Logic and Computation – CS 2800
Fall 2019

Lecture 11
Propositional logic continued
The SAT problem

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Outline

- The SAT problem
- P vs NP
- The power of XOR – a bit of cryptography

P vs NP and the SAT problem

Motivating questions

- Is it possible to check automatically whether a given Boolean formula is satisfiable?
- Yes:
  1. Build the truth table of the formula
  2. Check whether there is at least one “T” in the last column
  3. The row which has the “T” gives a satisfying assignment

Motivating questions

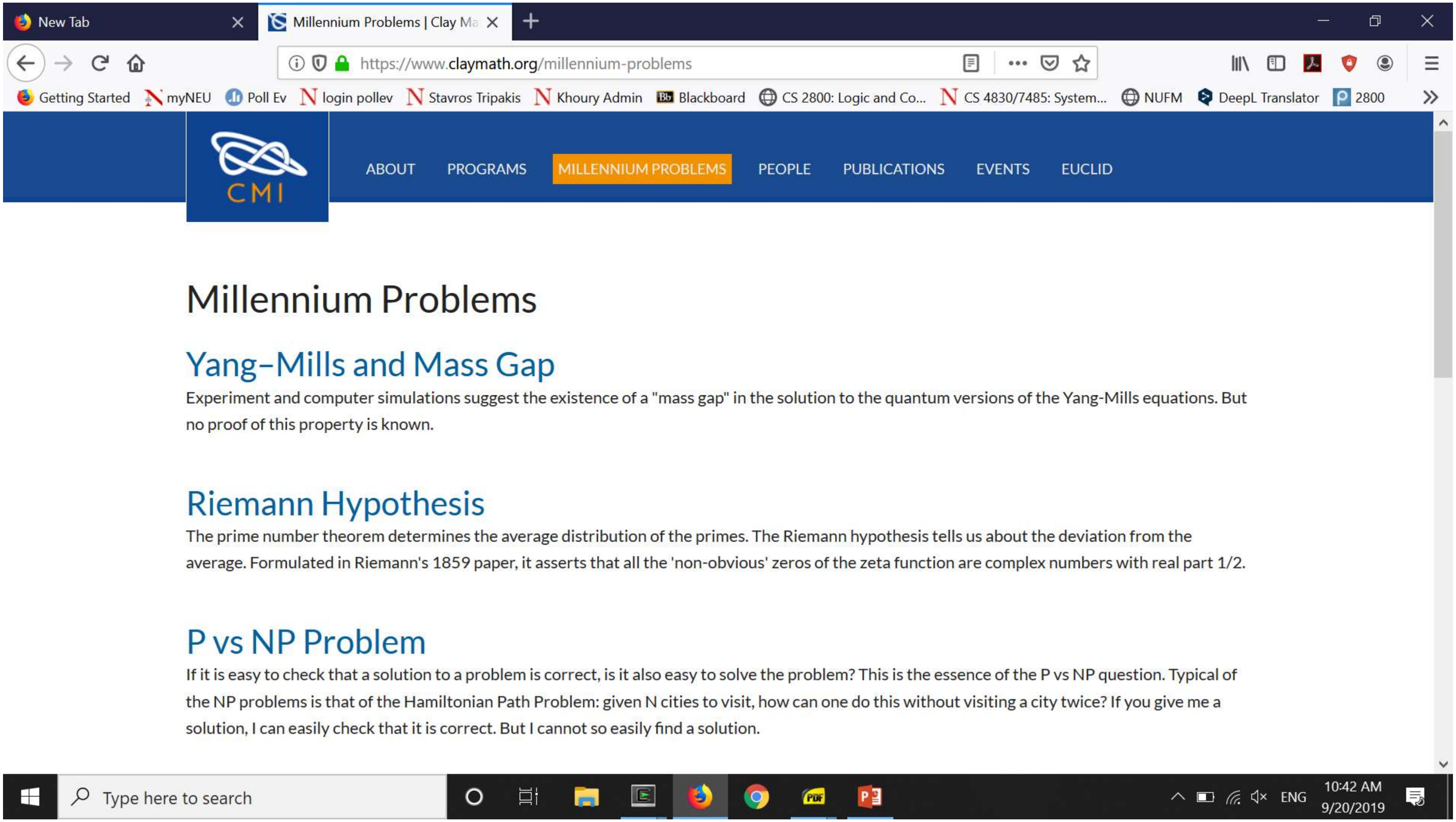
- Is it possible to check automatically whether a given Boolean formula is satisfiable?
- Yes: e.g., build the truth table
- Is this an easy problem? Is the truth-table method a good one? How big is the truth table?
- If the formula has *N* variables, the size of the truth table is 2^N
  – $2^{10} = 1024$
  – $2^{100} = 1267650600228229401496703$
  – $2^{1000} = \ldots$

P vs NP

- P = the class of efficiently (polynomial-time) computable problems
- NP = the class of problems whose solution can be checked efficiently (in polynomial-time)

- Every problem in P is also in NP: P $\subseteq$ NP – why?
  - Note that P and NP are <u>sets of problems</u>
- Most computer scientists believe that NP $\neq$ P
- But nobody has been able to prove that yet!

Clay institute "millennium problems"



The Hamiltonian path problem

Given a graph, does there exist a path that visits all nodes exactly once?

This graph has no Hamiltonian path:

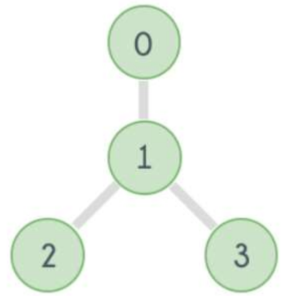


Fig. 1

This graph has two Hamiltonian paths:

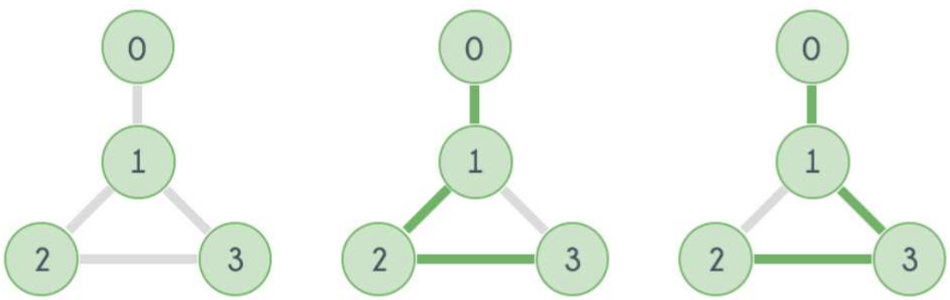


Fig. 2 Fig. 3 Fig. 4

Figure taken from https://www.hackerearth.com/practice/algorithms/graphs/hamiltonian-path/tutorial/

What does NP $\neq$ P mean?

- We know that: P $\subseteq$ NP
- So NP $\neq$ P means P $\subset$ NP: P is a strict subset of NP
- i.e., there exists at least one problem in NP, that is not in P

- We know many problems that are in NP, but we haven't yet found one that we can prove is not in P

The SAT problem

- The Boolean satisfiability (SAT) problem: given a Boolean (propositional logic) formula ϕ, check whether ϕ is satisfiable.

- Fact: **SAT is in NP**. Why?

- If you give me an assignment, I can easily check whether it's a satisfying assignment for ϕ :
  1. Replace all the variables in ϕ by their values as given by the assignment – this is linear in the length of ϕ
  2. Evaluate the resulting formula (which only has constants)
  3. Evaluating a formula is efficient too: polynomial in the length of the formula

The SAT problem

- The Boolean satisfiability (SAT) problem: given a Boolean (propositional logic) formula ϕ, check whether ϕ is satisfiable.

- Theorem [Cook-Levin ~1970]: **SAT is NP-complete**

- What this means is:
  1. SAT is in NP.
  2. Every other problem in NP is no harder than SAT: if you can solve SAT, you can solve any other problem in NP with pretty much the same computational cost.

Reducing Hamiltonian path to SAT

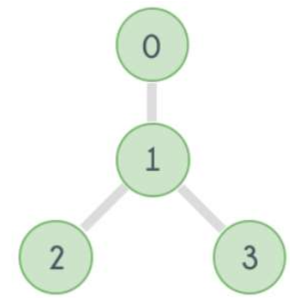
- *If I could solve SAT efficiently (in polynomial time) then I could also solve Hamiltonian path efficiently.*
- How?
- Idea: Given a graph, create a Boolean formula such that the graph has a Hamiltonian path iff the formula is satisfiable.
- How?

Reducing Hamiltonian path to SAT

- Idea: Given a graph, create a Boolean formula such that the graph has a Hamiltonian path iff the formula is satisfiable.
- Let N be the number of nodes in the graph.
- The formula will have N^2 propositional variables: $x_{i,j}$ for $0 \leq i, j \leq N - 1$
- Variable $x_{i,j}$ means: graph node i appears at position j in the Hamiltonian path
- Now all that remains is to encode the constraints of what it means to be a valid Hamiltonian path!

Reducing Hamiltonian path to SAT

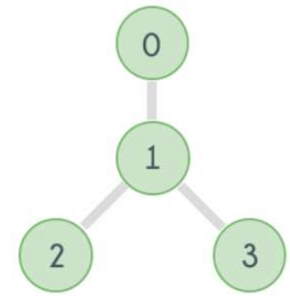
- $x_{i,j}$: graph node i appears at position j in the Hamiltonian path
- Hamiltonian path constraints for our example:
  - Every graph node must appear in the path:



$$\left(x_{0,0} \vee x_{0,1} \vee x_{0,2} \vee x_{0,3}\right) \wedge \left(x_{1,0} \vee x_{1,1} \vee x_{1,2} \vee x_{1,3}\right) \wedge \cdots$$

Reducing Hamiltonian path to SAT

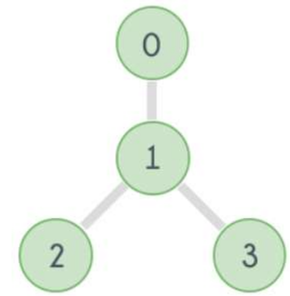
- $x_{i,j}$: graph node i appears at position j in the Hamiltonian path
- Hamiltonian path constraints for our example:
  - Every node appears exactly once:



$$\left(x_{0,0} \rightarrow (\neg x_{0,1} \wedge \neg x_{0,2} \wedge \neg x_{0,3})\right) \wedge \left(x_{0,1} \rightarrow (\neg x_{0,0} \wedge \neg x_{0,2} \wedge \neg x_{0,3})\right) \wedge$$
$$\cdots \wedge \left(x_{1,0} \rightarrow (\neg x_{1,1} \wedge \neg x_{1,2} \wedge \neg x_{1,3})\right) \wedge \cdots$$

Reducing Hamiltonian path to SAT

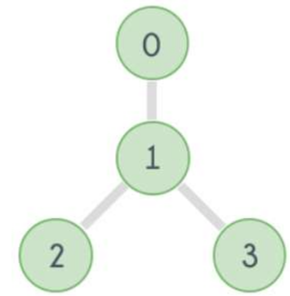
- $x_{i,j}$: graph node i appears at position j in the Hamiltonian path
- Hamiltonian path constraints for our example:
  - Every position in the path is occupied by some node:



$$(x_{0,0} \vee x_{1,0} \vee x_{2,0} \vee x_{3,0}) \wedge (x_{0,1} \vee x_{1,1} \vee x_{2,1} \vee x_{3,1}) \wedge \cdots$$

Reducing Hamiltonian path to SAT

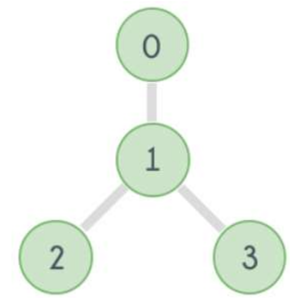
- $x_{i,j}$: graph node i appears at position j in the Hamiltonian path
- Hamiltonian path constraints for our example:
  - Two nodes cannot occupy the same position:



$$\left(x_{0,0} \rightarrow (\neg x_{1,0} \wedge \neg x_{2,0} \wedge \neg x_{3,0})\right)$$
$$\wedge\ \left(x_{0,1} \rightarrow (\neg x_{1,1} \wedge \neg x_{2,1} \wedge \neg x_{3,1})\right) \wedge \cdots$$

Reducing Hamiltonian path to SAT

- $x_{i,j}$: graph node i appears at position j in the Hamiltonian path
- Hamiltonian path constraints for our example:
  - If two nodes are adjacent in the path, then there must be an edge between them in the graph:



$$\neg(x_{0,0} \land x_{3,1}) \land \neg(x_{0,0} \land x_{2,1}) \land \cdots$$

The SAT problem

- Putting it all together:
  1. We don’t know if SAT is in P
  2. The question whether SAT is in P is equivalent to the question whether P = NP
  3. Many combinatorial problems can be reduced to SAT
  4. SAT is a very important problem!

SAT solvers

- Tools that check Boolean satisfiability
- Impressive progress in the last 20 years
- Today: SAT solvers can solve formulas with millions of variables!
- How many assignments does a formula with a million variables have?
  - $2^{1,000,000}$
- See paper "Boolean Satisfiability: From Theoretical Hardness to Practical Success", by Malik and Zhang, CACM 2009

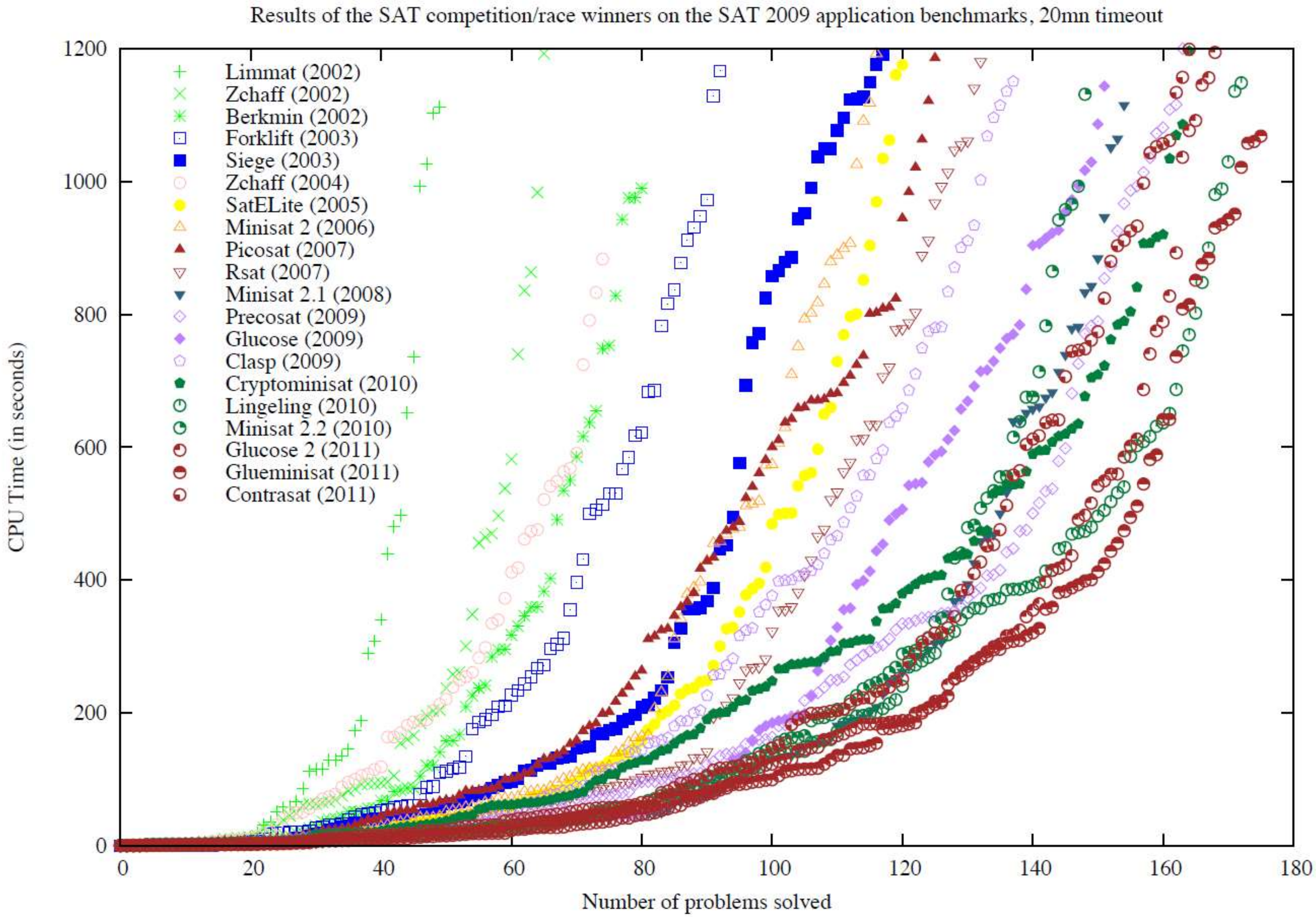


Figure 1: Evolution of the best solvers from 2002 to 2010 on the application benchmarks from the SAT 2009 competition using the cumulative number of problems solved (x axis) within a specific amount of time (y axis). The farther to the right the data points are, the better the solver.

https://www.cs.helsinki.fi/u/mjarvisa/papers/jarvisalo-leberre-roussel-simon.aimag.pdf

Can I use ACL2s as a SAT solver?

- Can I use ACL2s as a completely automated SAT solver, and how?
  - Demo

Quiz

- Suppose you are given a tool that checks VALIDITY
  - Given a formula ϕ, the tool returns YES if ϕ is valid, and NO if ϕ is not valid.
  - Let VALID?(ϕ) be the result of the tool.
- Can you use this tool to check whether ϕ is FALSIFIABLE, and how?
  A. No, it cannot be done
  B. Yes, FALSIFIABLE(ϕ) := (VALID?(ϕ) = NO)
  C. Yes, FALSIFIABLE(ϕ) := (VALID?($\neg\phi$) = NO)
  D. Yes, FALSIFIABLE(ϕ) := (VALID?($\neg\phi$) = YES)

Quiz

- Suppose you are given a tool that checks VALIDITY
  - Given a formula ϕ, the tool returns YES if ϕ is valid, and NO if ϕ is not valid.
  - Let VALID?(ϕ) be the result of the tool.
- Can you use this tool to check whether ϕ is SATISFIABLE, and how?
  A. No, it cannot be done
  B. Yes, SATISFIABLE(ϕ) := (VALID?(ϕ) = YES)
  C. Yes, SATISFIABLE(ϕ) := (VALID?(ϕ) = NO)
  D. Yes, SATISFIABLE(ϕ) := (VALID?($\neg\phi$) = YES)
  E. Yes, SATISFIABLE(ϕ) := (VALID?($\neg\phi$) = NO)

Going further on SAT/SMT solvers

- Going further:
  - http://www.satcompetition.org/
  - https://www.cs.helsinki.fi/u/mjarvisa/papers/jarvisalo-leberre-roussel-simon.aimag.pdf
  - Proceedings of SAT COMPETITION 2018: Solver and Benchmark Descriptions: https://helda.helsinki.fi/bitstream/handle/10138/237063/sc2018_proceedings.pdf?sequence=6
  - Prof. Manolios' CS-4820 class
- SMT solvers:
  - E.g., Z3: https://rise4fun.com/Z3/tutorial/guide

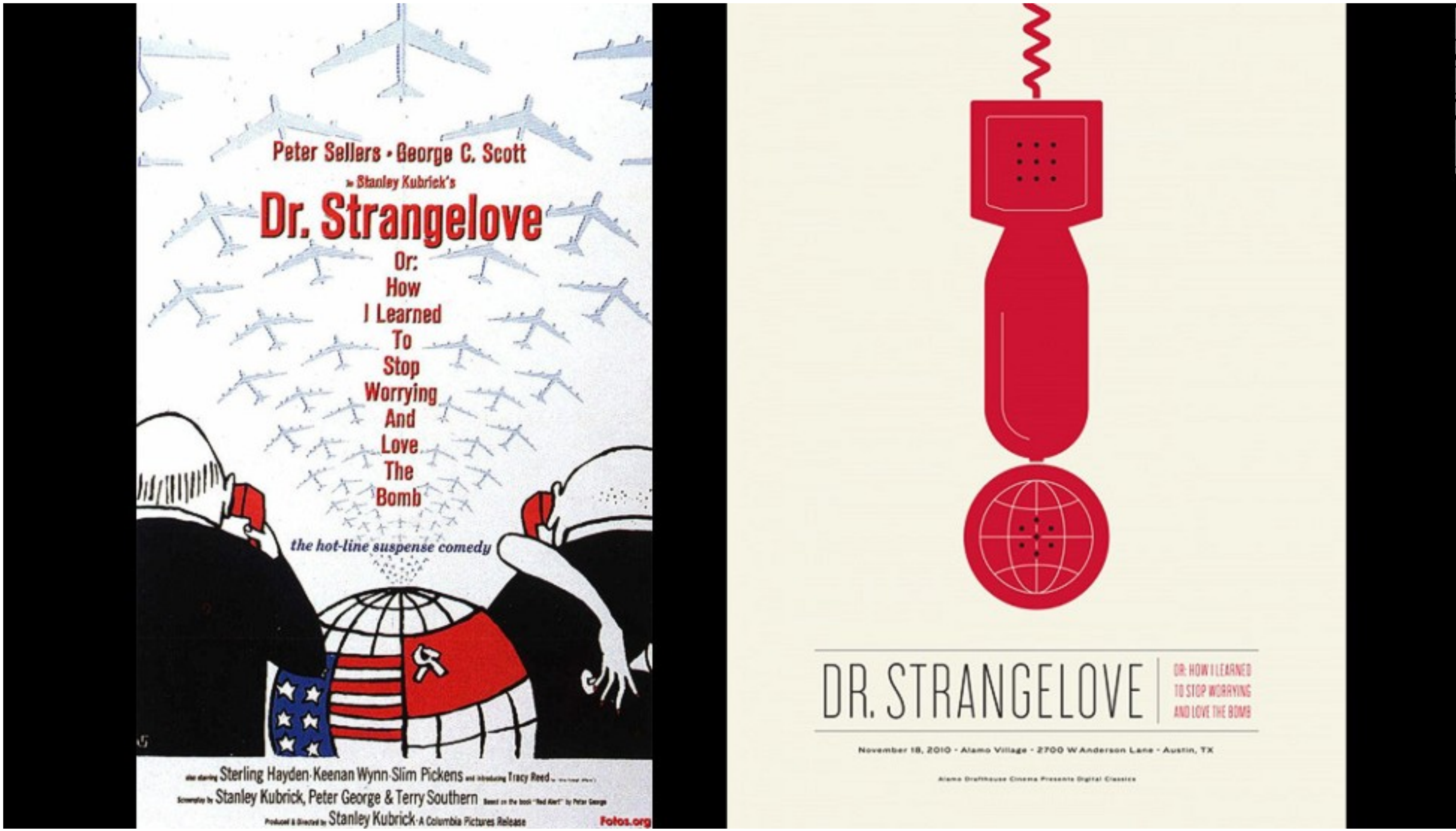
A bit of cryptography – the power of XOR

See extra slides by Pete Manolios

The Power of Xor

- You have probably seen movies with the "red telephone" that connects the Pentagon with the Kremlin
- A classic is Dr. Strangelove

The Red Phone



Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

The Power of Xor

The Power of Xor

- You have probably seen movies with the “red telephone” that connects the Pentagon with the Kremlin
- A classic is Dr. Strangelove
- View https://www.youtube.com/watch?v=VEB-OoUrNuk to 1:24

The Power of Xor

- You have probably seen movies with the “red telephone” that connects the Pentagon with the Kremlin
- A classic is Dr. Strangelove
- View https://www.youtube.com/watch?v=VEB-OoUrNuk to 1:24
- There was no red phone but there was a teletype-based encryption mechanism in place between the US and USSR that used the encryption method we will cover next

Cryptography

Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Cryptography

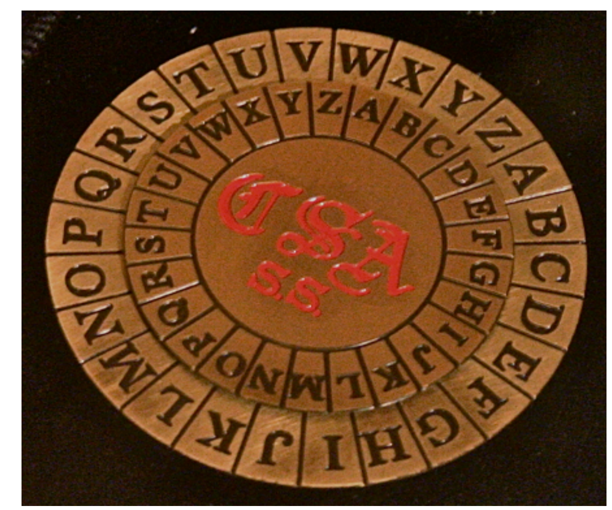
- Goal: secret communication
  - crypto, graphy are Greek for hidden, writing
- Date back to Egypt (1900 BCE)
- Used for commerce, war, love letters, religion, …

Cryptography

- Goal: secret communication
  - crypto, graphy are Greek for hidden, writing
- Date back to Egypt (1900 BCE)
- Used for commerce, war, love letters, religion, …
- Examples
  - Scytale: Archilochus 7th century BC

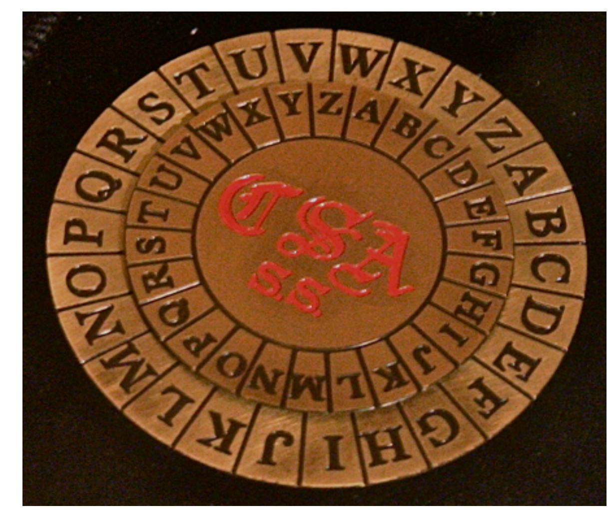
Cryptography

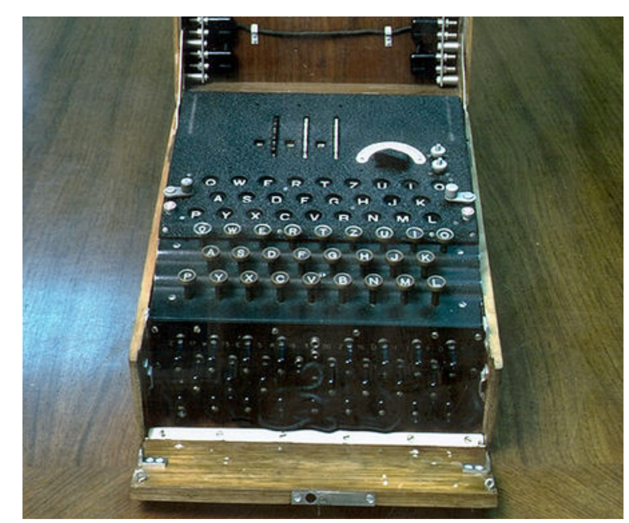
- Goal: secret communication
  - crypto, graphy are Greek for hidden, writing
- Date back to Egypt (1900 BCE)
- Used for commerce, war, love letters, religion, …
- Examples
  - Scytale: Archilochus 7th century BC
  - Caesar Shift Cipher: shift letters by some number
  - Confederate Cipher Disc: Civil War



Cryptography

- Goal: secret communication
  - crypto, graphy are Greek for hidden, writing
- Date back to Egypt (1900 BCE)
- Used for commerce, war, love letters, religion, …
- Examples
  - Scytale: Archilochus 7th century BC
  - Caesar Shift Cipher: shift letters by some number
  - Confederate Cipher Disc: Civil War
  - Enigma: used by Germany in WWII
  - Breaking Enigma shortened the war (Turing et al)





Slides by Pete Manolios for CS2800, Logic & Computation, NU 2019

Exercise

Exercise

- You got the following encrypted message. Decrypt it.
  - Uif tfdsfu pshbojabujpo nffut upojhiu
- Quiz: A. I got it! B: This is hard!

Exercise

- You got the following encrypted message. Decrypt it.
  - Uif tfdsfu pshbojabujpo nffut upojhiu
- Quiz: A. I got it! B: This is hard!
- Frequency analysis: the most common letters are e, t
  - u: 6
  - f: 5

Exercise

- You got the following encrypted message. Decrypt it.
  - Uif tfdsfu pshbojabujpo nffut upojhiu
- Quiz: A. I got it! B: This is hard!
- Frequency analysis: the most common letters are e, t
  - u: 6
  - f: 5
- Hint: Caesar Shift Cipher: shift letters by some number
  - Shift by 16, 1

Exercise

- You got the following encrypted message. Decrypt it.
  - Uif tfdsfu pshbojabujpo nffut upojhiu
- Quiz: A. I got it! B: This is hard!
- Frequency analysis: the most common letters are e, t
  - u: 6
  - f: 5
- Hint: Caesar Shift Cipher: shift letters by some number
  - Shift by 16, 1
- Answer? The secret organization meets tonight

One-Time Pad

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0’s & 1’s. Any ideas?

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0’s & 1’s. Any ideas?
- Alice and Bob agree on a secret, a sequence of random 0’s & 1’s

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0’s & 1’s. Any ideas?
- Alice and Bob agree on a secret, a sequence of random 0’s & 1’s
- To send message m, Alice xor’s m with s, the secret: $c = m \oplus s$

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0’s & 1’s. Any ideas?
- Alice and Bob agree on a secret, a sequence of random 0’s & 1’s
- To send message m, Alice xor’s m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor’s it with s: $c \oplus s = m$

One-Time Pad

- Allow us to encrypt messages with “perfect” secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0’s & 1’s. Any ideas?
- Alice and Bob agree on a secret, a sequence of random 0’s & 1’s
- To send message m, Alice xor’s m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor’s it with s: $c \oplus s = m$
- Example: m=1001000100011…

  s=1101011010111…

  c=0100011110100…

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
  - Compare: RSA can be broken, with enough computational resources
- A message is a sequence of bits, say 0's & 1's. Any ideas?
- Alice and Bob agree on a secret, a sequence of random 0's & 1's
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor's it with s: $c \oplus s = m$
- Example: m=1001000100011…

  s=1101011010111…

  c=0100011110100…

  $c \oplus s$=1001000100011…

One-Time Pad

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c= m\oplus s$
- When Bob gets c, he xor's it with s: $c\oplus s = m$

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor's it with s: $c \oplus s = m$
- Why is it "perfect"?
  - If we have c, the encrypted msg, then for every, m, an arbitrary msg of the same length, there is some secret, s, that when used to decode c yields m

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor's it with s: $c \oplus s = m$
- Why is it "perfect"?
  - If we have c, the encrypted msg, then for every, m, an arbitrary msg of the same length, there is some secret, s, that when used to decode c yields m
- Example: `c=0100011110100…` (the same c as before)

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: c= m⊕s
- When Bob gets c, he xor's it with s: c⊕s = m
- Why is it "perfect"?
  - If we have c, the encrypted msg, then for every, m, an arbitrary msg of the same length, there is some secret, s, that when used to decode c yields m
- Example: `c=0100011110100…` (the same c as before)

  `m=0110111011100…` (a different m)

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor's it with s: $c \oplus s = m$
- Why is it "perfect"?
  - If we have c, the encrypted msg, then for every, m, an arbitrary msg of the same length, there is some secret, s, that when used to decode c yields m
- Example: `c=0100011110100…` (the same c as before)

  `m=0110111011100…` (a different m)

  `s=0010100101000…` (the corresponding s)

One-Time Pad

- Allow us to encrypt messages with "perfect" secrecy
  - If an adversary intercepts an encoded message, they gain no information, except for an upper bound on the message length
- To send message m, Alice xor's m with s, the secret: $c = m \oplus s$
- When Bob gets c, he xor's it with s: $c \oplus s = m$
- Why is it "perfect"?
  - If we have c, the encrypted msg, then for every, m, an arbitrary msg of the same length, there is some secret, s, that when used to decode c yields m
- Example: `c=0100011110100…` (the same c as before)

  `m=0110111011100…` (a different m)

  `s=0010100101000…` (the corresponding s)

  $c \oplus s$`=0110111011100…`