AKUdOM5W76dwIDAQAB
AoGBAKDl8vuA9zUn2lTDddujAzBRp8ZEoJTxw7BVdLpZtgLWLuqPcXroyTkVBJC/
rbfPgYDdmgWc/lkpMufFe/TC+KgIDlWo50Pm/cwcCham9nEINbFF1dqaA5gVxv6g
yUWQNKVKerToh/L3OpbiApArfB2aiimXUDH0eiGev6i6h0ShAkEA/MCm4KwarMP9
gPy2V/9qlJ1mEgZXMjHG4nWBfgPQE+9Lq1+e6kMePpuFGAC5ZJC8an4PC0LU5QIV
XBUW2uLGOQJBAMBvClSWms3l1VT5IjKFNldz0ShSu0Fh5UzRpMkxtEGYs05VKn4
Psg1RMTRceI/z3d/3BiuDjiUiIRICfQ0XDscCQQDFea/ocg8VVLvH/6pn7oNTQfbx
tkqCSSne3XgjAM+eA6TXbIo49d+3gsM3U1mGHR9ZBMy0068ijhIqM7/7nJtBAkEA
jmkwiP2FyOtQ9heq4rx90ZfmixcWf/H6JldRy7kJ/qG6uDnPvH55mTRuGPpas044
7sJphlPEY8ofkwJj7K/ZKQJBAlc75HQi/Br1lRC4qPmF2vwYgwpYF9RbZW056Eo7
ipgts4FLFajgogOD+JxkkT1CXtEv7MqM6ihSxGVBD6UHN7I=

-----END RSA PRIVATE KEY-----



Lessons

- ▶ Keep your private keys private.
- ▶ Pastebin is not a secure cloud store.
- ▶ Probably shouldn't put your private key in a "secure" cloud store anyway.
- ▶ "FUCK A DUCK" is not good crypto.

Conclusions

Widely used crypto can fail in interesting ways that affect real-world security.

Entropy generation is a hard problem and failures can be masked in cryptographic output.

Complex systems can experience cascading failures that make otherwise good crypto trivial to break.

We've seen several examples of these failures that weren't detected until we beat some public keys over the head with math.