Factorization of widely used RSA moduli

Vulnerable RSA generation (CVE-2017-15361, VU#307015)

Received Real-World Impact Award ACM CCS 2017
https://roca.crocs.muni.cz

Matúš Nemec, **Marek Sýs**, Petr Švenda, Dušan Klinec and Vashek Matyáš
Centre for Research on Cryptography and Security
Masaryk University, Czech Republic
syso@mail.muni.cz *@sysox3*

CROCS
Centre for Research on Cryptography and Security

Overview

- Motivation
- Structure of RSA primes in specific library of Infineon AG
  - Fast prime algorithm
- Detection of vulnerable RSA keys
- Factorization method
  - Coppersmith’s algorithm
  - Basic factorization method - infeasible
  - Optimizations – practical attack
- Attack complexity

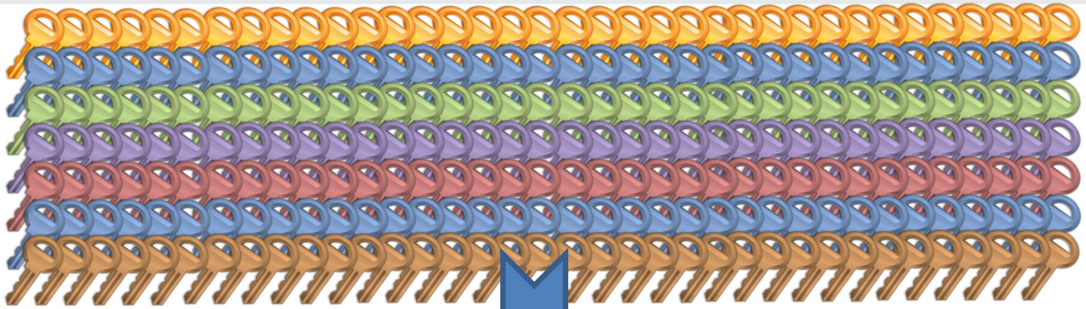
RSA primer – what does it mean and why should I care?

- RSA is widely used public-key cryptosystem (1977)
- Used for digital signatures (mail, software distribution, contracts…)
- Used for key exchange (HTTPS/TLS, PGP…)
- Private part: private exponent d, **random** primes P, Q
- Public part: public exponent e, modulus N

$$N = P \times Q$$

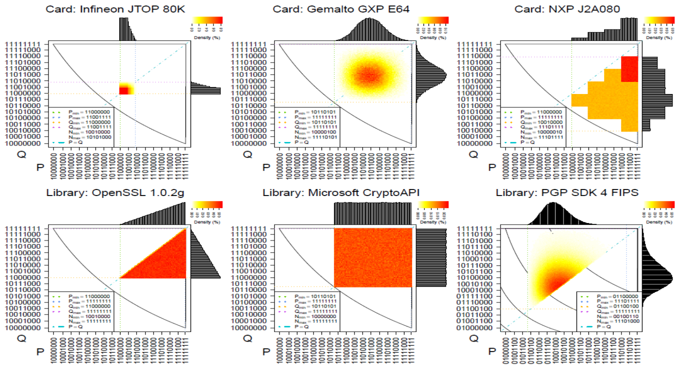
- Factorization attack: compute primes P and Q from the knowledge of N

60+ million fresh RSA keypairs

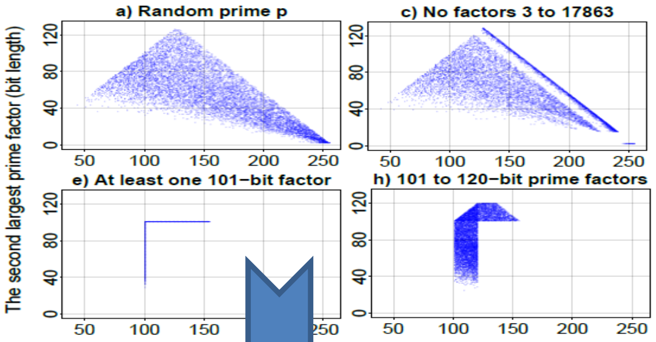


22 sw. libraries
16 smart cards

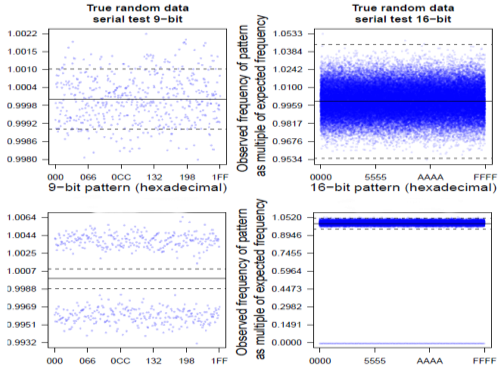
Distribution of primes (MSB)



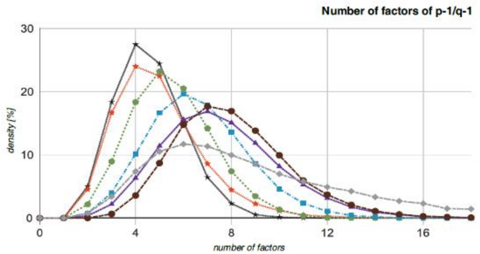
Large factors of p-1 / p+1



Bit stream statistics

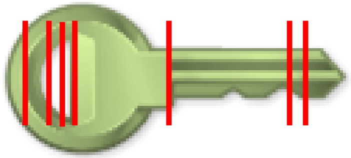


Number of factors



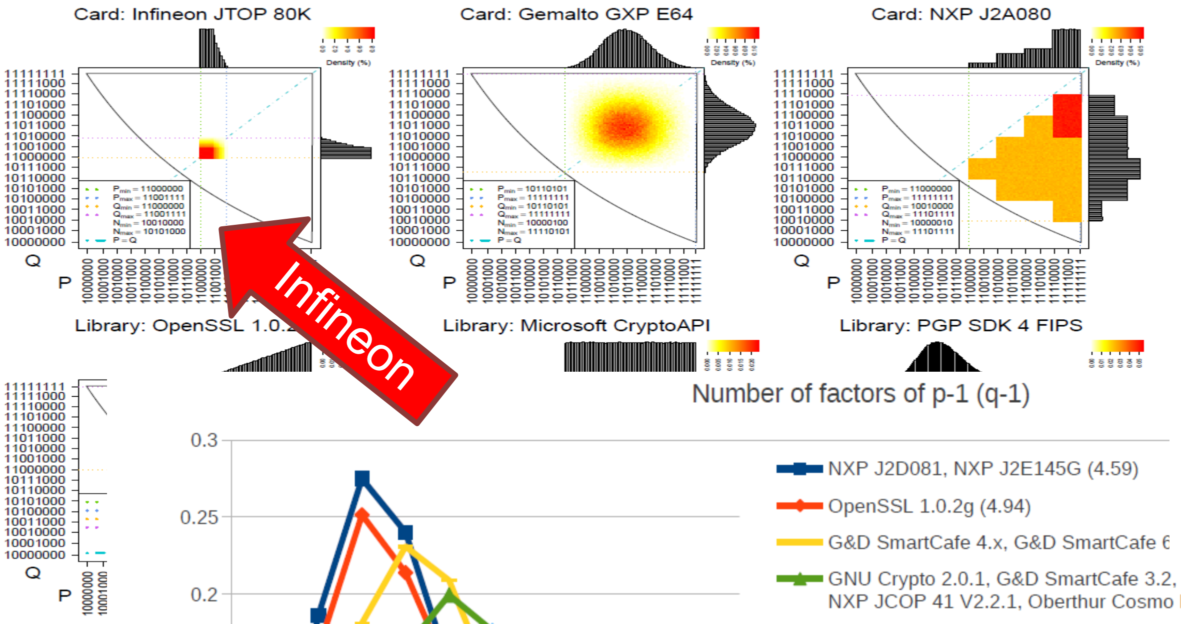
and more…

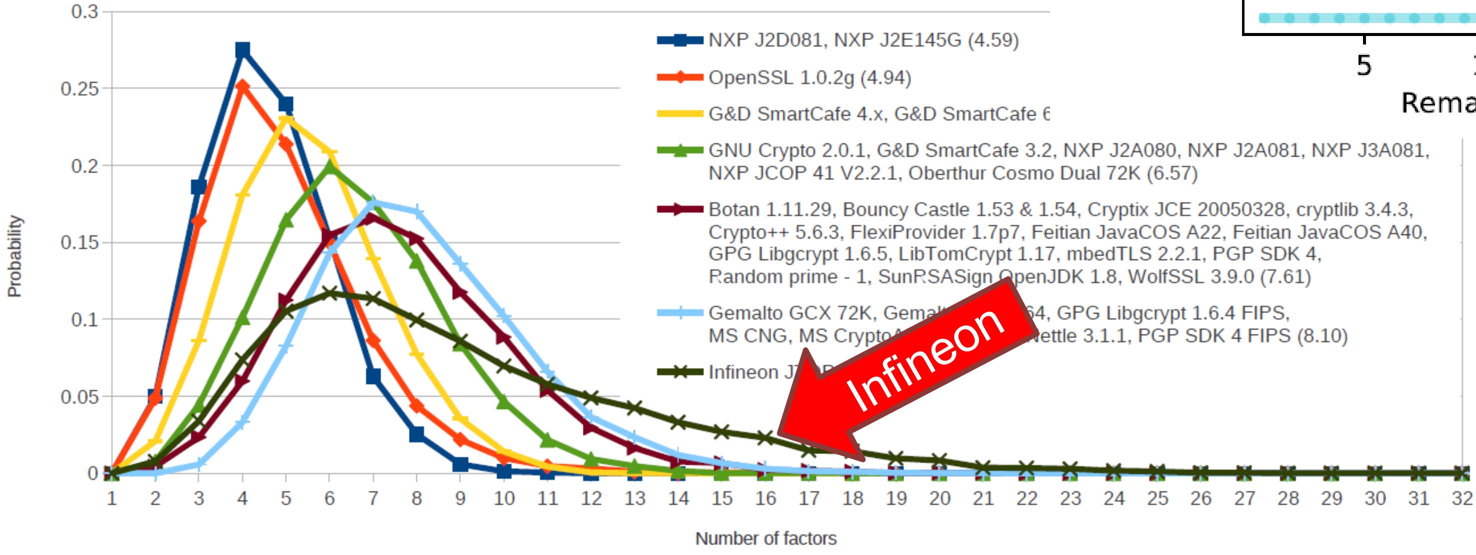
Identify library which generated given public key (USENIXSec'16)



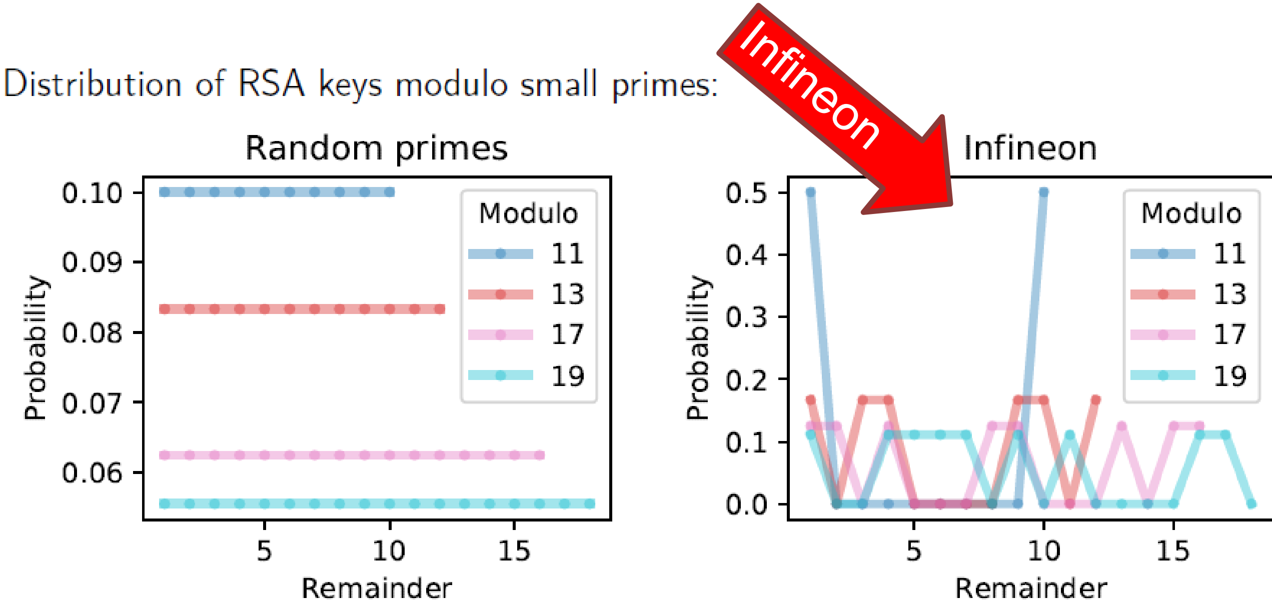
OpenSSL
Cryptography and SSL/TLS Toolkit

Biased Infineon keys





Distribution of RSA keys modulo small primes:



Motivation
Entropy loss estimation

- Findings:

  $N \bmod 7 \in \{1,2,\dots,6\}$ - OK (6 out of 6)

  $N \bmod 11 \in \{1,10\}$ - entropy loss (2 out of 10)

  $N \bmod 37 \in \{1,10,26\}$ - entropy loss (3 out of 36)

- Putting primes together:
  - $N \bmod 7*11*37 \in \{1,10,100,285,1000,1453\}$ (6 out of 2160)
  - even **greater** entropy loss – 6 instead of 6*2*3
- Further analysis:
  - $\{1,10\},\{1,10,26\}$ are **subgroups** of Z_{11}^*, Z_{37}^*,
  - Also $\{1,10,100,285,1000,1453\}$ is **subgroup** of $Z_{7.11.37}^*$

Main observation

- Generator of subgroups exists - 65537 (smallest):

$$N \equiv 65537^{c}\ mod\ 2 * 3 * 5 \ldots \quad \text{(for some } c\text{)}$$

- Same hold for primes:

$$p \equiv 65537^{a}\ mod\ 2 * 3 * 5 \ldots$$

$$q \equiv 65537^{b}\ mod\ 2 * 3 * 5 \ldots$$

- Different $M = 2 * 3 * 5 * \cdots * \boldsymbol{p_{max}}$ - related to key size
  - RSA**512** - $M = 2 * 3 * \cdots * \mathbf{167}$,
  - RSA**1024** - $M = 2 * 3 * \cdots * 167 * \cdots * \mathbf{353}$
  - $\boldsymbol{p_{max}}$= **167, 353, 701** or **1427**

Entropy loss of Infineon primes

- How many remainders $p \bmod M (\equiv 65537^a)$ of Infineon primes?
  - order $ord_M(65537)$ of generator !
- $ord_M(65537)$ - **minimal** $a\ (\neq 0)$ such that $65537^a \equiv 1 \bmod M$
  - $65537^a \equiv 1 \bmod M \Leftrightarrow$

$$\begin{array}{ll} 65537^a \equiv 1 \bmod 2 & \Rightarrow ord_2 \mid ord_M \\ 65537^a \equiv 1 \bmod 3 & \Rightarrow ord_3 \mid ord_M \\ \vdots & \vdots \\ 65537^a \equiv 1 \bmod p_{max} & \Rightarrow ord_{p_{max}} \mid ord_M \end{array}$$

- ord_M minimal multiple of $ord_2, ord_3, \ldots$

$$ord_M = lcm(ord_2, ord_3, \ldots)$$

Entropy loss of primes (Example RSA – 512)

- **Given only by structure:** $\boldsymbol{p \ mod \ M}$
- Random primes $mod\ M$ form group Z_M^*
  - size of Z_M^* = $\varphi(M) = (2-1).(3-1)\ldots(167-1)$
- Infineon primes $[65537] = 65537^a mod\ M$:
  - size of $[65537] = ord_M = lcm(ord_2, ord_3, \ldots, ord_{167})$
    - divisor of $\boldsymbol{lcm}(2-1, 3-1, \ldots, 167-1)$
- Random vs Infineon primes : product vs $\boldsymbol{lcm}$
  - 2^{62} vs 2^{216} - entropy loss 154 bits for RSA-512

Structure of Infineon primes

$$prime = k.M + 65537^{a} \bmod M, \; M = 2 * 3 * 5 * 7 \ldots .$$

- Entropy loss in prime:

Consequences:

- Strong fingerprint of RSA keys
- Practical factorization of RSA keys is possible

Why so strange structure?

Prime generation is **slow !** - primality tests (modular exponentiation)

Prime generation:

1. Random sampling – **generate** & test, **generate** & test, …
   - Many iterations – small prime factor of generated number

2. Incremental search – **generate** & test, **increment** & test, increment…
   - skip numbers with small prime factors
   - sieving methods, Joye & Pailier algorithm, “Fast Prime” algorithm (Infineon)

Fast prime (simplified)

Joye & Pailier method:

1. M – odd smooth number (M=3*5*7…)
2. Generate random k with $k * M$ of required size
3. Generate **random** $\boldsymbol{u_0} \in Z_M^*$
4. $p = k * M + u \bmod M$ (p coprime to M)
5. If p is not prime:

   $u = 2 * u \bmod M$ and go to Step 4 ($u = 2^i . \boldsymbol{u_0}$)

Infineon: M = **2***3*5*…, fixed $\boldsymbol{u_0} = \mathbf{65537}$ ($u = 65537^i . \boldsymbol{u_0}$)

Detection of vulnerable keys

- Based on public RSA moduli $N \equiv 65537^c \; mod \; M$
- Vulnerable if c exists $\Leftrightarrow$ c_i exist for **all** $p_i | M$

  i.e. $N \equiv 65537^{c_i} \; mod \; p_i$

  - small $p_i \Rightarrow$ very fast - microseconds
  - $[65537] = 65537^{c_i}$ can be pregenerated – even faster

- Errors:
  - False negatives - all Infineon primes have the specific form
  - False positives - negligible probability ($Pr < 2^{-150}$)

Factorization algorithm

$$p = k.M + 65537^{a} \; mod \; M$$

Input: $\boldsymbol{N}$

Output: $\boldsymbol{p}, \boldsymbol{q}$ (such that $N = p * q$)

1. Guess a
2. Compute k using Coppersmith's algorithm
3. **if** $p|N$ **return** $p,\ q = N/p$
   else $a = a + 1$ and go to step 1.

Perfectly parallelizable – 1000 cores ⇒ 1000x speedup

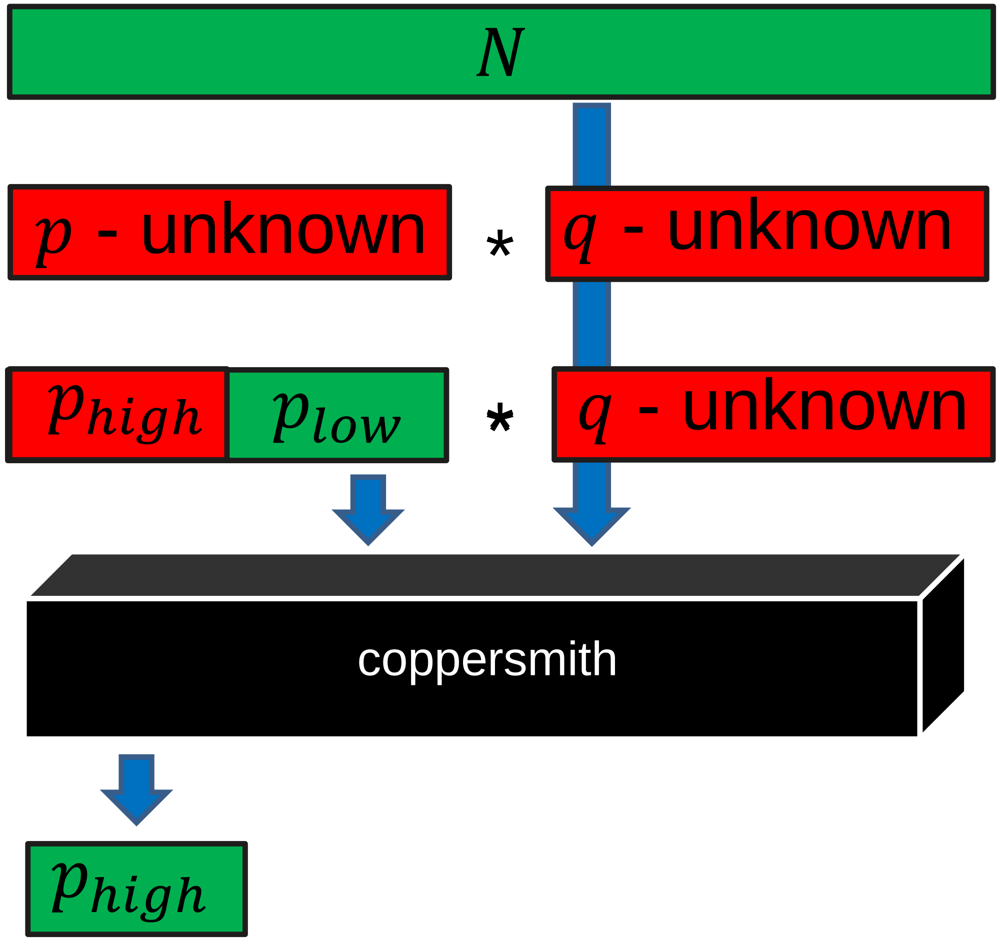
Coppersmith's attack as a black box

1. Modulus N

2. Unknown factors p, q

3. Partial knowledge of prime
   (**at least** ½ of bits of $\boldsymbol{p}$)

4. Apply Coppersmith's algorithm



Naïve algorithm

- $p = k.M + 65537^a \ mod \ M$
- Guess a

RSA -2048

256 bits

a

$k.M +$ $65537^a \ mod \ M$

54 bits

970 bits = size of M

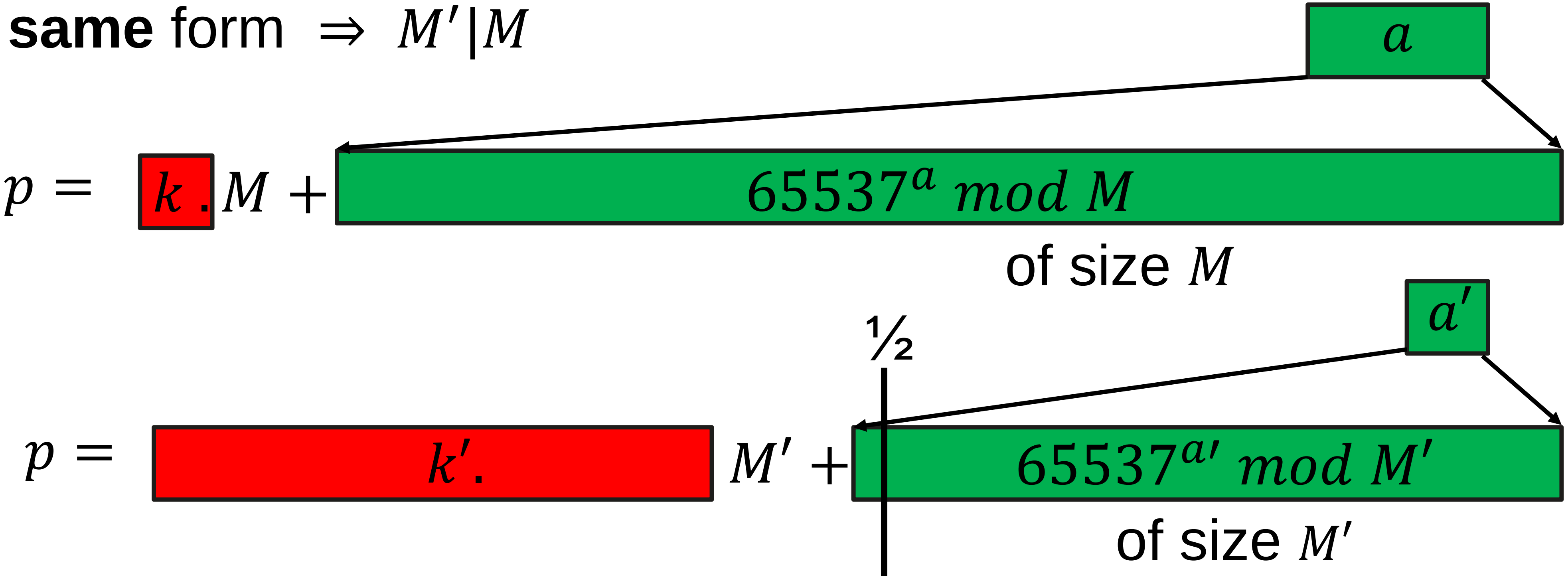
- compute k using Coppersmith's alg.
  (requires ½ of known bits – much more than that – large M)

- **Infeasible** – large a

How to make attack practical ?

Idea: ½ known (= size of M) bits of $\boldsymbol{p}$ is sufficient

- smaller M' ⇒ smaller (or equal) a'
- p of the **same** form ⇒ $M'|M$



Optimization

- Algorithm: guess a' and compute k' $(p = k'.M' + 65537^{a'} \; mod \; M')$

- Minimize number of guesses: $ord_{M'}(65537) - 1$

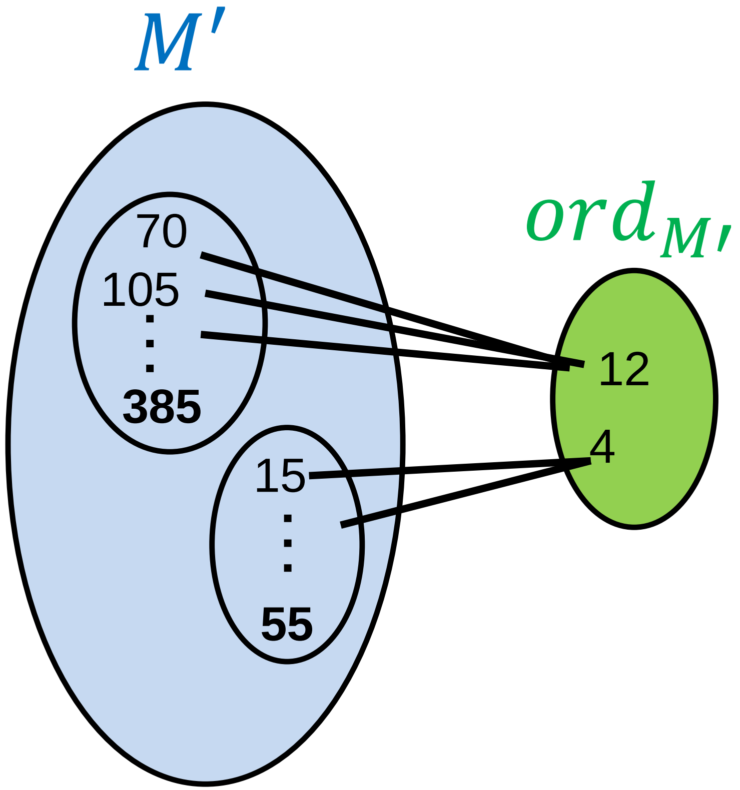
- **One** search for M':
  - $M' | M$ - same structure
  - size of $M' > ½$ size of p - required by Coppersmith's alg.
  - with **minimal** $ord_{M'}(65537)$ - minimal number of guesses

Optimize M
Search space

- Looking for $M' | M$ with:
  - $\boldsymbol{size}(\boldsymbol{M}') > \frac{size(p)}{2}$ and minimal $ord_{M'}$

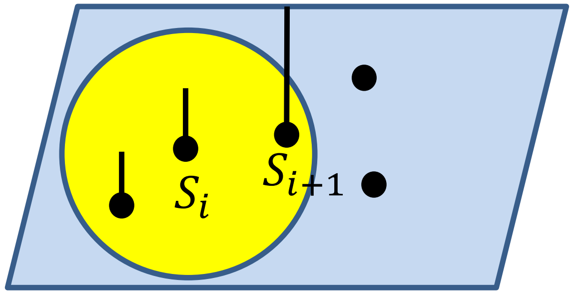
1. divisors $M' \mid M$ – large space
   – brute force infeasible ($0.5 * 10^{12}$ for RSA-512)

2. divisors $ord_{M'}$ of ord_M – smaller space
   – $ord_{M'} \rightarrow M'$ (**maximal**)
   – small keys (e.g. 38400 for RSA-512) – brute force feasible
   – larger keys - **greedy strategy**

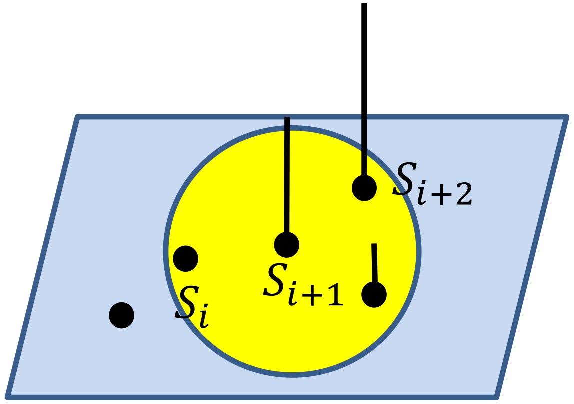


Optimize M
Greedy strategy

- Greedy strategy:
  - iterative – local optimal improvement
  - $S_i, S_{i+1}, S_{i+2}, \ldots$ (S_{j+1} is “**biggest**“ neighbor of S_j)



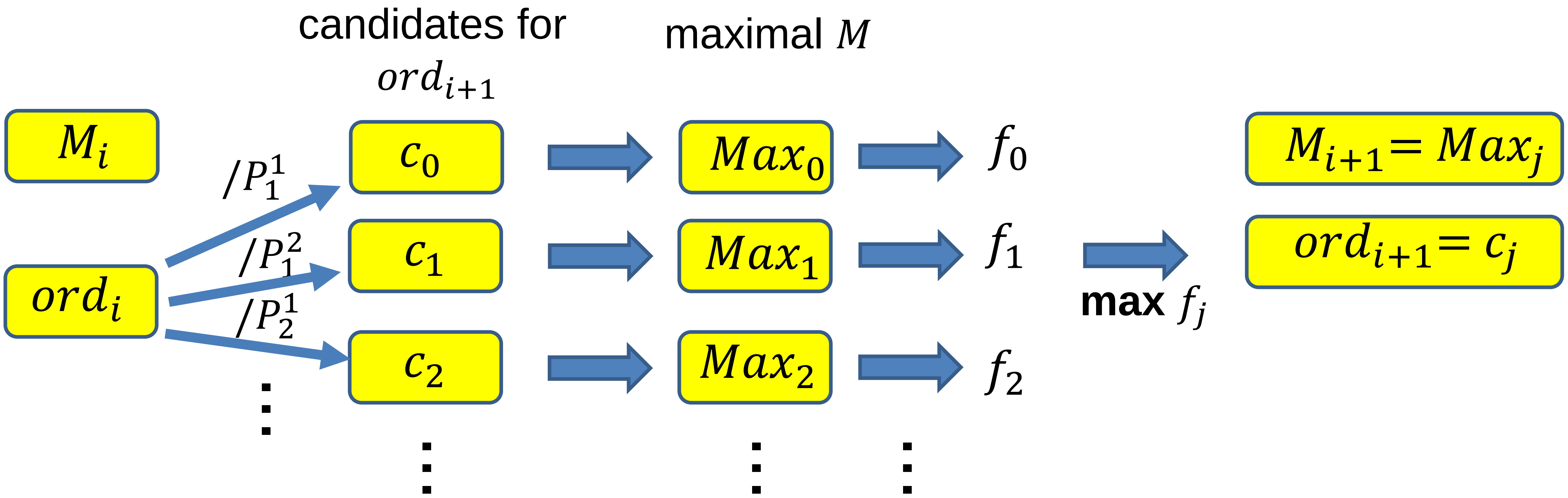
Optimize M
Greedy strategy



- Greedy strategy:
  - iterative – local optimal improvement
  - $S_i, S_{i+1}, S_{i+2}, \ldots$ (S_{j+1} is "**biggest**" neighbor of S_j)
- Looking for M': **minimal** $ord_{M'}$(divisor of ord_M) and $\boldsymbol{size(M')} > \frac{size(p)}{2}$
- Optimize M
  - neighbors: $ord_{i+1} \mid ord_i$
  - "**biggest value**" $= \frac{size(ord_i) - size(ord_{i+1})}{size(M'_i) - size(M'_{i+1})} = \frac{\Delta\, size(ord)}{\Delta\, size(M')}$ **- maximize** **- minimize**

i-th iteration

Idea: divide order by prime power - $ord_{i+1} = ord_i / P_j^l$,



Maximal M

- How to find maximal $M_{i+1} | M_i$ for given $\boldsymbol{ord_{i+1}} | ord_i$?
- Let $M_i = 11 * 13 * 17 * 19$
- Compute partial orders $ord_{11}, \dots, ord_{19}$ mod 11,13,17,19 and factorize.
- $ord_i = lcm(ord_{11}, ord_{13}, ord_{17}, ord_{19}) = 2^3 3^2$
  - maximal exponents of partial orders
- Let $ord_{i+1} = 2^1 3^1$
  - **maximal** $M_{i+1} = 11.13$

	2	3
11 :	1	.
13 :	1	1
17 :	**3**	.
19 :	.	**2**

Maximal M

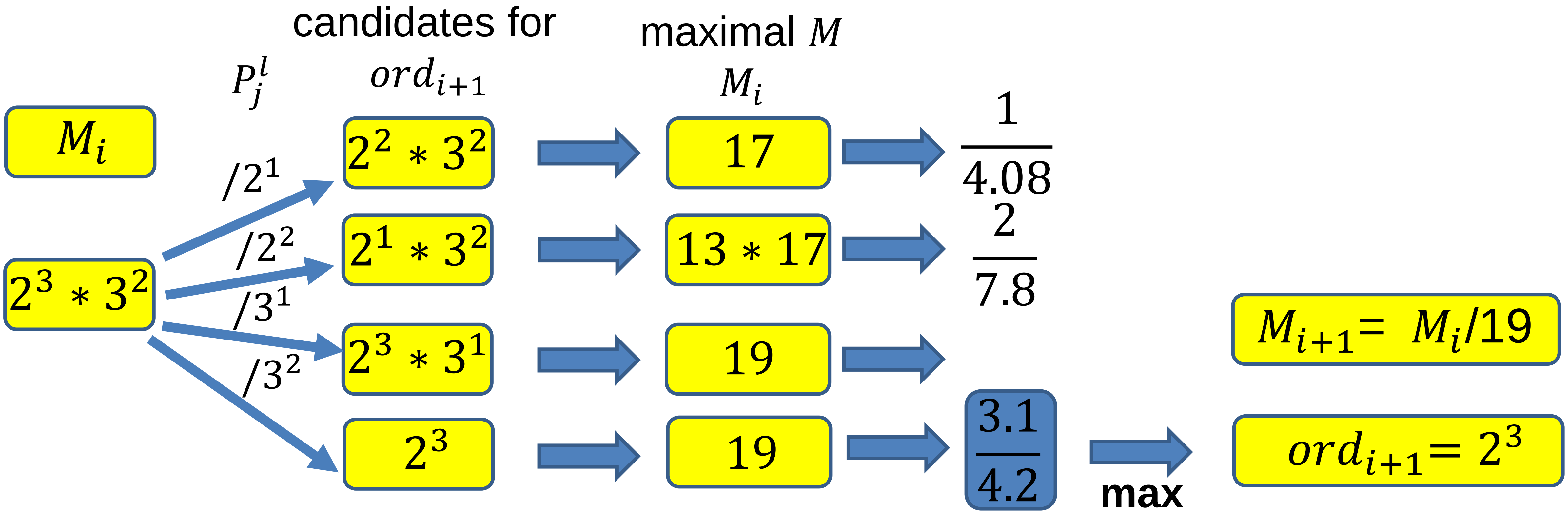
- How to find maximal $M_{i+1} | M_i$ for given $\boldsymbol{ord_{i+1}} | ord_i$?
- Let $M_i = 11 * 13 * 17 * 19$
- Compute partial orders $ord_{11}, \dots, ord_{19}$ $mod\ 11,13,17,19$ and factorize.

	2	3
11 :	1	.
13 :	1	1
17 :	**3**	.
19 :	.	**2**

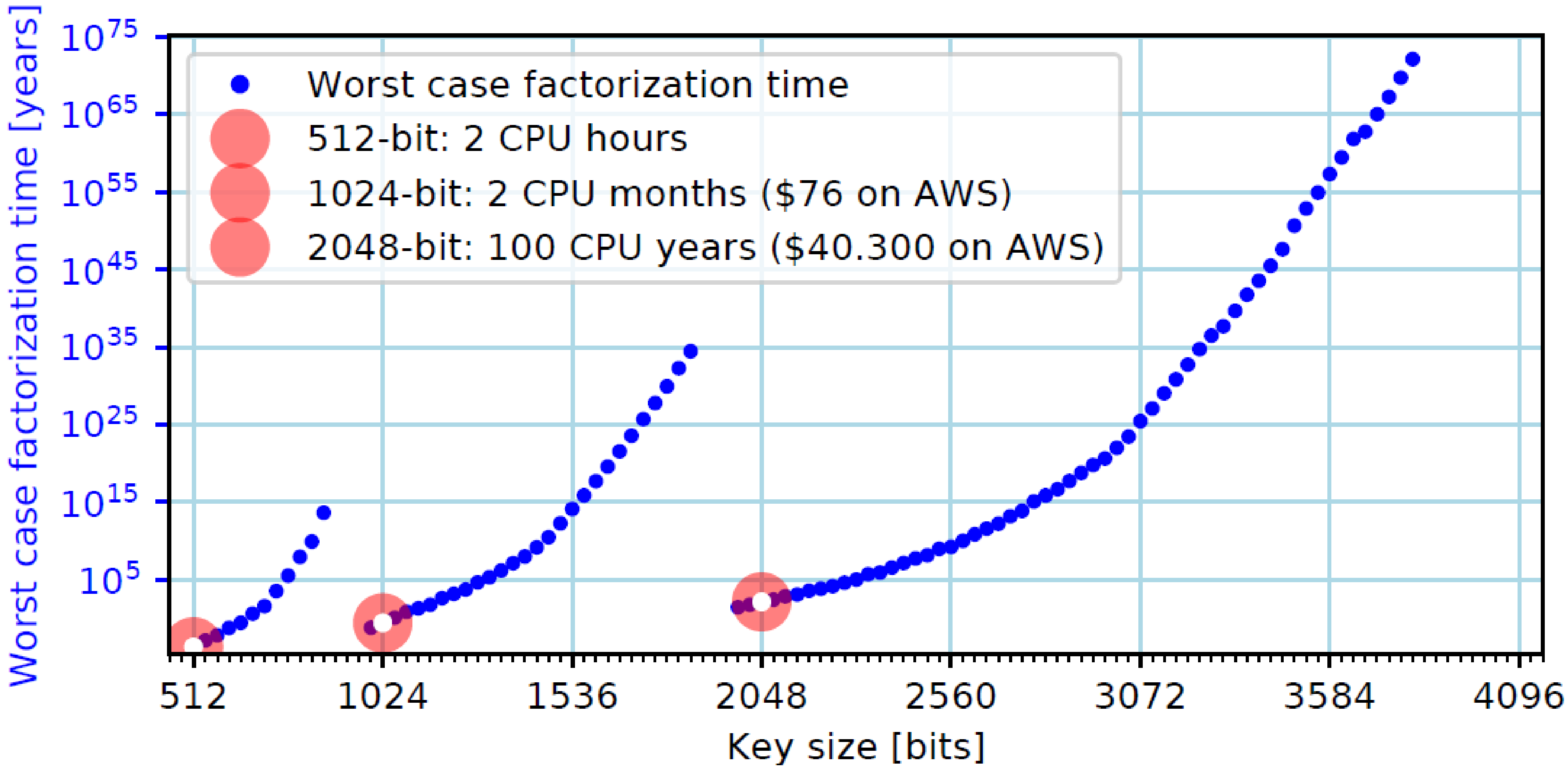
- $ord_i = lcm(ord_{11}, ord_{13}, ord_{17}, ord_{19}) = 2^{\mathbf{3}}3^{\mathbf{2}}$
  - maximal exponents of partial orders
- Let $ord_{i+1} = 2^1 3^2$
  - **maximal** $M_{i+1} = 11.13.19$

Example
i-th iteration

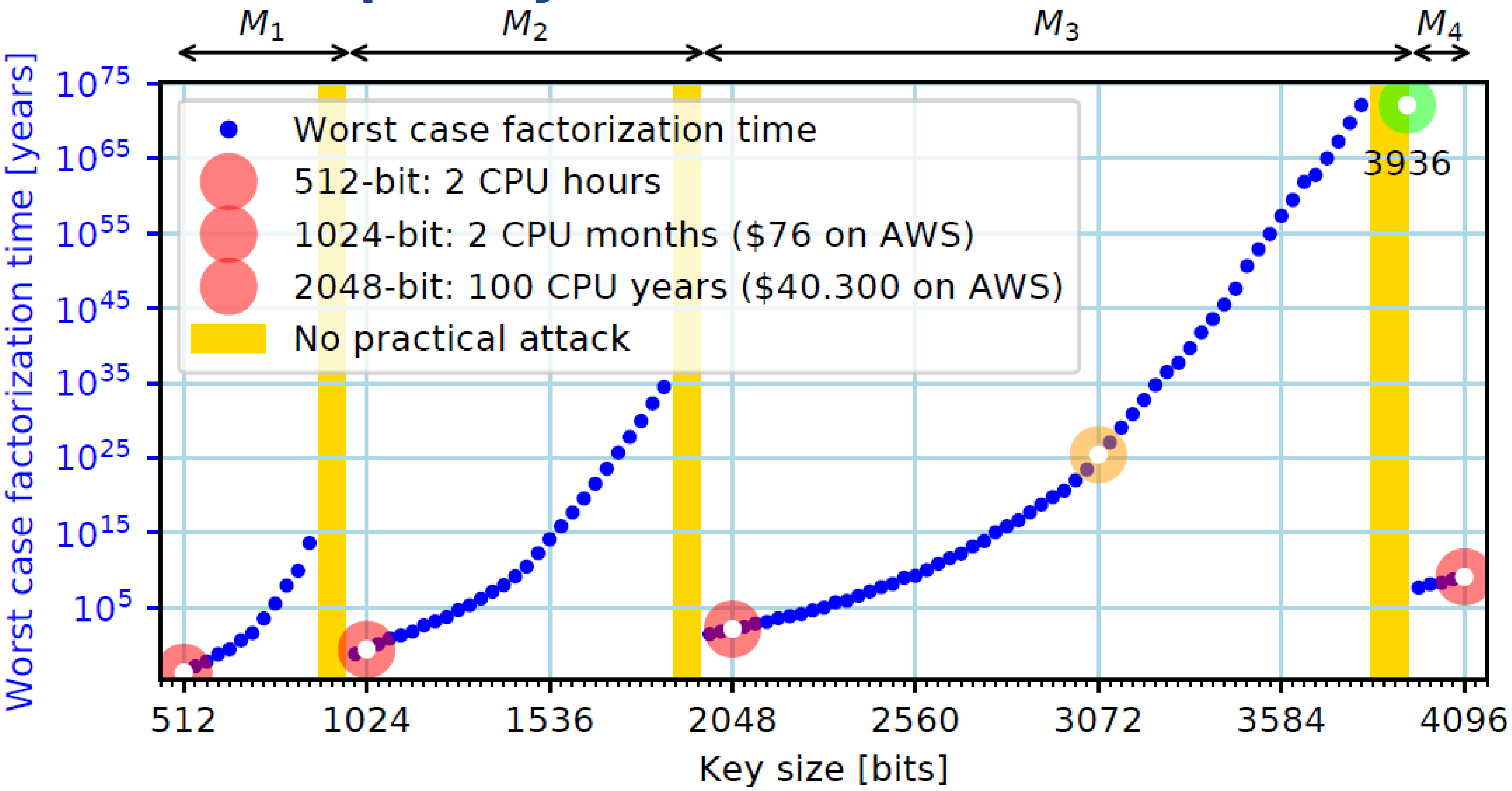
- $M_i = 11 * 13 * 17 * 19,\ ord_i = 2^3 * 3^2,\ P_j^l = 2^1, 2^2, 2^3, 3^1, 3^2$



Attack complexity



Attack complexity



Attack complexity, cost and speed

Key size	University cluster (Intel E5-2650 v3@3GHz Q2/2014)	Rented Amazon c4 instance (2x Intel E5-2666 v3@2.90GHz, estimated)	Energy-only price ($0.2/kWh) (Intel E5-2660 v3@2.60GHz, estimated)
512 b	1.93 CPU hours *(verified)*	0.63 hours, $0.063	$0.002
1024 b	97.1 CPU days *(verified)*	31.71 days, $76	$1.78
2048 b	140.8 CPU years	45.98 years, $40,305	$944
3072 b	$2.84 * 10^{25}$ years	$9.28 * 10^{24}$ years, $\$8.13 * 10^{27}$	$\$1.90 * 10^{26}$
4096 b	$1.28 * 10^{9}$ years	$4.18 * 10^{8}$ years, $\$3.66 * 10^{11}$	$\$8.58 * 10^{9}$

- Worst case shown, average is half, uniform distribution of complexity

Coppersmith’s algorithm Characteristics

- Usage:
  - Attack on RSA – **private key** or message recovery
  - Factorization, smooth numbers

- Requirements: partial information **must** be known
  - Key recovery - bits of primes,
  - Message - bits of message

Coppersmith's algorithm
Problem transformation

Steps:

1. Problem – **known partial information** about solution
2. **Modular** polynomial equation - with solution x_0
3. Polynomial equation over $\boldsymbol{Z}$ - same solution x_0
4. Solution – standard algorithms (Berlekamp-Zassenhaus)

$$problem \xrightarrow{\textbf{partial info}} f(x) \equiv 0 \; mod \; b \xrightarrow{\textbf{Coppersmith's}} g(x) = 0 \xrightarrow{\textbf{standard algs}} x_0$$

Coppersmith's algorithm Factorization (simplified)

1. **known** partial information about prime factor p of N :
   - lower bits, upper bits or $\boldsymbol{p\ mod\ M}$

$$p = k * M + 65537^{a}\ mod\ M$$

2. Equation modulo unknown factor p - solution k

$$(x * M + 65537^{a}\ mod\ M) \equiv 0\ mod\ p$$

3. Equation over Z – same solution k

$factor\ N$ → ($\boldsymbol{p\ mod\ M}$) → $f(x) \equiv 0\ mod\ p$ → (**Coppersmith's**) → $g(x) = 0$

Coppersmith's algorithm
Idea

Idea: For $f(x) \equiv 0 \; mod \; p$ find $g(x)$ with the same solution k:

$g(k) \equiv 0 \; mod \; p^m \; \wedge \; |g(k)| < p^m \quad \Rightarrow \quad g(k) = 0$ over Z

How to construct $g(x)$?

1. Linear combination of polynomials with the same roots as $f(x)$
   $g(x) = \sum_l a_l * f_l(x)$ for $f_l(x) = x^i N^{m-j}.f^j(x)$
   $f_l(k) \equiv 0 \; mod \; p^m$ since $p^{m-j} | N^{m-j}$ and $p^j | f^j(k)$
2. Small $g(k)$ - use LLL algorithm

Links

- Our paper: The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli https://dl.acm.org/citation.cfm?id=3133969
- Our page with some info and detection tool: https://crocs.fi.muni.cz/public/papers/rsa_ccs17
- Joye, Pailier: Fast Generation of Prime Numbers on Portable Devices https://link.springer.com/chapter/10.1007/11894063_13
- Svenda et. al: The Million-Key Question—Investigating the Origins of RSA Public Keys https://www.usenix.org/node/197198
  - Technical report https://crocs.fi.muni.cz/_media/public/papers/usenixsec16_1mrsakeys_trfimu_201603.pdf

Thank you for your attention!

Questions are welcome.