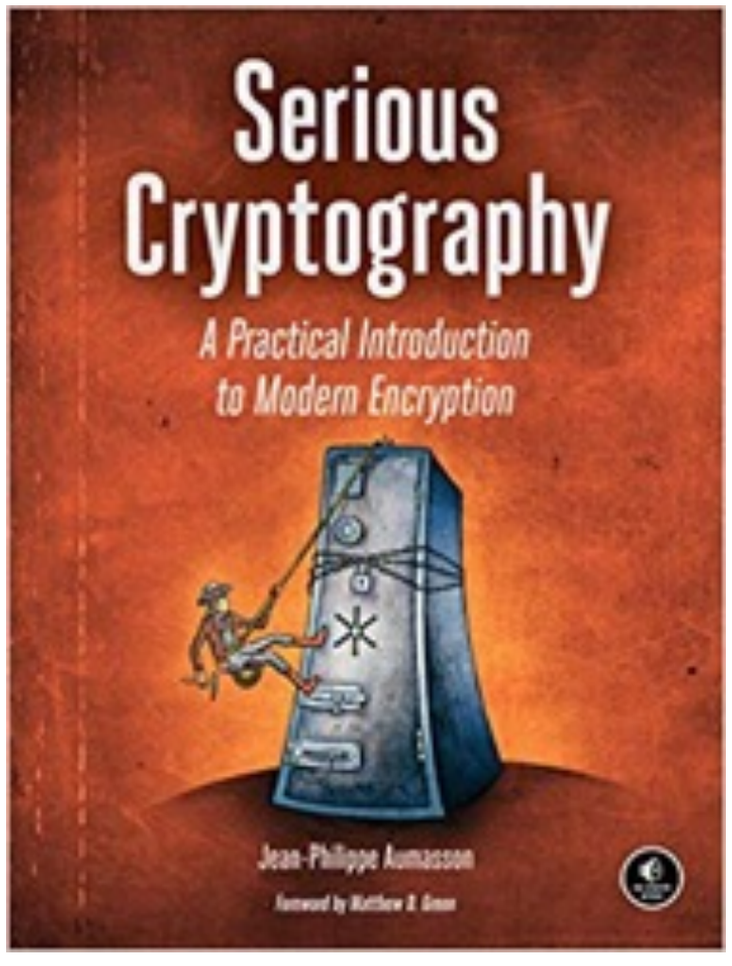
CNIT 141

Cryptography for Computer Networks



9. Hard Problems

Updated 11-2-22

Topics

- Computational Hardness
- Complexity Classes
- The Factoring Problem
- The Discrete Logarithm Problem
- How Things Can Go Wrong

Computational Hardness

Computational Hardness

- ***Computationally hard*** problems
  - Also called ***intractable problems***
  - Take an unreasonable amount of time to solve
  - Regardless of hardware

Measuring Running Time

- Search an array of ***n*** elements to find ***x***
  - Loop goes from 1 to ***n***
  - Expected value: ***n/2***
- Complexity is ***linear*** in ***n***
  - Doubling ***n*** doubles running time

```
search(x, array, n):
    for i from 1 to n
        if (array[i] == x) return i;
    return 0;
```

Complexity Classes

- Searching a list: **linear** or ***O*(*n*)**
- Sorting a list: ***linear-logarithmic*** or ***O*(*n log n*)**
- Brute-force key recovery:
  exponential or ***O*(2^n)**
- ***Quadratic time*** is ***O*(*n*^{2})**

Linear is Fast (Easy)

- Compared to exponential or quadratic

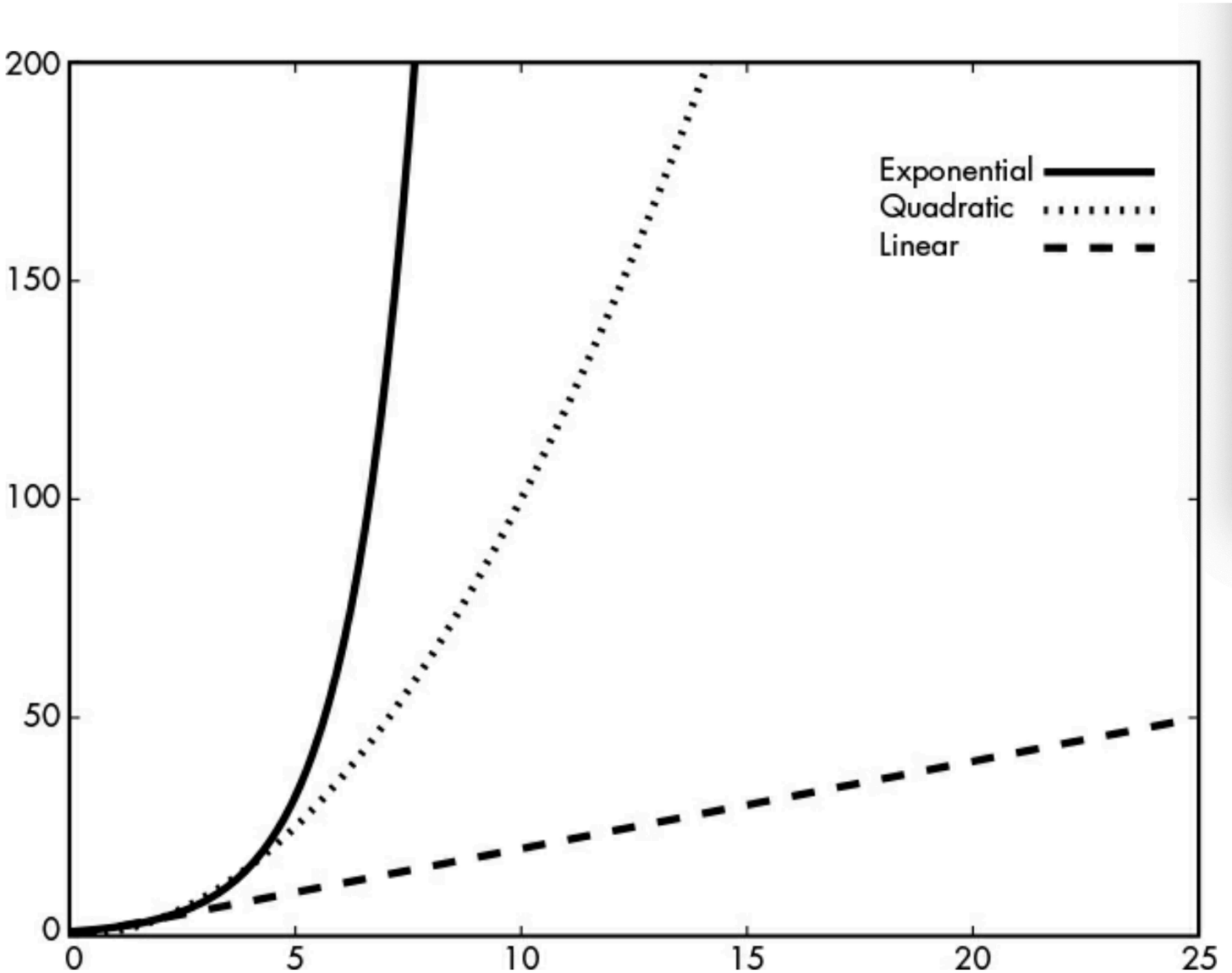


Figure 9-1: Growth of exponential, quadratic, and linear complexities, from the fastest to the slowest growing

Polynomial vs. Superpolynomial Time

- ***Polynomial time*** includes
  $O(n^2)$, $O(n^3)$, $O(n^4)$, etc.
  - They are considered practically feasible
- ***Superpolynomial time*** is anything that grows faster than polynomial, like $O(2^n)$ or $O(n^{\log(n)})$.
  - They are considered impractical, or ***hard***

Quadratic v. Superpolynomial

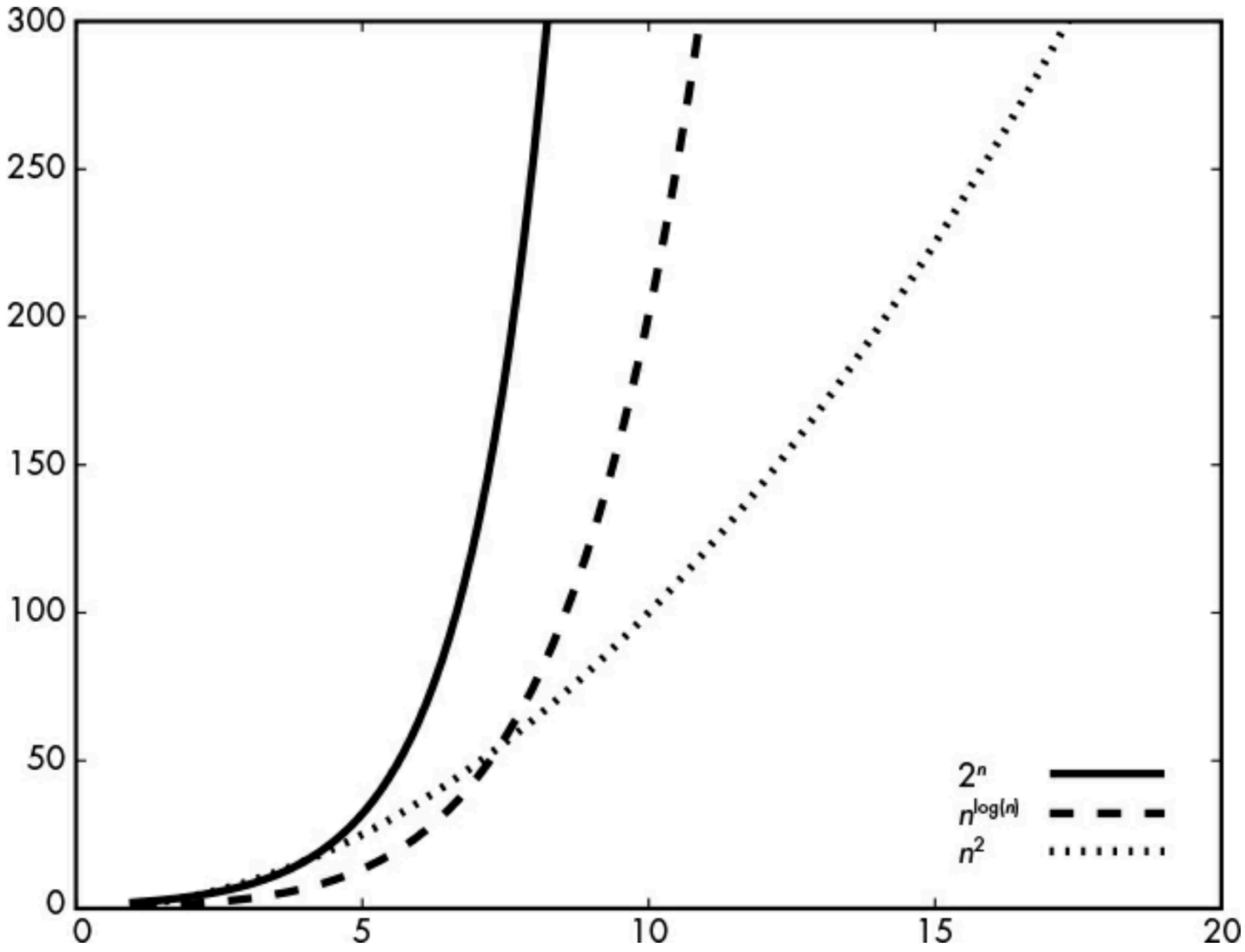


Figure 9-2: Growth of the 2^n, $n^{\log(n)}$, *and* n^2 *functions, from the fastest to the slowest growing*

Complexity Classes

Complexity Classes

- ***Time complexity***
- **TIME(n^2)**
  - All problems solvable in ***O***(n^2)
- **TIME(2^n)**
  - All problems solvable in ***O***(2^n)

Space Complexity

- The memory required by an algorithm
- **SPACE(n^2)**
  - Require ***O*(n^2)** bits of memory

Nondeterministic Polynomial Time

- **P** is the class of all polynomial-time algorithms
- **NP** is the class of non-deterministic polynomial-time algorithms
  - Problems for which a solution can be verified in polynomial time
  - Even though the solution may be hard to find

NP Problems

- Recovering a secret key with known plaintext
- ***Easy*** to verify whether a key is correct (**P**)
- Finding the key is ***hard*** but that's a different problem

Problems Outside NP and P

- Consider brute-forcing the one-time pad
  - When the correct plaintext is unknown
  - You cannot recognize the solution when you find it
  - This is very hard, not in **P** and not in **NP**
- Verify that no solution exists to a problem
  - Must test all possible solutions
  - An unlimited number of possibilities

NP-Complete Problems

- The hardest problems in the class **NP**
- We don't know how to solve them in polynomial time
- But they are all equally hard
- An efficient solution for any one **NP-complete** problem can be used to solve all the others

NP-Complete Problems

The traveling salesman problem Given a set of points on a map (cities, addresses, or other geographic locations) and the distances between each point from each other point, find a path that visits every point such that the total distance is smaller than a given distance of *x*.

The clique problem Given a number, *x*, and a graph (a set of nodes connected by edges, as in Figure 9-3), determine if there's a set of *x* points or less such that all points are connected to each other.

The knapsack problem Given two numbers, *x* and *y*, and a set of items, each of a known value and weight, can we pick a group of items such that the total value is at least *x* and the total weight at most *y*?

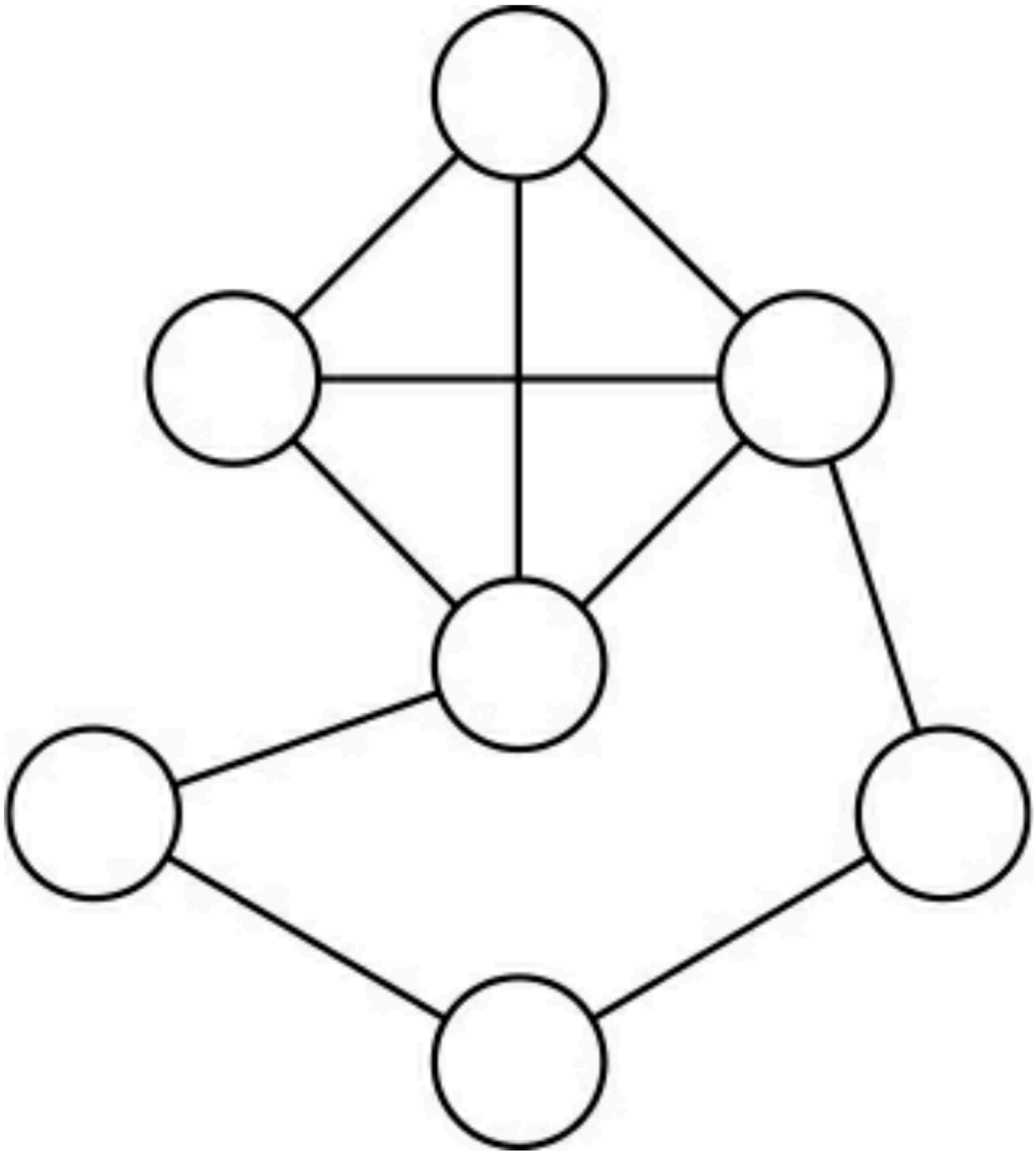


*Figure 9-3: A graph containing a clique of four points. The general problem of finding a clique (set of nodes all connected to each other) of given size in a graph is **NP**-complete.*

NP-Hard

- Some video games are **NP-complete**
  - Tetris, Super Mario Brothers, Candy Crush Saga
- Some are even harder: **NP-hard**
  - Provably as difficult as **NP-complete** problems
  - May not be in **NP**

P vs. NP

- If you could solve the hardest **NP** problem in polynomial time
  - You could solve all **NP** problems in polynomial time
  - **NP** would equal **P**
- No one has proven this yet
- There's a $1 million bounty for the proof

Does P = NP?

- Most theorists say no
- If it did, any easily-verified solution would be easy to find, in principle
- All cryptography would be insecure, in principle
- In practice, it might not matter, if easy solutions are difficult to find

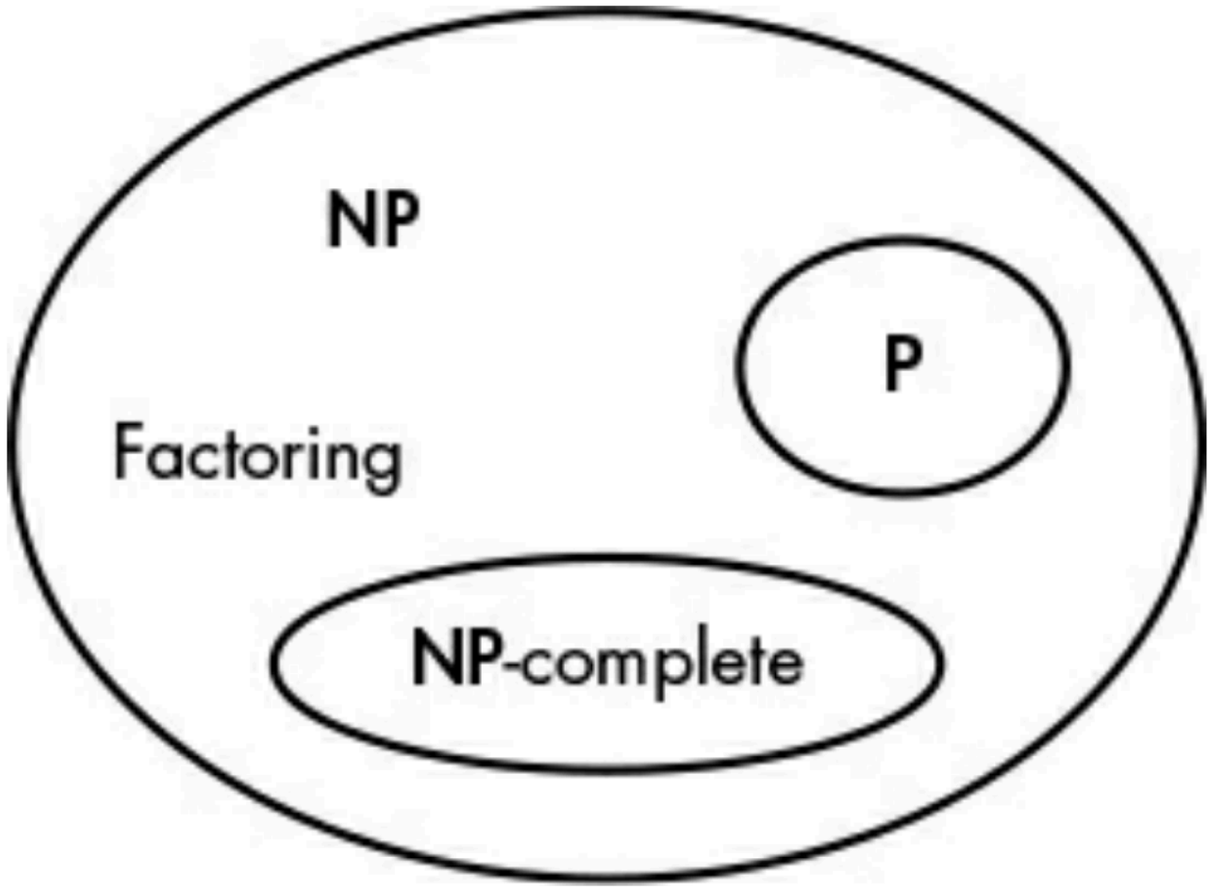
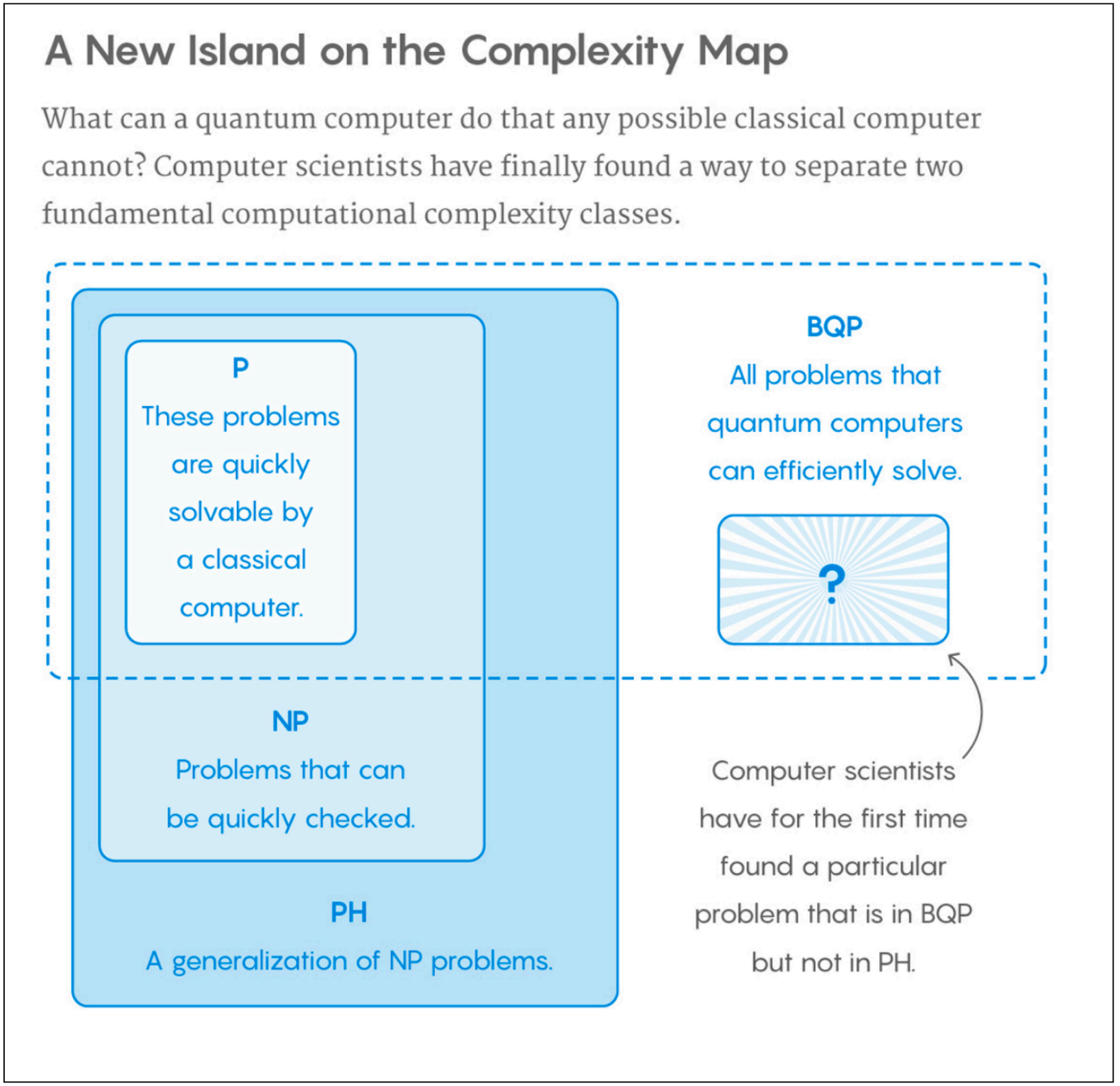


*Figure 9-4: The classes **NP**, **P**, and the set of **NP**-complete problems*

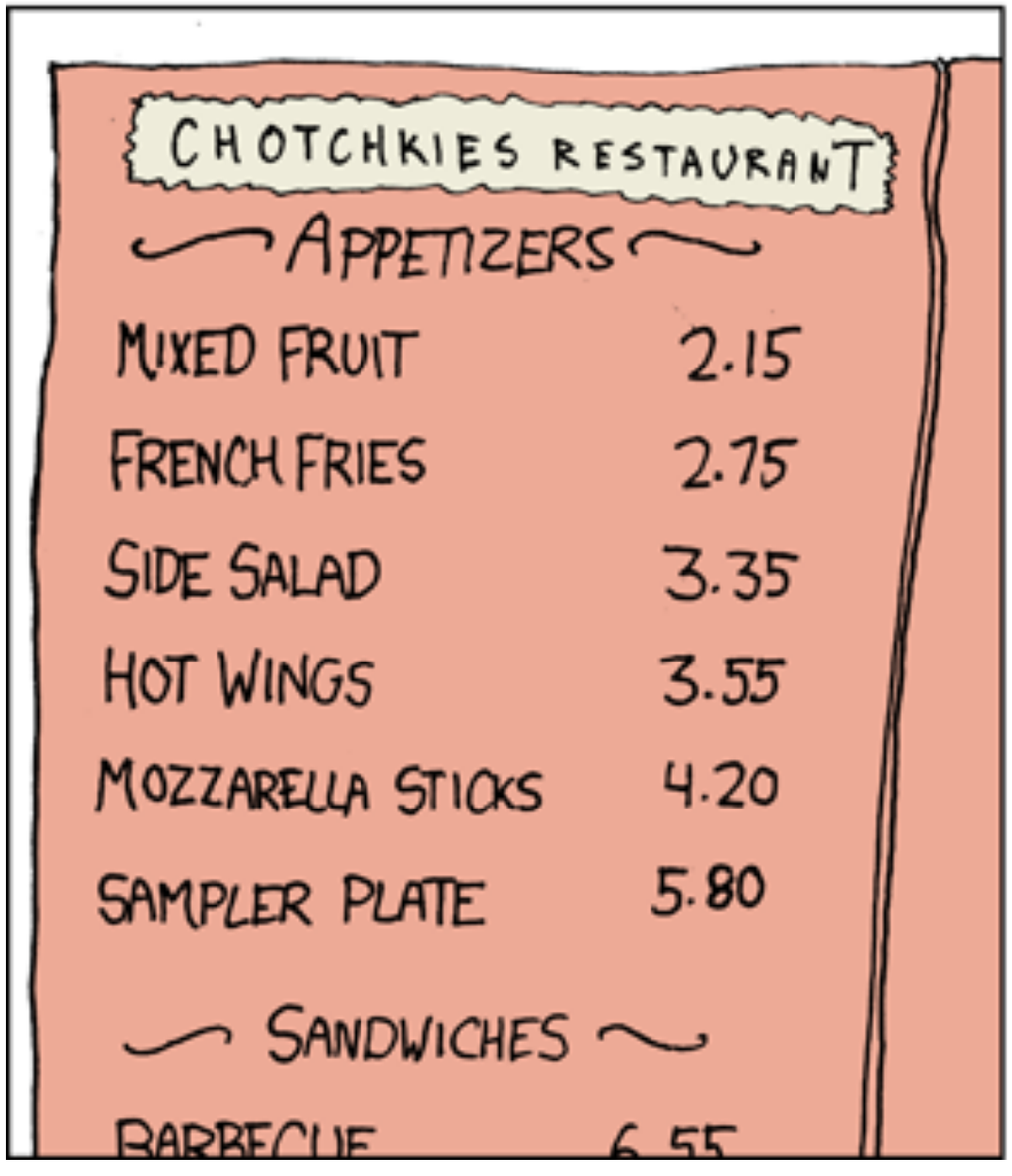
Quantum Computers

- Link Ch 9e



MY HOBBY:

EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



- Link Ch 9f

Practical Cryptography

- If breaking a cipher were **NP-complete**
  - That would be a very strong cipher
  - But **NP-complete** problems are impractical for cryptography
    - Because they are easy in specific cases
- So real cryptography uses problems that are ***probably not*** **NP-hard**

Lattice Problems

- Including Learning With Errors
- The only **NP-hard** problems successfully used in cryptography
- The basis for ***New Hope***
- Made it to round 2, but not round 3, of the NIST Post-Quantum Cryptography Standardization Process
  - Links Ch 9i, 9j, 9k

The Factoring Problem

The Factoring Problem

- Given a large number ***N*** that is the product of two primes ***p*** and ***q***
  - Find ***p*** and ***q***
- How difficult is this problem?
- ***Prime numbers*** cannot be divided evenly by any number other than themselves and one
  - 1, 2, 3, 5, 7, 11 are prime
  - 9=3x3 and 15=3x5 are not prime

Factoring Large Numbers in Practice

- Simplest algorithm
  - Try dividing by all numbers from 2 to ***N***-1
  - If ***n*** is the number of bits in ***N***
    - This is $O(2^n)$ --a hard problem
- Requires 2^{256} operations for 256-bit ***N***

Factoring Large Numbers in Practice

- Improved algorithm
  - Try only primes from 2 to sqrt(***N***)
    - This is $O(2^{n/2}/n)$ --still hard, but easier
- Requires 2^{120} operations for 256-bit ***N***

Factoring Large Numbers in Practice

- Fastest known algorithm
  - **General number field sieve (GNFS)**
    - Requires $\mathbf{2^{70}}$ operations for 1024-bit ***N***
    - Requires $\mathbf{2^{90}}$ operations for 2048-bit ***N***
- So we recommend 4096-bit keys for 128 bits of security

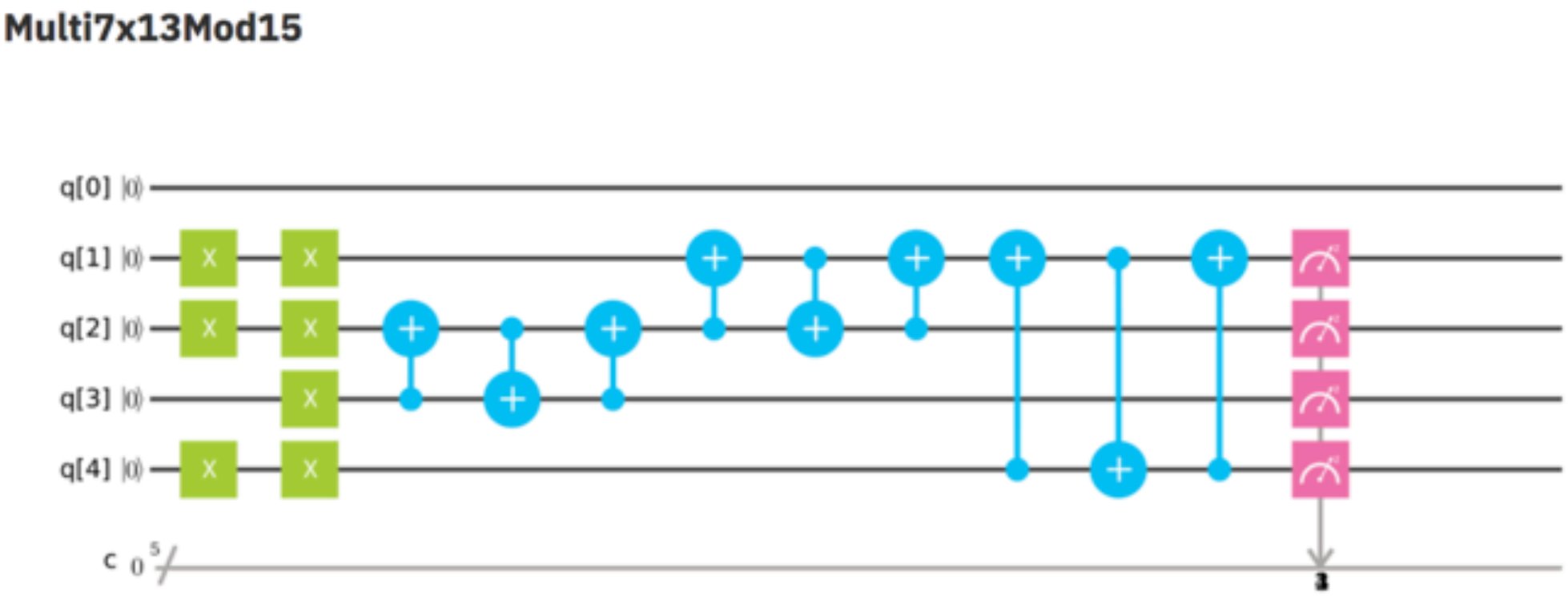
Experimental Results

- In 2005, a 663-bit ***N*** was factored using 75 cpu-years
- In 2009, a 768-bit ***N*** was factored using 2000 cpu-years
- People speculate that the NSA can factor a 1024-bit ***N***

Is Factoring NP-Complete?

- No polynomial-time algorithm is known
  - Suggesting that factoring is not in **P**
- However, we can easily verify a factor once it is found
  - So factoring is in **NP**
- Factoring is probably easier than **NP-complete** problems, but this has not been proven

Quantum Computers



- Can factor numbers easily using quantum algorithms
- But they don't work well enough yet

Hardness Assumption

- Cryptography starts from a problem which is ***assumed*** to be hard
- The encryption is proven to be at least as hard as that "hard" problem
- ***Factoring*** and ***discrete logarithm*** problems are used as hardness assumptions

The Discrete Logarithm Problem

What is a Group?

- A set of ***elements*** and an ***operation*** × that obey certain ***group axioms***
- Example: $\mathbf{Z}_p^*$
  - Numbers from 1 to ***p***-1, where ***p*** is prime
- $\mathbf{Z}_5^*$ contains {1, 2, 3, 4}

Group Axioms

- **Closure**
- **Associativity**
- **Identity existence**
- **Inverse existence**

Group Axioms

- **Closure**
  - For any two elements ***x*** and ***y*** in the group ***x* × *y*** is in the group
- **Associativity**
  - For any three elements ***x***, ***y***, and ***z***

  (*x* × *y*) × z = *x* × (*y* × z)

Group Axioms

- **Identity existence**
  - There is an identity element ***e*** such that

    $e \times x = x \times e = x$
- **Inverse existence**
  - For any ***x*** in the group, there exists ***y*** such that

    $x \times y = y \times x = e$

Commutative Groups

- For all ***x*** and ***y*** in the group.

  $$x \times y = y \times x$$

Cyclic Groups

- There's at least one element ***g*** such that
  $g^1, g^2, g^3, \ldots \bmod p$
- Span all group elements
- ***g*** is called the ***generator*** of the group

The Hard Thing

- The DLP consists of finding ***y*** for which
  $g^y = x$
- Within a group Z_p^*, where ***p*** is a prime number
- And ***x*** is a known group element
- This problem is about as hard as factoring

How Things Can Go Wrong

Unlikely Problems

- These are possible but experts don't expect them to happen
  - Someone finding a fast algorithm to factor numbers
  - Someone proving that **P** = **NP**

When Factoring is Easy

- This 1024-bit number is easily factored, because it has a small factor

$$179769313486231590772930519078902473361797697894230657273430081157739343819933$$
$$842986982557174198257278917258638193709265819186026626180659730665062710995556$$
$$578639447715608415186895652841691982921107202317165369124890481512388558039053$$
$$427125099290315449262324709315263256083132540461407052872832790915388014592$$

$$2^{800} \times 641 \times 6700417 \times 167773885276849215533569$$
$$\times 3741405716132237595740814883432396 9$$

Other Easily-Factored Numbers

- If ***p*** and ***q*** are not random
  - Near a known value 2^{b}
  - Or some bits of ***p*** or ***q*** are known
- Or if ***N*** is small, such as 128-bit RSA

OpenSSL Allows Short Keys

```
$ openssl genrsa 31
Generating RSA private key, 31 bit long modulus
.++++++++++++++++++++++++++++++
.++++++++++++++++++++++++++++++
e is 65537 (0x10001)
-----BEGIN RSA PRIVATE KEY-----
MCsCAQACBHHqFuUCAwEAAQIEP6zEJQIDANATAgMAjCcCAwCSBwICTGsCAhpp
-----END RSA PRIVATE KEY-----
```

Listing 9-3: Generating an insecure RSA private key using the OpenSSL toolkit

Original RSA Paper

Digits	*Number of operations*	*Time*
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

We recommend that n be about 200 digits long. Longer or shorter lengths can be used depending on the relative importance of encryption speed and security in the application at hand. An 80-digit n provides moderate security against an attack using current technology; using 200 digits provides a margin of safety against future developments. This flexibility to choose a key-length (and thus a level of security) to

- Recommended 512-bit keys (in 1978)
  - Link Ch 9g

Weak Diffie-Hellman and the Logjam Attack

- 2015 paper presents two attacks
  - **Logjam** MITM attack downgrades TLS to "export-grade" with 512-bit keys
  - **State-level adversaries** can probably find a 1024-bit secret prime number used by millions of servers, and it appears that the NSA has done so
- Link Ch 9h

Protocol	Vulnerable to Logjam
HTTPS — Top 1 Million Domains	8.4%
HTTPS — Browser Trusted Sites	3.4%
SMTP+StartTLS — IPv4 Address Space	14.8%
POP3S — IPv4 Address Space	8.9%
IMAPS — IPv4 Address Space	8.4%

	Vulnerable if most common 1024-bit group is broken
HTTPS — Top 1 Million Domains	17.9%
HTTPS — Browser Trusted Sites	6.6%
SSH — IPv4 Address Space	25.7%
IKEv1 (IPsec VPNs) — IPv4 Address Space	66.1%


Kahoot!