Neat Ideas in Computer Science

Ooi Wei Tsang
School of Computing

DISCLAIMER

a personal view of what’s neat

not comprehensive

not rigorous

some problems can't be solved by a computer

some problems can be solved easily in one way but difficult in the reverse direction

some problems can be solved randomly (but still gives right solution most of the time)

Is it possible to pose an Algo*Mania contest problem that is impossible to solve?

Given a program P with input I, will the program halt?

YES: P(I) eventually halt
NO: P(I) loops forever

Write another program

```
X(P) {
   while (HALT(P, P)) {
      // loop forever if P halts
   }
}
```

What is the output of HALT(X,X)?

```
X(P) {
   while (HALT(P, P)) {
      // loop forever if P(P) halts
   }
}
```

Suppose HALT(X,X) is YES
(that is HALT tells us X(X) will halt)

Then the while loop will loop forever, meaning X(X) will not halt!

```
X(P) {
   while (HALT(P, P)) {
      // loop forever if P(P) halts
   }
}
```

HALT(X,X) must be false!
(that is, HALT says X(X) will loop forever)

But if HALT(X,X) is false, the while loop won’t execute and X(X) will exit.

Halting Problem

First problem shown to be non-computable

Why is this neat?

Computer can’t program better than human!

Given two programs P1 and P2,
are they equivalent?

Is a given program buggy?

Given a program P,
output optimized version of P

Computer can’t replace mathematician

Fermat's Last Theorem

$x^n + y^n = z^n$ has no non-zero integer solution for $n > 2$

```
Fermat() {
   for all possible non-zero integer values
   of x, y, z, and n > 2 do
      if x^n + y^n = z^n  // found a solution
         return true
}
```

HALT(Fermat, nil) would proved
Fermat's Last Theorem by returning NO

Other Non-computable Problems

Given a set of substitution rules,
and two strings s and t,
can we transform s to t
by applying the set of rules?

P and NP

Not all problems has known efficient solutions some problems are known to have efficient solutions

e.g. **shortest** path on a graph

some problems have
no known efficient solutions

e.g. **longest** path on a graph

No one knows if
integer factoring
can be done efficiently

Factor the following 200-digit integer:

2799783391122132787082946763872260162107
0446786955428537560009929326128400107609
3456710529553608560618223519109513657886
3710595448200657677509858055761357909873
4950144178863178946295187237869221823983

2799783391122132787082946763872260162107044678695542853756000992932612840010760934567105295536085606182235191095136578863710595448200657677509858055761357909873495014417886317894629518723786922182398

3

=

3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349

x

7925869954478333033347085841480059687737975857364219960734330341455767872818152135381409304740185467

Christmas 2003 - May 2005

Equivalent of 55 years of CPU time on a 2.2 GHz CPU

Some problems are easy to compute one way, but computing the reverse is difficult (unless you know a secret)

A x B = C

given A and B, find C is easy
given C, find A and B is hard

Why is this neat?

Public Key Cryptography

Easy: **encrypt** a message

Hard: **decrypt** the message
(unless know the secret)

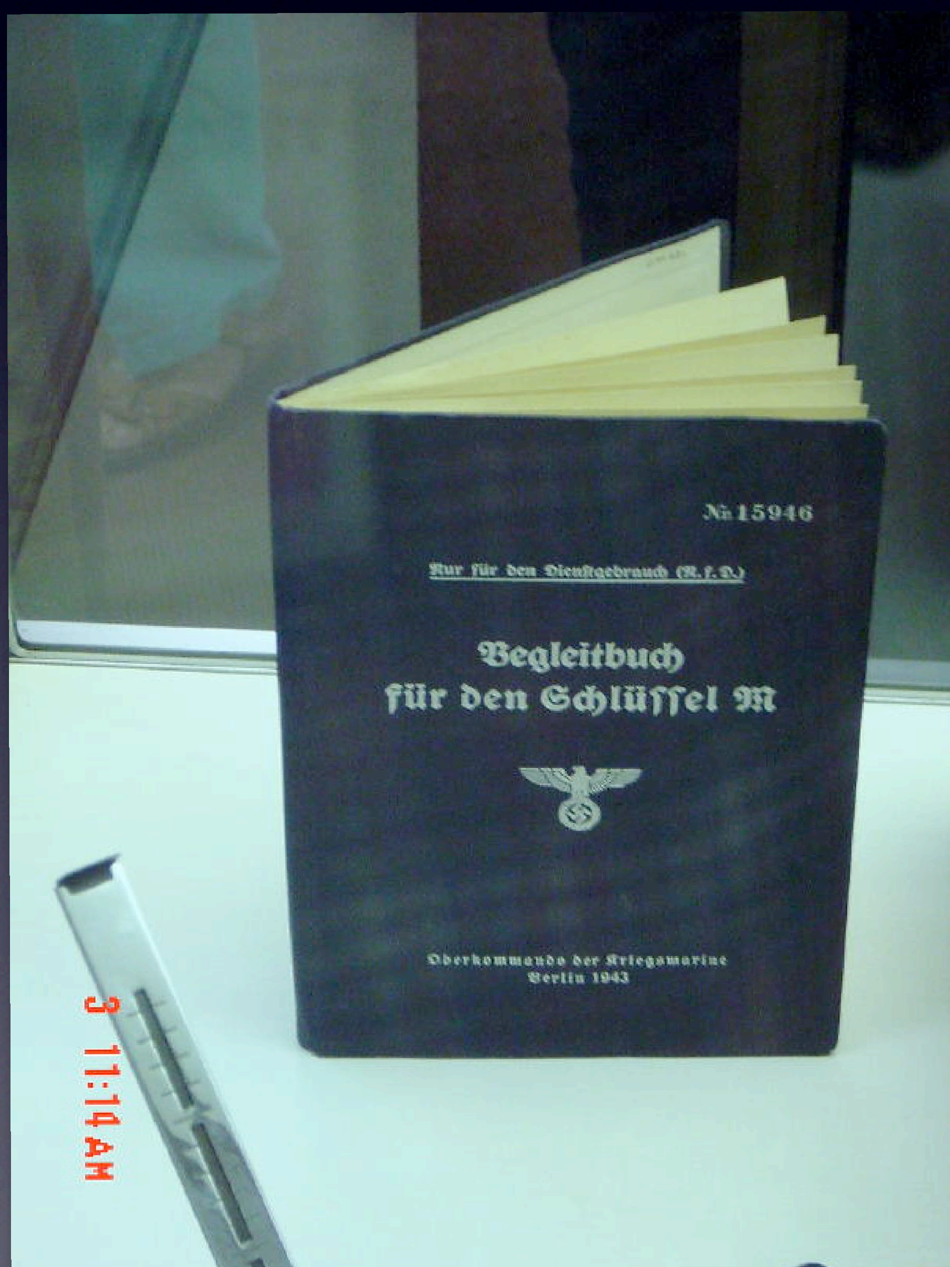
Public Key Cryptography

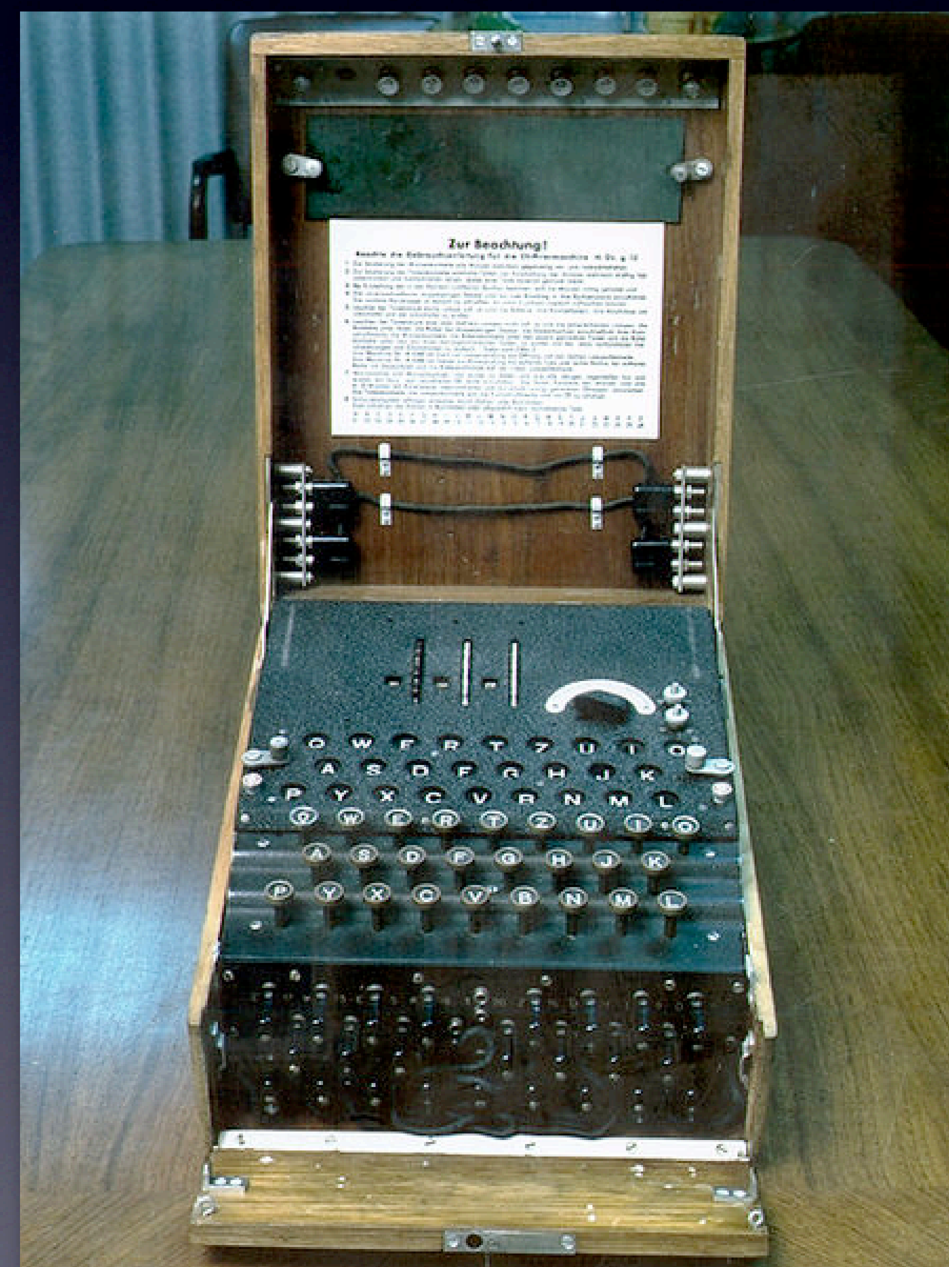
publish C (product of two large primes A and B)

encrypt message using C

can only decrypt the message if we know A and B

Sender and receiver no longer have to agree on a common key before communication!





A hash function transforms input into a fixed length string.

hash(input) = k

e.g., MD5(“Algo*Mania”) =
2e8f46a660fb57201b93ed9c1cf86d08

hash(input) = k

good hash function:
slight change in input gives totally different k

e.g., MD5(“Algo*Mania”) =
2e8f46a660fb57201b93ed9c1cf86d08

MD5(“algo*mania”) =
92ae377f2f5cccf585eb84ccd7c8156c

hash(input) = k

good hash function:
given k, hard to guess input

e.g., MD5(?) =
2e8f46a660fb572012343ed9c1cf86d08

build data structures (hash tables)

store passwords

verify file integrity

use as fingerprint to identify files

Authenticated Messages

with common secret

Sender: h = hash(msg + secret)

send msg and h

Receiver: compute h’ = hash(msg’ + secret)

if h’ = h then very likely msg = msg’

Mitigate Spam

Sender must spend some effort to show it's sincerity before receiver accepts the email.

Sender must find a number X such that
first k bits of
hash(X + time + receipient email)
are zeros.

include X in the email

X-Hashcash: 1:20:060408:adam@cypherspace.org::1QTjaYd7niiQA/sc:ePa

Receiver verifies that the first
k bits of the hash are all zeros

Other one way function can be used.

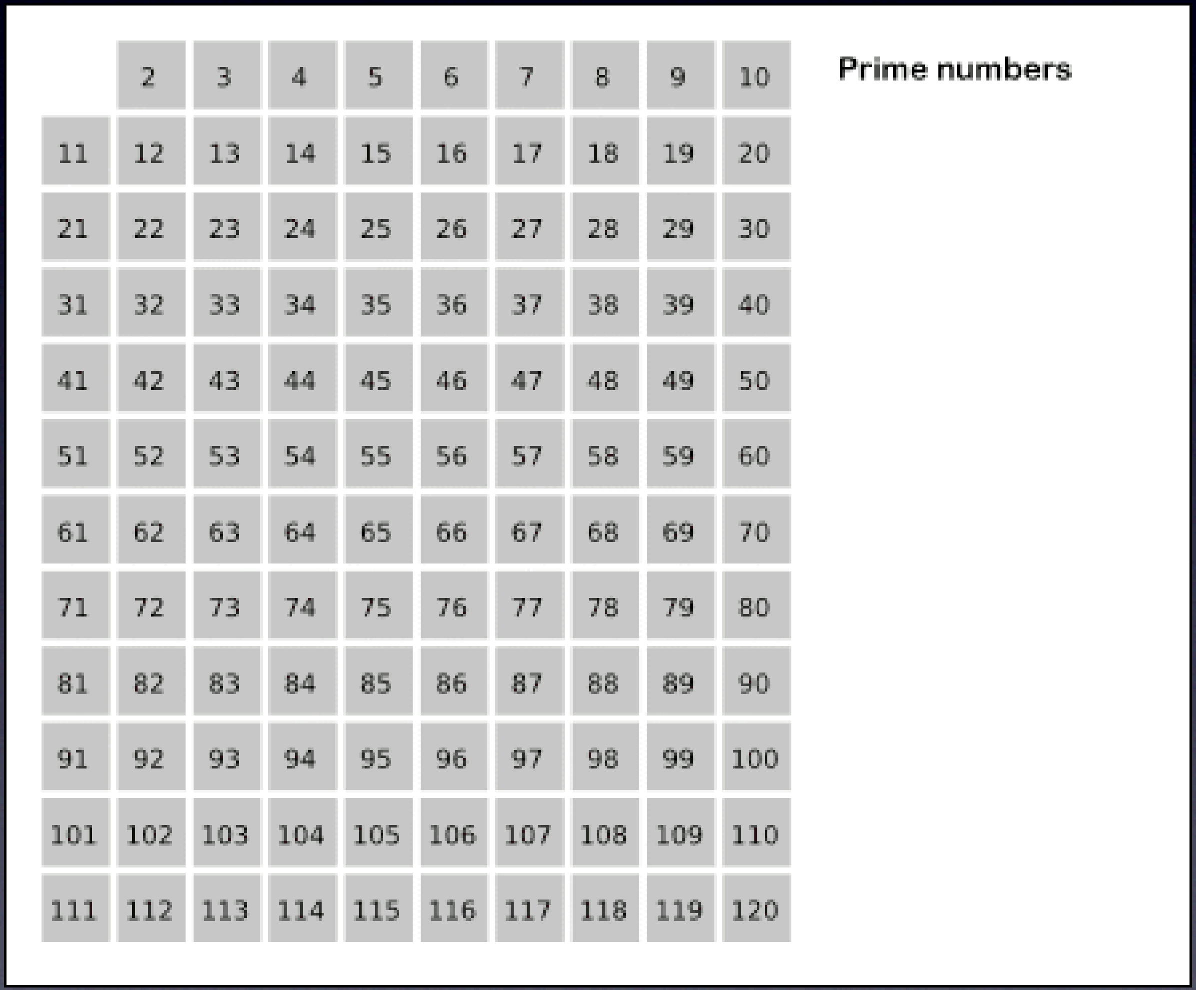
Recipient can also issue a challenge (e.g. factor this!) to sender

Integer factoring is especially hard
if the number is a
product of two very large primes.

How to test if a number is prime?

```
IsPrime? (n) {
    for (k = 2 to n-1) {
        if n is divisible by k then
            return not a prime
    }
    return it's a prime
}
```

Sieve of Eratosthenes



(Animation from Wikipedia)

is

3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349

prime?

I can be **99.9999**% sure
this number is a prime
by looping only **20** times.

Fermat showed that if n is prime then

$$a^{n-1} = 1 \mod n$$

for all values of a in [1 .. n-1]

but if n is not prime then

$$a^{n-1} = 1 \bmod n$$

for at most half the values of a in [1 .. n-1]

A Probabilistic Algorithm

```
IsPrime? (n) {
    repeat k times
      randomly pick a from between 1 and n-1
      if a^(n-1) != 1 mod n then
          return not a prime

    return it's a prime // with prob >= 1 - 1/2^k
}
```

I can tell if

35324619344027701212726049781984643686711974001976250236493034687761212536794232000585479565280883 49

is a prime with a probability 0.999999 by looping 20 times instead of 10^{100} times

NOTE: The above discussion ignores the existance of Carmichael numbers, which are odd composites that satisfies Fermat's little theorem.

some problems can’t be solved by a computer

some problems can be solved easily in one way but difficult in the reverse direction

some problems can be solved randomly (but still gives right solution most of the time)

The End