Factorization, based on FactHacks talk with

Daniel J. Bernstein
University of Illinois at Chicago
Technische Universiteit Eindhoven

Nadia Heninger
now at University of Pennsylvania

http://facthacks.cr.yp.to

Danger: Bad random-number generators

Can the attacker get lucky and *guess* your p?

"Nvo. There are $>2^{502}$ primes between 2^{511} and 2^{512}.
Each guess has chance $<2^{-501}$ of matching *your* p or q."

Danger: Bad random-number generators

Can the attacker get lucky and *guess* your p?

"Nvo. There are $>2^{502}$ primes between 2^{511} and 2^{512}.
Each guess has chance $<2^{-501}$ of matching *your* p or q."

What if your system's random-number generator is busted?
What if it's generating only 2^{40} different choices for p?

The hard attack

Download a target user's public key $N = pq$.

Buy a bunch of devices.
Try different software configurations.
Generate billions of *private* keys.
Check whether any of these primes divide the target N.

Does anyone screw up random-number generation so badly?

The hard attack

Download a target user's public key $N = pq$.

Buy a bunch of devices.
Try different software configurations.
Generate billions of *private* keys.
Check whether any of these primes divide the target N.

Does anyone screw up random-number generation so badly?
Yes!

The hard attack

Download a target user's public key $N = pq$.

Buy a bunch of devices.
Try different software configurations.
Generate billions of *private* keys.
Check whether any of these primes divide the target N.

Does anyone screw up random-number generation so badly? Yes!

1995 Goldberg–Wagner: During any particular second,
the Netscape browser generates only $\approx 2^{47}$ possible keys.

The hard attack

Download a target user's public key $N = pq$.

Buy a bunch of devices.
Try different software configurations.
Generate billions of *private* keys.
Check whether any of these primes divide the target N.

Does anyone screw up random-number generation so badly? Yes!

1995 Goldberg–Wagner: During any particular second,
the Netscape browser generates only $\approx 2^{47}$ possible keys.

2008 Bello: Since 2006, Debian and Ubuntu
are generating $< 2^{20}$ possible keys for SSH, OpenVPN, etc.

The easy attack

Download *two* target public keys N_1, N_2.
Hope that they share a prime p: i.e., $N_1 = pq_1$, $N_2 = pq_2$.
Not a surprise if $N_1 = N_2$, but what if $N_1 \neq N_2$?

The easy attack

Download *two* target public keys N_1, N_2.
Hope that they share a prime p: i.e., $N_1 = pq_1$, $N_2 = pq_2$.
Not a surprise if $N_1 = N_2$, but what if $N_1 \neq N_2$?

Euclid's algorithm prints the shared prime p given N_1, N_2.

```
def greatestcommondivisor(n1,n2):
  # Euclid's algorithm
  while n1 != 0: n1,n2 = n2%n1,n1
  return abs(n2)
```

Built into Sage as `gcd`.

A small example of Euclid's algorithm

```
sage: n1,n2 = 4187,5989
sage: n1,n2 = n2%n1,n1; print n1
1802
sage: n1,n2 = n2%n1,n1; print n1
583
sage: n1,n2 = n2%n1,n1; print n1
53
sage: n1,n2 = n2%n1,n1; print n1
0
sage: 4187/53  # / is exact division, as in Python 3
79
sage: 5989/53  # use // if you want rounding division
113
```

So $\gcd\{4187, 5989\} = 53$ and $4187 = 53 \cdot 79$ and $5989 = 53 \cdot 113$.

Euclid's algorithm is very fast

```
sage: p = random_prime(2^512)
sage: q1 = random_prime(2^512)
sage: q2 = random_prime(2^512)
sage: time g = gcd(p*q1,p*q2)
Time: CPU 0.00 s, Wall: 0.00 s
sage: g == p
True
```

Finding shared factors of many inputs

Download millions of public keys $N_1, N_2, N_3, N_4, \ldots$.
There are **millions of millions** of pairs to try:
(N_1, N_2); (N_1, N_3); (N_2, N_3); (N_1, N_4); (N_2, N_4); etc.

Finding shared factors of many inputs

Download millions of public keys $N_1, N_2, N_3, N_4, \ldots$.
There are **millions of millions** of pairs to try:
(N_1, N_2); (N_1, N_3); (N_2, N_3); (N_1, N_4); (N_2, N_4); etc.

That's feasible; but **batch gcd** finds the shared primes much faster.

Our real goal is to compute

$\gcd\{N_1, N_2 N_3 N_4 \cdots\}$ (this gcd is > 1 if N_1 shares a prime);
$\gcd\{N_2, N_1 N_3 N_4 \cdots\}$ (this gcd is > 1 if N_2 shares a prime);
$\gcd\{N_3, N_1 N_2 N_4 \cdots\}$ (this gcd is > 1 if N_3 shares a prime);
etc.

Batch gcd, part 1: product tree

First step: Multiply all the keys! Compute $R = N_1 N_2 N_3 \cdots$.

```
def producttree(X):
  result = [X]
  while len(X) > 1:
    X = [prod(X[i*2:(i+1)*2])
         for i in range((len(X)+1)/2)]
    result.append(X)
  return result

# for example:
print producttree([10,20,30,40])
# output is [[10, 20, 30, 40], [200, 1200], [240000]]
```

Batch gcd, part 2: remainder tree

Reduce $R = N_1 N_2 N_3 \cdots$ modulo N_1^2 and N_2^2 and N_3^2 and so on.

Obtain $\gcd\{N_1, N_2 N_3 \cdots\}$ as $\gcd\{N_1, (R \bmod N_1^2)/N_1\}$;
obtain $\gcd\{N_2, N_1 N_3 \cdots\}$ as $\gcd\{N_2, (R \bmod N_2^2)/N_2\}$;
etc.

```
def batchgcd(X):
  prods = producttree(X)
  R = prods.pop()
  while prods:
    X = prods.pop()
    R = [R[floor(i/2)] % X[i]**2 for i in range(len(X))]
  return [gcd(r/n,n) for r,n in zip(R,X)]
```

Batch gcd is very fast

```
sage: # two-year-old laptop, clocked down to 800MHz
sage: def myrand():
....:       return Integer(randrange(2^1024))
....:
sage: time g = batchgcd([myrand() for i in range(100)])
Time: CPU 0.05 s, Wall: 0.05 s
sage: time g = batchgcd([myrand() for i in range(1000)])
Time: CPU 1.08 s, Wall: 1.08 s
sage: time g = batchgcd([myrand() for i in range(10000)])
Time: CPU 23.21 s, Wall: 23.29 s
sage:
```

Are random-number generators really this bad?

Are random-number generators really this bad?

2012 Heninger–Durumeric–Wustrow–Halderman,
best-paper award at USENIX Security Symposium:

Factored tens of thousands of public keys on the Internet
. . . typically keys for your home router, not for your bank.
Why? **Many deployed devices are generating guessable p's.**
Most common problem: horrifyingly bad interactions between
OpenSSL key generation, `/dev/urandom` seeding, entropy sources.

http://factorable.net

Are random-number generators really this bad?

2012 Heninger–Durumeric–Wustrow–Halderman,
best-paper award at USENIX Security Symposium:

Factored tens of thousands of public keys on the Internet
. . . typically keys for your home router, not for your bank.
Why? **Many deployed devices are generating guessable p's.**
Most common problem: horrifyingly bad interactions between
OpenSSL key generation, `/dev/urandom` seeding, entropy sources.

http://factorable.net

2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter,
independent "Ron was wrong, Whit is right" paper, Crypto:

Factored tens of thousands of public keys on the Internet.
Dunno why, but OMG! Insecure e-commerce! Call the NYTimes!

Factoring RSA keys from certified smart cards: Coppersmith in the wild

Daniel J. Bernstein, Yun-An Chang,
Chen-Mou Cheng, Li-Ping Chou,
Nadia Heninger, Tanja Lange,
Nicko van Someren

Nice followup student projects in data mining

1. Download all certificates of type X; extract RSA keys.
2. Check for common factors.
3. Write report that you've done the work and there are none.

Nice followup student projects in data mining

1. Download all certificates of type X; extract RSA keys.
2. Check for common factors.
3. Write report that you've done the work and there are none.

This started as such a student project on a very nice system: MOICA: Certificate Authoritiy of MOI (Ministry of the Interior). In Taiwan all citizens can get a smartcard with signing and encryption ability to

- file personal income taxes,
- update car registration,
- make transactions with government agencies (property registries, national labor insurance, public safety, and immigration),
- file grant applications,
- interact with companies (e.g. Chunghwa Telecom).

Taiwan Citizen Digital Certificate

- Smart cards are issued by the government.
- FIPS-140 and Common Criteria Level 4+ certified.
- RSA keys are generated on card.
- About 3,002,000 certificates (all using RSA keys) stored on national LDAP directory. This is publicly accessible to enable citizen-to-citizen and citizen-to-commerce interactions.

Certificate of Chen-Mou Cheng

```
Data: Version: 3 (0x2)
Serial Number: d7:15:33:8e:79:a7:02:11:7d:4f:25:b5:47:e8:ad:38
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=TW, O=XXX
Validity
     Not Before: Feb 24 03:20:49 2012 GMT
     Not After : Feb 24 03:20:49 2017 GMT
Subject: C=TW, CN=YYY serialNumber=0000000112831644
Subject Public Key Info:
     Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit) Modulus:
     00:bf:e7:7c:28:1d:c8:78:a7:13:1f:cd:2b:f7:63:
     2c:89:0a:74:ab:62:c9:1d:7c:62:eb:e8:fc:51:89:
     b3:45:0e:a4:fa:b6:06:de:b3:24:c0:da:43:44:16:
     e5:21:cd:20:f0:58:34:2a:12:f9:89:62:75:e0:55:
     8c:6f:2b:0f:44:c2:06:6c:4c:93:cc:6f:98:e4:4e:
     3a:79:d9:91:87:45:cd:85:8c:33:7f:51:83:39:a6:
     9a:60:98:e5:4a:85:c1:d1:27:bb:1e:b2:b4:e3:86:
     a3:21:cc:4c:36:08:96:90:cb:f4:7e:01:12:16:25:
     90:f2:4d:e4:11:7d:13:17:44:cb:3e:49:4a:f8:a9:
     a0:72:fc:4a:58:0b:66:a0:27:e0:84:eb:3e:f3:5d:
     5f:b4:86:1e:d2:42:a3:0e:96:7c:75:43:6a:34:3d:
     6b:96:4d:ca:f0:de:f2:bf:5c:ac:f6:41:f5:e5:bc:
     fc:95:ee:b1:f9:c1:a8:6c:82:3a:dd:60:ba:24:a1:
     eb:32:54:f7:20:51:e7:c0:95:c2:ed:56:c8:03:31:
     96:c1:b6:6f:b7:4e:c4:18:8f:50:6a:86:1b:a5:99:
     d9:3f:ad:41:00:d4:2b:e4:e7:39:08:55:7a:ff:08:
     30:9e:df:9d:65:e5:0d:13:5c:8d:a6:f8:82:0c:61:
     c8:6b
Exponent: 65537 (0x10001)

        .
        .
        .
```

This project took a slightly different turn

HITCON 2012 (July 20–21):
Prof. Li-Ping Chou presents “Cryptanalysis in real life”
(based on work with Yun-An Chang and Chen-Mou Cheng)

Factored 103 Taiwan Citizen Digital Certificates
(out of 2.26 million keys with 1024 bits).

This project took a slightly different turn

HITCON 2012 (July 20–21):
Prof. Li-Ping Chou presents "Cryptanalysis in real life"
(based on work with Yun-An Chang and Chen-Mou Cheng)

Factored 103 Taiwan Citizen Digital Certificates
(out of 2.26 million keys with 1024 bits).

Wrote report that some keys are factored, informed MOI.

This project took a slightly different turn

HITCON 2012 (July 20–21):
Prof. Li-Ping Chou presents “Cryptanalysis in real life”
(based on work with Yun-An Chang and Chen-Mou Cheng)

Factored 103 Taiwan Citizen Digital Certificates
(out of 2.26 million keys with 1024 bits).

Wrote report that some keys are factored, informed MOI.

End of story.

This project took a slightly different turn

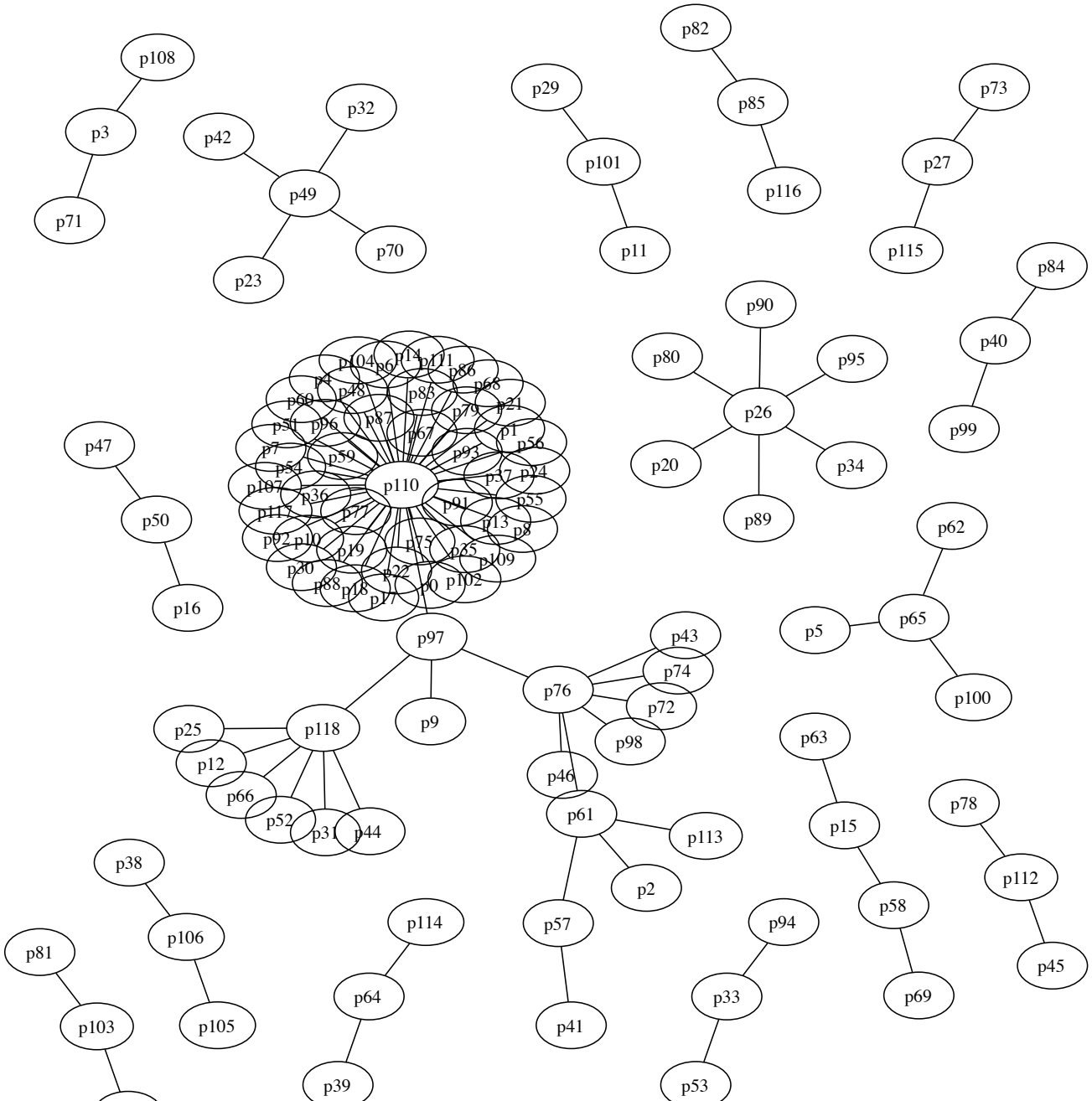
HITCON 2012 (July 20–21):
Prof. Li-Ping Chou presents "Cryptanalysis in real life"
(based on work with Yun-An Chang and Chen-Mou Cheng)

Factored 103 Taiwan Citizen Digital Certificates
(out of 2.26 million keys with 1024 bits).

Wrote report that some keys are factored, informed MOI.

End of story?

January 2013: Closer look at the 119 primes



http://facthacks.cr.yp.to

Look at the primes!

Prime factor p110 appears 46 times

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
00000000000000000000000000002f9
```

Look at the primes!

Prime factor p110 appears 46 times

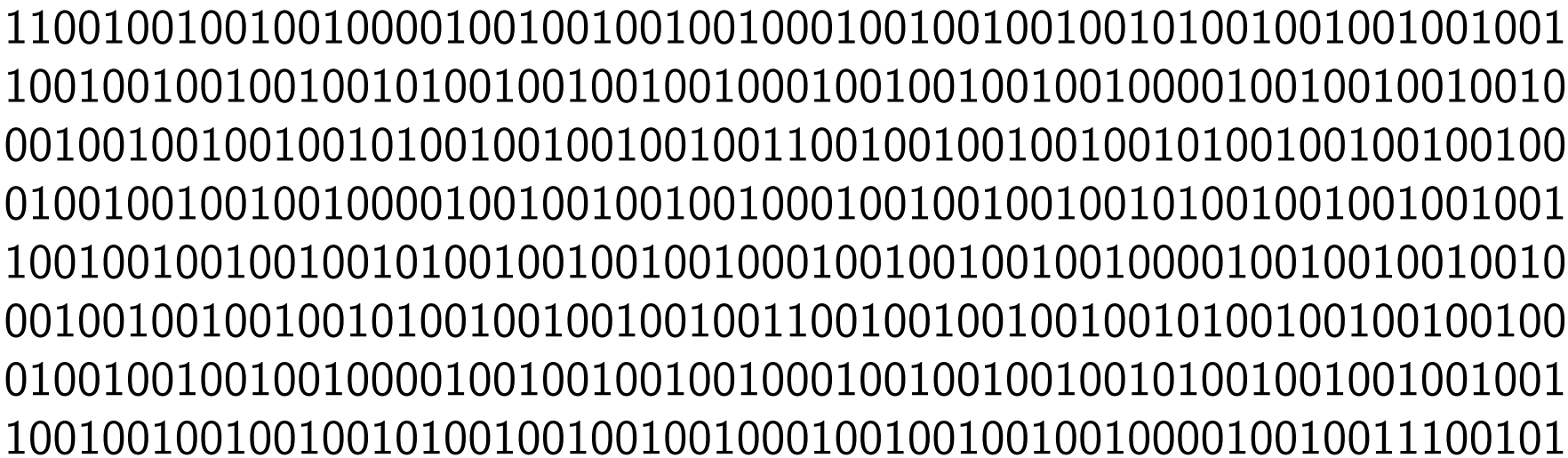
c00000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
000000000000000000000000002f9

which is the next prime after $2^{511} + 2^{510}$.
The next most common factor, repeated 7 times, is

c92424922492924992494924492424922
49292499249492449242492249292499
24949244924249224929249924949249
24924922492924992494924492424e5

Several other factors exhibit such a pattern.

How is this pattern generated?



How is this pattern generated?

Swap every 16 bits in a 32 bit word

0010010010010010 1100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010010010010 0100100100100100
1001001001001001 0010010010010010 0100100100100100 1001001001001001
0010010010010010 0100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010010010010 0100100100100100
1001001001001001 0010010010010010 0100100100100100 1001001001001001
0010010010010010 0100100100100100 1001001001001001 0010010010010010
0100100100100100 1001001001001001 0010010011100101 0100100100100100

How is this pattern generated?

Realign

00100100100100101100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100110010101001001001001001001001001001

How is this pattern generated?

Realign

00100100100100101100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100100100100100100100100100100100100100
00100100100100100100100100110010101001001001001001001

The 119 factors had patterns of period 1,3,5, and 7.

Prime generation

1. Choose a bit pattern of length 1, 3, 5, or 7 bits, repeat it to cover more than 512 bits, and truncate to exactly 512 bits.
2. For every 32-bit word, swap the lower and upper 16 bits.
3. Fix the most significant two bits to 11.
4. Find the next prime greater than or equal to this number.

Factoring by trial division

1. Choose a bit pattern of length 1, 3, 5, or 7 bits, repeat it to cover more than 512 bits, and truncate to exactly 512 bits.
2. For every 32-bit word, swap the lower and upper 16 bits.
3. Fix the most significant two bits to 11.
4. Find the next prime greater than or equal to this number.

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...

Factoring by trial division

1. Choose a bit pattern of length 1, 3, 5, or 7 bits, repeat it to cover more than 512 bits, and truncate to exactly 512 bits.
2. For every 32-bit word, swap the lower and upper 16 bits.
3. Fix the most significant two bits to 11.
4. Find the next prime greater than or equal to this number.

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.

Factoring by trial division

1. Choose a bit pattern of length 1, 3, 5, or 7 bits, repeat it to cover more than 512 bits, and truncate to exactly 512 bits.
2. For every 32-bit word, swap the lower and upper 16 bits.
3. Fix the most significant two bits to 11.
4. Find the next prime greater than or equal to this number.

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.
Breaking RSA-1024 by "trial division".

Factoring by trial division

1. Choose a bit pattern of length 1, 3, 5, or 7 bits, repeat it to cover more than 512 bits, and truncate to exactly 512 bits.
2. For every 32-bit word, swap the lower and upper 16 bits.
3. Fix the most significant two bits to 11.
4. Find the next prime greater than or equal to this number.

Do this for any pattern:
0,1,001,010,011,100,101,110
00001,00010,00011,00100,00101,0011,00111,01000,01001,01010,...
00000001,0000011,0000101,0000111,0001001,...
Computing GCDs factored 105 moduli, of which 18 were new.
Breaking RSA-1024 by "trial division".
Factored 4 more keys using patterns of length 9.

Patterns do not find all factors

These primes

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
00000000000000000000000002030b

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
00000000000000000000100000177

were found via GCDs, but not from the patterns.

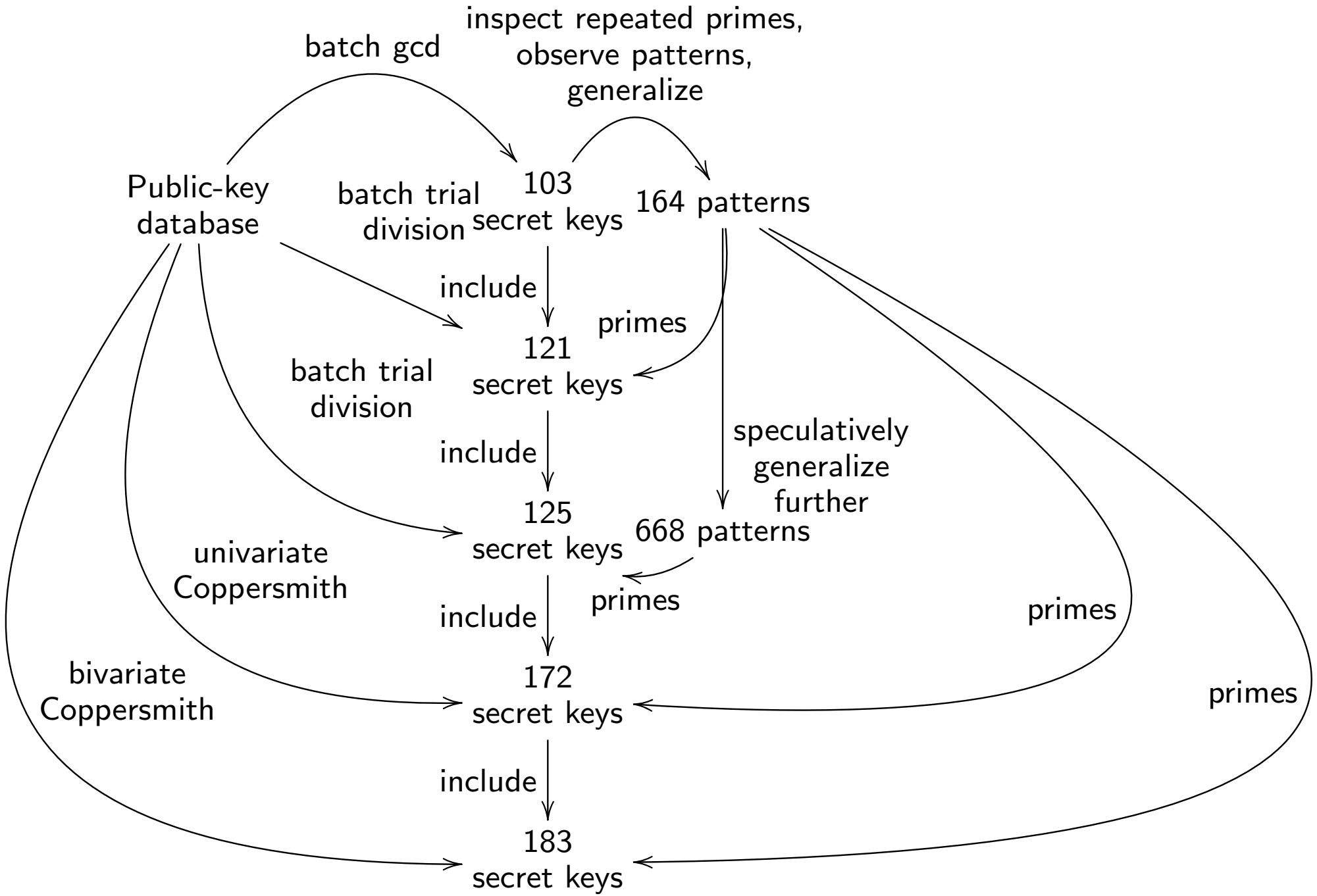
Patterns do not find all factors

These primes

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
000000000000000000000000002030b
```

```
c0000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000100000177
```

were found via GCDs, but not from the patterns.
Looks like base pattern 0 with some bits flipped.
You'll learn about Coppersmith's method in the master's level.


batch gcd
inspect repeated primes,
observe patterns,
generalize
Public-key
database
batch trial
division
103
secret keys
164 patterns
include
primes
121
secret keys
batch trial
division
include
speculatively
generalize
further
125
secret keys
668 patterns
univariate
Coppersmith
primes
include
primes
bivariate
Coppersmith
172
secret keys
primes
include
183
secret keys

Back to more general factoring
Let c be a random number
(not RSA modulus)
that we want to factor

The rho method

Define $\rho_0 = 0$, $\rho_{k+1} = \rho_k^2 + 11$.

Every prime $\leq 2^{20}$ divides

$$S = (\rho_1 - \rho_2)(\rho_2 - \rho_4)(\rho_3 - \rho_6) \cdots (\rho_{3575} - \rho_{7150}).$$

Also many larger primes.

Can compute $\gcd(c, S)$ using $\approx 2^{14}$ multiplications mod c, very little memory.

Compare to $\approx 2^{16}$ divisions for trial division up to 2^{20}.

More generally: Choose z. Compute $\gcd(c, S)$ where

$$S = (\rho_1 - \rho_2)(\rho_2 - \rho_4) \cdots (\rho_z - \rho_{2z}).$$

How big does z have to be for all primes $\leq y$ to divide S?

Plausible conjecture: $y^{1/2+o(1)}$; so $y^{1/2+o(1)}$ mults mod c.

Reason: Consider first collision in ρ_1 mod p, ρ_2 mod $p, \ldots$.

If $\rho_i \bmod p = \rho_j \bmod p$ then $\rho_k \bmod p = \rho_{2k} \bmod p$ for $k \in (j-i)\mathbf{Z} \cap [\mathbf{i}, \infty] \cap [\mathbf{j}, \infty]$.

The $p - 1$ method

$S_1 = 2^{232792560} - 1$ has prime divisors
3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 53, 61, 67, 71, 73, 79, 89, 97, 103, 109, 113, 127, 131, 137, 151, 157, 181, 191, 199
etc.

These divisors include
70 of the 168 primes $\leq 10^3$;
156 of the 1229 primes $\leq 10^4$;
296 of the 9592 primes $\leq 10^5$;
470 of the 78498 primes $\leq 10^6$;
etc.

An odd prime p divides $2^{232792560} - 1$ iff order of 2 in the multiplicative group $\mathbf{F}_p^*$ divides $s = 232792560$.

Many ways for this to happen:
232792560 has 960 divisors.

Why so many?

An odd prime p divides $2^{232792560} - 1$ iff order of 2 in the multiplicative group $\mathbf{F}_p^*$ divides $s = 232792560$.

Many ways for this to happen:
232792560 has 960 divisors.

Why so many?
Answer:

$$\begin{aligned} s &= 232792560 = \operatorname{lcm}(1, 2, 3, 4, 5, \ldots, 20) \\ &= 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19. \end{aligned}$$

Given positive integer n, can compute $2^{232792560} - 1 \bmod n$ using 41 operations in $\mathbf{Z}/\mathbf{n}$.
Notation: $a \bmod b = a - b\lfloor a/b \rfloor$.

e.g. $n = 8597231219$: ...
$2^{27} \bmod n = 134217728$;
$2^{54} \bmod n = 134217728^2 \bmod n$
$= 935663516$;
$2^{55} \bmod n = 1871327032$;
$2^{110} \bmod n = 1871327032^2 \bmod n$
$= 1458876811$; ...;
$2^{232792560} - 1 \bmod n = 5626089344$.

Given positive integer n, can compute $2^{232792560} - 1 \bmod n$ using 41 operations in $\mathbf{Z}/\mathbf{n}$.
Notation: $a \bmod b = a - b\lfloor a/b \rfloor$.

e.g. $n = 8597231219$: ...
$2^{27} \bmod n = 134217728$;
$2^{54} \bmod n = 134217728^2 \bmod n$
$= 935663516$;
$2^{55} \bmod n = 1871327032$;
$2^{110} \bmod n = 1871327032^2 \bmod n$
$= 1458876811$; ...;
$2^{232792560} - 1 \bmod n = 5626089344$.

Easy extra computation (Euclid):
$\gcd 5626089344, n = 991$.