RSA IX

Bad randomness

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see recent GitKraken bug.

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see recent GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.

Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see recent GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.
- Follow-up projects

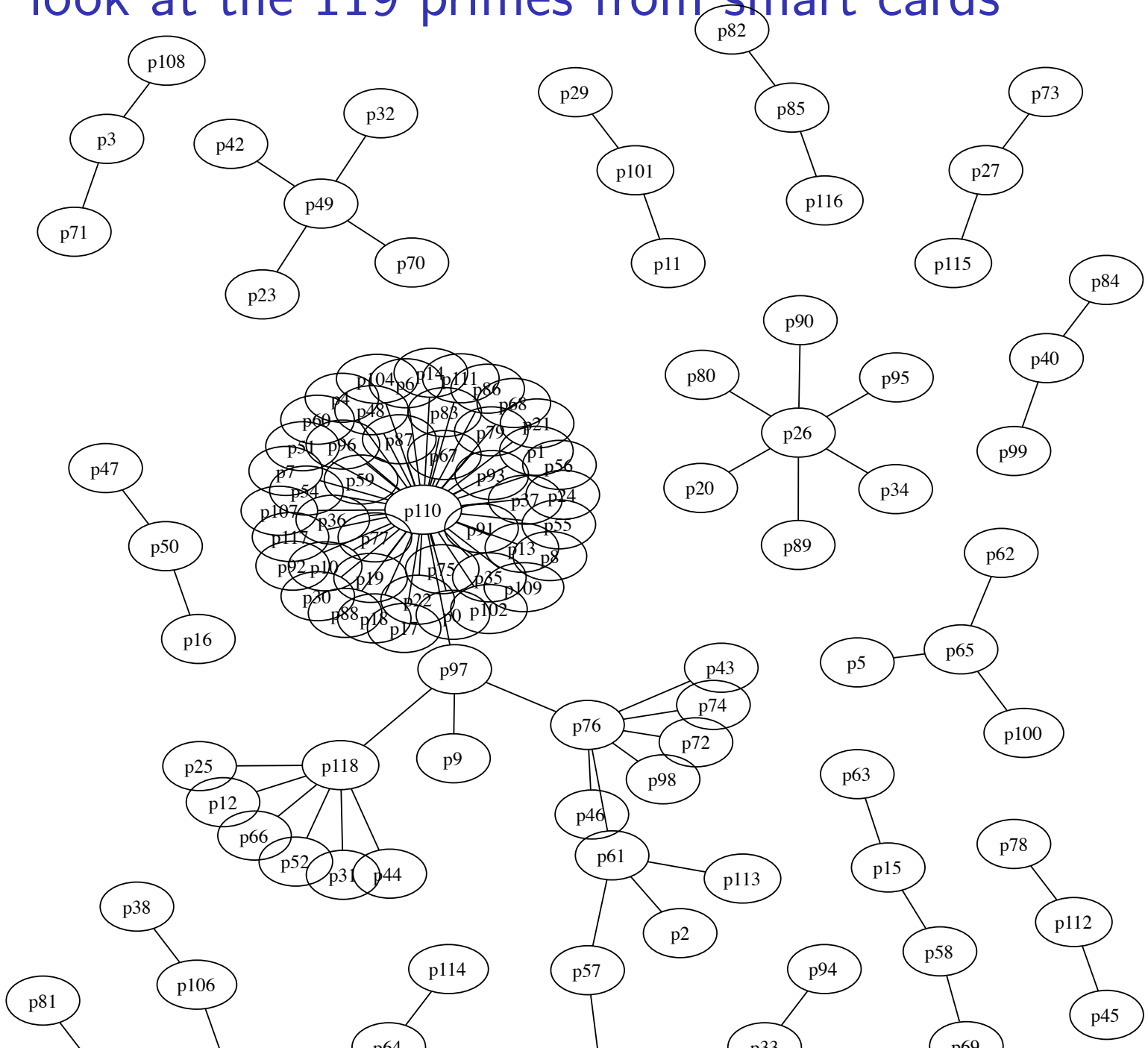
Problems with non-randomness

- 2004 Bauer–Laurie: checked 18000 PGP RSA keys; found 2 keys sharing a factor.
- 2012 Lenstra–Hughes–Augier–Bos–Kleinjung–Wachter and Heninger–Durumeric–Wustrow–Halderman,
  Factored tens of thousands of public keys on the Internet
  ... typically keys for your home router, not for your bank.
- Different devices produced same key or shared prime factors.
- Most common problem: horrifyingly bad interactions between OpenSSL key generation, `/dev/urandom` seeding, entropy sources.
- 2021 still problems with randomness, see recent GitKraken bug.
- These computations find q_2 in

$$p_1q_1, p_2q_2, p_3q_3, p_4q_2, p_5q_5, p_6q_6;$$

  and thus also p_2 and p_4.
  Obvious:GCD computation. Faster: scaled remainder trees.
- Follow-up projects incl shared primes from certified smart cards.

Closer look at the 119 primes from smart cards



Look at the primes!

Prime factor p110 appears 46 times

Look at the primes!

Prime factor p110 appears 46 times

```
c0000000000000000000000000000000
0000000000000000000000000000000
0000000000000000000000000000000
00000000000000000000000000002f9
```

Look at the primes!

Prime factor p110 appears 46 times

c000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
0000000000000000000000000002f9

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

c92424922492924992494924492424 92
24929249924949244924249224929249
92494924492424922492924992494924
49242492249292499249244924 24e5

Several other factors exhibit such a pattern.

How is this pattern generated?

Prime written in binary

110010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100001001001001001
001001001001001010010010010010011001001001001001010010010010010
010010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100001001001001001
001001001001001010010010010010011001001001001001010010010010010
010010010010010000100100100100100010010010010010100100100100100
100100100100100101001001001001000100100100100100001001011100101

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
00100100100100100100100100110010101001001001001001001001001001

Each of the 119 keys had at least one prime factor
with patterns of period 1,3,5, or 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.

We understand that this is not what they wanted to do.
Best guess: random number generator got stuck with some test pattern,
output was not whitened using AES.

Reply to disclosure was "human error", confirmation that TRNG output was used instead of PRNG output.

TRNG: True random-number generator.
PRNG: Pseudo random-number generator.

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.

Sneak preview: We want more keys!