

# Codes and Cryptography

Jorge L. Villar

MAMME, Fall 2015

**PART X**

# Outline

1 Algorithmic Complexity

2 Complexity Classes

# Bounded Adversaries

Bounding adversarial resources:

- **Space** (memory).
- **Time** (running time).

# Bounded Adversaries

Bounding adversarial resources:

- **Space** (memory).
- **Time** (running time).

# Bounded Adversaries

Bounding adversarial resources:

- **Space** (memory). Not a big deal. Memory is cheap, but read/write operations take time!
- **Time** (running time).

# Bounded Adversaries

Bounding adversarial resources:

- **Space** (memory). Not a big deal. Memory is cheap, but read/write operations take time!
- **Time** (running time). It actually depends on the computing device (e.g., parallel processors).

# Bounded Adversaries

Bounding adversarial resources:

- **Space** (memory). Not a big deal. Memory is cheap, but read/write operations take time!
- **Time** (running time). It actually depends on the computing device (e.g., parallel processors).

Fix a computational model or machine: **Turing Machine**

# Turing Machine

A model for mechanic computation, consisting of

- A finite automaton
  
- A storage device (“tape”)

# Turing Machine

A model for mechanic computation, consisting of

- A finite automaton
  - state information  $s \in S \cup \{\text{init}, \text{halt}\}$
  - transition function  $f$
- A storage device (“tape”)

# Turing Machine

A model for mechanic computation, consisting of

- A finite automaton
  - state information  $s \in S \cup \{\text{init}, \text{halt}\}$
  - transition function  $f$
- A storage device (“tape”)
  - a “tape”, or a sequence of cells containing ‘0’, ‘1’ or ‘blank’
  - a read/write moving head on a specific cell in the tape

# Turing Machine

A model for mechanic computation, consisting of

- A finite automaton
  - state information  $s \in S \cup \{\text{init}, \text{halt}\}$
  - transition function  $f$
- A storage device (“tape”)
  - a “tape”, or a sequence of cells containing ‘0’, ‘1’ or ‘blank’
  - a read/write moving head on a specific cell in the tape

Computation step:  $(s, wr, mv) \leftarrow f(s, c)$

$wr \in \{\text{write}_0, \text{write}_1, \text{erase}\}$

$mv \in \{\text{move\_left}, \text{move\_right}\}$

$c =$  contents of the current cell

# Turing Machine

A model for mechanic computation, consisting of

- A finite automaton
  - state information  $s \in S \cup \{\text{init}, \text{halt}\}$
  - transition function  $f$
- A storage device (“tape”)
  - a “tape”, or a sequence of cells containing ‘0’, ‘1’ or ‘blank’
  - a read/write moving head on a specific cell in the tape

Computation step:  $(s, wr, mv) \leftarrow f(s, c)$

$wr \in \{\text{write}_0, \text{write}_1, \text{erase}\}$

$mv \in \{\text{move\_left}, \text{move\_right}\}$

$c =$  contents of the current cell

Input/Output of the Turing Machine is the content of the tape in the `init/halt` state

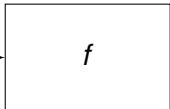
# Turing Machine

state



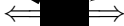
transition func.

$(s, c)$

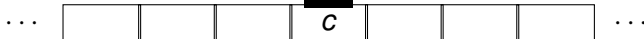


$(s^+, wr, mv) = f(s, c)$

read/write head



tape



▶ example...

# Turing Machine

state

$s$

transition func.

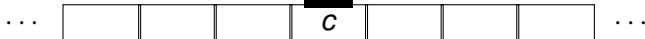
$(s, c)$

$f$

$(s^+, wr, mv) = f(s, c)$

read/write head

tape



$s \in S \cup \{\text{init}, \text{halt}\}$

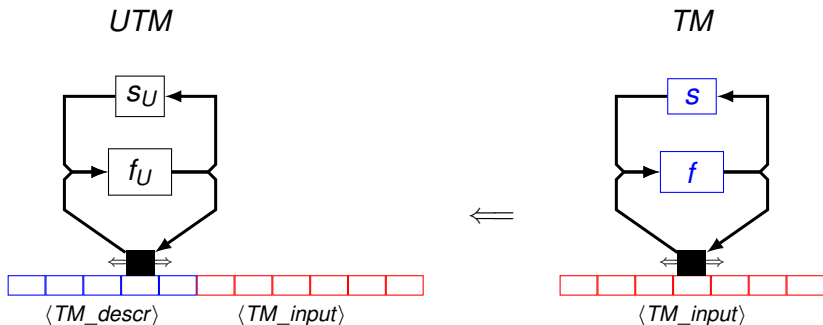
$wr \in \{\text{write}_0, \text{write}_1, \text{erase}\}$

$mv \in \{\text{move\_left}, \text{move\_right}\}$

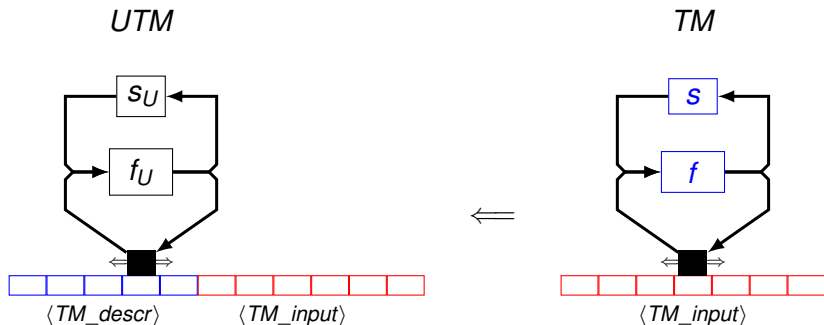
▶ example...



# Universal Turing Machine



# Universal Turing Machine



It can simulate any known (classical) computing device with reasonable efficiency

# Computational Problems

**Parameterized Problem Families:** A set  $P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  of problem instances parameterized by a size parameter  $\ell \in \mathbb{Z}^+$  (e.g., length of the instance description)

# Computational Problems

**Parameterized Problem Families:** A set  $P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  of problem instances parameterized by a size parameter  $\ell \in \mathbb{Z}^+$  (e.g., length of the instance description) (Related to the security parameter in cryptography)

# Computational Problems

**Parameterized Problem Families:** A set  $P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  of problem instances parameterized by a size parameter  $\ell \in \mathbb{Z}^+$  (e.g., length of the instance description) (Related to the security parameter in cryptography)

- **Search Problems:** Unique correct answer among a large set of possibilities
- **Flexible Problems:** Several correct answers among a large set of possibilities
- **Decision Problems:** The answer is just a single bit (i.e., 'YES' or 'NO')

# Decision Problems vs. Search Problems (I)

Most 'natural' problems are stated as Search Problems  
... but the theory of Decision Problems is cleaner

# Decision Problems vs. Search Problems (I)

Most 'natural' problems are stated as Search Problems  
... but the theory of Decision Problems is cleaner  
However, complexity of Decision Problems and of Search Problems can be related

# Decision Problems vs. Search Problems (I)

Most 'natural' problems are stated as Search Problems

... but the theory of Decision Problems is cleaner

However, complexity of Decision Problems and of Search Problems can be related

We can describe a Decisional Problem Family  $P$  as the set of problem 'YES'-instances (encoded as binary strings):

$$L_P = \{x \in P : \text{sol}(x) = \text{'YES'}\} \subset \{0, 1\}^*$$

Solving  $P$  means checking membership to the **language**  $L_P$

# Decision Problems vs. Search Problems (II)

Given a search problem family  $P$ , we can define the language

$$L_P = \{(x, y) : x \in P, y \in \text{sol}(x)\}$$

# Decision Problems vs. Search Problems (II)

Given a search problem family  $P$ , we can define the language

$$L_P = \{(x, y) : x \in P, y \in \text{sol}(x)\}$$

Checking membership to  $L_P$  is not harder than solving  $P$  (if the solution  $y$  to  $x$  is unique)

## Decision Problems vs. Search Problems (II)

Given a search problem family  $P$ , we can define the language

$$L_P = \{(x, y) : x \in P, y \in \text{sol}(x)\}$$

Checking membership to  $L_P$  is not harder than solving  $P$  (if the solution  $y$  to  $x$  is unique)

REDUCTION: Given  $(x, y)$ , solve the instance  $x$  and check whether  $y = \text{sol}(x)$

## Decision Problems vs. Search Problems (II)

Given a search problem family  $P$ , we can define the language

$$L_P = \{(x, y) : x \in P, y \in \text{sol}(x)\}$$

Checking membership to  $L_P$  is not harder than solving  $P$  (if the solution  $y$  to  $x$  is unique)

REDUCTION: Given  $(x, y)$ , solve the instance  $x$  and check whether  $y = \text{sol}(x)$

but sometimes verifying that  $y$  solves instance  $x$  is easier than directly finding  $y$  from  $x$ , e.g.,

$$P = \{\text{"find a nontrivial factorization of } n" : n \in \mathbb{Z}^+\},$$

$$L_P = \{(pq, p, q) : p, q > 1\}$$

# Problem Hardness (I)

How to measure the efficiency (complexity) of an algorithm?

# Problem Hardness (I)

How to measure the efficiency (complexity) of an algorithm?

**Running time** (computation steps in a Turing Machine)

# Problem Hardness (I)

How to measure the efficiency (complexity) of an algorithm?

**Running time** (computation steps in a Turing Machine)

... but it depends on the particular problem instance (input) and on the instance description size  $\ell$

# Problem Hardness (I)

How to measure the efficiency (complexity) of an algorithm?

**Running time** (computation steps in a Turing Machine)

... but it depends on the particular problem instance (input) and on the instance description size  $\ell$

- **Worst case time** for every fixed  $\ell$
- **Average time**, or expected time for a randomly chosen instance of size  $\ell$

# Problem Hardness (I)

How to measure the efficiency (complexity) of an algorithm?

**Running time** (computation steps in a Turing Machine)

... but it depends on the particular problem instance (input) and on the instance description size  $\ell$

- **Worst case time** for every fixed  $\ell$   
Easy to analyze but not always meaningful: e.g. only a few instances taking a huge time
- **Average time**, or expected time for a randomly chosen instance of size  $\ell$   
Depends on the probability distribution and does not give an upper bound of the time

# Problem Hardness (II)

- **Concrete analysis.** Useful to make the best choice of an algorithm solving a specific problem instance, based on the value of  $\ell$

# Problem Hardness (II)

- **Concrete analysis.** Useful to make the best choice of an algorithm solving a specific problem instance, based on the value of  $\ell$
- **Asymptotic analysis.** Measures how well an algorithm scales with the input size, but cannot be used to compare two algorithms for a precise value of  $\ell$ .

# Problem Hardness (II)

- **Concrete analysis.** Useful to make the best choice of an algorithm solving a specific problem instance, based on the value of  $\ell$
- **Asymptotic analysis.** Measures how well an algorithm scales with the input size, but cannot be used to compare two algorithms for a precise value of  $\ell$ .

Meaningful in cryptography, as one can ever choose the size of the keys, with a reasonable impact on efficiency

# Problem Hardness (II)

- **Concrete analysis.** Useful to make the best choice of an algorithm solving a specific problem instance, based on the value of  $\ell$
- **Asymptotic analysis.** Measures how well an algorithm scales with the input size, but cannot be used to compare two algorithms for a precise value of  $\ell$ .

Meaningful in cryptography, as one can ever choose the size of the keys, with a reasonable impact on efficiency

Uniform vs. Non-Uniform complexity notions

## Problem Hardness (II)

- **Concrete analysis.** Useful to make the best choice of an algorithm solving a specific problem instance, based on the value of  $\ell$
- **Asymptotic analysis.** Measures how well an algorithm scales with the input size, but cannot be used to compare two algorithms for a precise value of  $\ell$ .

Meaningful in cryptography, as one can ever choose the size of the keys, with a reasonable impact on efficiency

### Uniform vs. Non-Uniform complexity notions

- Uniform means a single algorithm or Turing Machine for all values of  $\ell$
- Non-Uniform means different algorithms (circuits) can be used for different values of  $\ell$

# Outline

1 Algorithmic Complexity

2 Complexity Classes

$\mathcal{P}$ 

**The class  $\mathcal{P}$ :** The worst-case running time of the best algorithm **solving all problem instances** in a family

$P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  is **upper bounded by a polynomial** in  $\ell$

$\mathcal{P}$ 

**The class  $\mathcal{P}$ :** The worst-case running time of the best algorithm **solving all problem instances** in a family

$P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  is **upper bounded by a polynomial** in  $\ell$

**NOTATION:** **poly** denotes the set of nonnegative functions  $f : \mathbb{Z}^+ \rightarrow \mathbb{R}$  upper bounded by a polynomial. Addition, multiplication and composition are internal operations.

$\mathcal{P}$ 

**The class  $\mathcal{P}$ :** The worst-case running time of the best algorithm **solving all problem instances** in a family

$P = \bigcup_{\ell \in \mathbb{Z}^+} P_\ell$  is **upper bounded by a polynomial** in  $\ell$

**NOTATION:** **poly** denotes the set of nonnegative functions  $f : \mathbb{Z}^+ \rightarrow \mathbb{R}$  upper bounded by a polynomial. Addition, multiplication and composition are internal operations.

### Definition ( $\mathcal{P}$ )

A decision problem family  $P$  is in  $\mathcal{P}$  if there exists a Turing Machine  $TM$  such that  $\forall x \in P$ ,  $TM(x) = \text{sol}(x)$ , and

$$\max_{x \in P_\ell} \text{time}(TM, x) \in \mathbf{poly}(\ell)$$

$\mathcal{P}$ Definition ( $\mathcal{P}$ )

A decision problem family  $P$  is in  $\mathcal{P}$  if there exists a Turing Machine  $TM$  such that  $\forall x \in P$ ,  $TM(x) = \text{sol}(x)$ , and

$$\max_{x \in P_\ell} \text{time}(TM, x) \in \mathbf{poly}(\ell)$$

Definition ( $\mathcal{P}$ )

A language  $L$  is in  $\mathcal{P}$  if there exists a Turing Machine  $TM$  such that

$$\forall x \in L, TM(x) = 1$$

$$\forall x \in \{0, 1\}^* \setminus L, TM(x) = 0$$

$$\max_{x \in \{0, 1\}^\ell} \text{time}(TM, x) \in \mathbf{poly}(\ell)$$

$\mathcal{P}$ 

Typical easy computational problems (e.g., basic arithmetic operations) are solved with polynomial-time algorithms

$\mathcal{P}$ 

Typical easy computational problems (e.g., basic arithmetic operations) are solved with polynomial-time algorithms

Polynomial-time algorithms form a consistent set, they can be combined in a number of ways producing more polynomial-time algorithms, e.g.

$\mathcal{P}$ 

Typical easy computational problems (e.g., basic arithmetic operations) are solved with polynomial-time algorithms

Polynomial-time algorithms form a consistent set, they can be combined in a number of ways producing more polynomial-time algorithms, e.g.

- polynomially long sequence
- polynomially bounded iteration
- composition (subroutines)
- constant depth recursion

$\mathcal{NP}$ 

**The class  $\mathcal{NP}$ :** The worst-case time of the best algorithm solving all **decision** problem 'YES'-instances in  $P$ , **with an auxiliary input**, is upper bounded by a polynomial in  $\ell$

The auxiliary input can be seen as a hint

$\mathcal{NP}$ 

**The class  $\mathcal{NP}$ :** The worst-case time of the best algorithm solving all **decision** problem 'YES'-instances in  $P$ , **with an auxiliary input**, is upper bounded by a polynomial in  $\ell$

The auxiliary input can be seen as a hint

**Definition ( $\mathcal{NP}$ )**

A language  $L$  is in  $\mathcal{NP}$  if there exists a integer-valued function  $q \in \mathbf{poly}$  and a Turing Machine  $TM$  such that

$$\forall x \in \{0, 1\}^{\ell}, x \in L \Leftrightarrow \exists w \in \{0, 1\}^{q(\ell)} TM(x, w) = 1$$

$$\max_{x \in \{0, 1\}^{\ell}, w \in \{0, 1\}^{q(\ell)}} \text{time}(TM, x, w) \in \mathbf{poly}(\ell)$$

$\mathcal{NP}$

Most interesting problems in cryptography are in  $\mathcal{NP}$

$\mathcal{NP}$ 

Most interesting problems in cryptography are in  $\mathcal{NP}$

Actually  $TM$  just verifies efficiently that some  $x \in L$  from a suitable witness  $w$ , **but there is no witness for  $x \in \{0, 1\}^* \setminus L$**

$\mathcal{NP}$ 

Most interesting problems in cryptography are in  $\mathcal{NP}$

Actually  $TM$  just verifies efficiently that some  $x \in L$  from a suitable witness  $w$ , **but there is no witness for  $x \in \{0, 1\}^* \setminus L$**

$$L \in \text{co-}\mathcal{NP} \Leftrightarrow \{0, 1\}^* \setminus L \in \mathcal{NP}$$

$\mathcal{NP}$ 

Most interesting problems in cryptography are in  $\mathcal{NP}$

Actually  $TM$  just verifies efficiently that some  $x \in L$  from a suitable witness  $w$ , **but there is no witness for  $x \in \{0, 1\}^* \setminus L$**

$$L \in \text{co-}\mathcal{NP} \Leftrightarrow \{0, 1\}^* \setminus L \in \mathcal{NP}$$

Trivially,  $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$

$\mathcal{NP}$ 

Most interesting problems in cryptography are in  $\mathcal{NP}$

Actually  $TM$  just verifies efficiently that some  $x \in L$  from a suitable witness  $w$ , **but there is no witness for  $x \in \{0, 1\}^* \setminus L$**

$$L \in \text{co-}\mathcal{NP} \Leftrightarrow \{0, 1\}^* \setminus L \in \mathcal{NP}$$

Trivially,  $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$

... but it is not known whether  $\mathcal{P} = \mathcal{NP}$

$\mathcal{NP}$ 

Most interesting problems in cryptography are in  $\mathcal{NP}$

Actually  $TM$  just verifies efficiently that some  $x \in L$  from a suitable witness  $w$ , **but there is no witness for  $x \in \{0, 1\}^* \setminus L$**

$$L \in \text{co-}\mathcal{NP} \Leftrightarrow \{0, 1\}^* \setminus L \in \mathcal{NP}$$

Trivially,  $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$

... but it is not known whether  $\mathcal{P} = \mathcal{NP}$

**$\mathcal{P} = \mathcal{NP}$  would imply that most cryptographic systems are weak!**

# Randomized Algorithms

A **randomized algorithm** can use random values during the computation (even when the problem instance description is deterministic)

# Randomized Algorithms

A **randomized algorithm** can use random values during the computation (even when the problem instance description is deterministic)

Practice tells us that randomized algorithms can solve problems more efficiently than deterministic ones

# Randomized Algorithms

A **randomized algorithm** can use random values during the computation (even when the problem instance description is deterministic)

Practice tells us that randomized algorithms can solve problems more efficiently than deterministic ones

... but everything in the algorithm (output, running time) can depend on the randomness,

# Randomized Algorithms

A **randomized algorithm** can use random values during the computation (even when the problem instance description is deterministic)

Practice tells us that randomized algorithms can solve problems more efficiently than deterministic ones

... but everything in the algorithm (output, running time) can depend on the randomness, and typically we have to tolerate some small error probability

# Randomized Algorithms

A **randomized algorithm** can use random values during the computation (even when the problem instance description is deterministic)

Practice tells us that randomized algorithms can solve problems more efficiently than deterministic ones

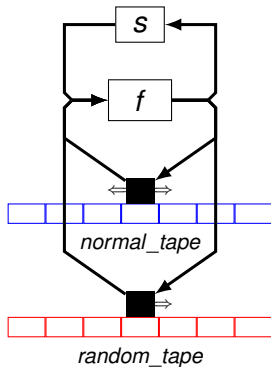
... but everything in the algorithm (output, running time) can depend on the randomness, and typically we have to tolerate some small error probability

The model: **Probabilistic Turing Machine**

- The same as the plain Turing Machine with an extra read-only tape initialized with random (binary) data
- At every computation step, the current random bit is an input to the transition function

# Probabilistic Turing Machine

*PTM*



Computation step:

$$(s, wr, mv, mvr) \leftarrow f(s, c, r)$$

$wr \in \{\text{write}_0, \text{write}_1, \text{erase}\}$

$mv \in \{\text{move\_left}, \text{move\_right}\}$

$mvr \in \{\text{keep}, \text{move\_right}\}$

$c$  = contents of the current normal cell

$r$  = contents of the current random cell

# *BPP*

**The class *BPP*:** The worst-case time of the best randomized algorithm **correctly guessing most** decision problem 'YES' and 'NO'-instances in  $P$ , is upper bounded by a polynomial in  $\ell$

# BPP

**The class  $BPP$ :** The worst-case time of the best randomized algorithm **correctly guessing most** decision problem 'YES' and 'NO'-instances in  $P$ , is upper bounded by a polynomial in  $\ell$

## Definition ( $BPP$ )

A language  $L$  is in  $BPP$  if there exists a integer-valued function  $p \in \mathbf{poly}$  and a Probabilistic Turing Machine  $PTM$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > 2/3$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < 1/3$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

# BPP

**The class BPP:** The worst-case time of the best randomized algorithm **correctly guessing most** decision problem 'YES' and 'NO'-instances in  $P$ , is upper bounded by a polynomial in  $\ell$

## Definition (BPP)

A language  $L$  is in  $BPP$  if there exists a integer-valued function  $p \in \mathbf{poly}$  and a Probabilistic Turing Machine  $PTM$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > 2/3$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < 1/3$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

The thresholds  $2/3$  and  $1/3$  are arbitrarily fixed

# Probability Amplification by Repetition

Given  $PTM$  and  $\alpha : \mathbb{Z}^+ \rightarrow (0, 1/2)$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > \frac{1}{2} + \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < \frac{1}{2} - \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

we can build  $PTM_n$  for any (odd)  $n \in \mathbb{Z}^+$  that on the input of  $x$  it runs  $PTM(x)$   $n$  times, and decides its output by majority voting.

# Probability Amplification by Repetition

Given  $PTM$  and  $\alpha : \mathbb{Z}^+ \rightarrow (0, 1/2)$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > \frac{1}{2} + \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < \frac{1}{2} - \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

we can build  $PTM_n$  for any (odd)  $n \in \mathbb{Z}^+$  that on the input of  $x$  it runs  $PTM(x)$   $n$  times, and decides its output by majority voting. Then, by Chernoff's bound,

[▶ details...](#)

$$\alpha_n(\ell) > \frac{1}{2} - \left( \sqrt{1 - 4\alpha(\ell)^2} \right)^n \quad \text{and} \quad p_n(\ell) = np(\ell)$$

# Probability Amplification by Repetition

Given  $PTM$  and  $\alpha : \mathbb{Z}^+ \rightarrow (0, 1/2)$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > \frac{1}{2} + \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < \frac{1}{2} - \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

we can build  $PTM_n$  for any (odd)  $n \in \mathbb{Z}^+$  that on the input of  $x$  it runs  $PTM(x)$   $n$  times, and decides its output by majority voting. Then, by Chernoff's bound,

[▶ details...](#)

$$\alpha_n(\ell) > \frac{1}{2} - \left(\sqrt{1 - 4\alpha(\ell)^2}\right)^n \quad \text{and} \quad p_n(\ell) = np(\ell)$$

In particular,

$$\alpha(\ell) = 1/6, n > 12\ell \quad \Rightarrow \quad \frac{1}{2} + \alpha_n(\ell) > 1 - 2^{-\ell}, \quad \frac{1}{2} - \alpha_n(\ell) < 2^{-\ell}$$

# Probability Amplification by Repetition

Given  $PTM$  and  $\alpha : \mathbb{Z}^+ \rightarrow (0, 1/2)$  such that

$$\forall x \in \{0, 1\}^\ell, x \in L \Rightarrow \Pr[PTM(x) = 1] > \frac{1}{2} + \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, x \notin L \Rightarrow \Pr[PTM(x) = 1] < \frac{1}{2} - \alpha(\ell)$$

$$\forall x \in \{0, 1\}^\ell, \Pr[\text{time}(PTM, x) < p(\ell)] = 1$$

we can build  $PTM_n$  for any (odd)  $n \in \mathbb{Z}^+$  that on the input of  $x$  it runs  $PTM(x)$   $n$  times, and decides its output by majority voting. Then, by Chernoff's bound,

[details...](#)

$$\alpha_n(\ell) > \frac{1}{2} - \left( \sqrt{1 - 4\alpha(\ell)^2} \right)^n \quad \text{and} \quad p_n(\ell) = np(\ell)$$

In particular,

$$\alpha(\ell) = 1/6, n > 12\ell \Rightarrow \frac{1}{2} + \alpha_n(\ell) > 1 - 2^{-\ell}, \frac{1}{2} - \alpha_n(\ell) < 2^{-\ell}$$

The same for any  $q \in \mathbf{poly}$  if  $\alpha(\ell) = 1/q(\ell)$  and  $n > \frac{3}{8}\ell(q(\ell))^2$

# Bounded Computational Power

NOTATION: **negl** denotes the set of nonnegative functions  $f : \mathbb{Z}^+ \rightarrow \mathbb{R}$  vanishing faster than the inverse of any nonzero polynomial

$$f \in \mathbf{negl} \Leftrightarrow \forall c \in \mathbb{Z}^+ \lim_{t \rightarrow +\infty} f(t)t^{-c} = 0$$

Addition, multiplication and multiplication by functions in **poly** preserve negligible functions

# Bounded Computational Power

The previous considerations motivate the following definitions for infeasible computation:

# Bounded Computational Power

The previous considerations motivate the following definitions for infeasible computation:

## Definition (Hard-on-Average Search Problem)

A search problem is considered hard if any Probabilistic Turing Machine can solve a **random** problem instance in polynomial time only with a **negligible probability**

# Bounded Computational Power

The previous considerations motivate the following definitions for infeasible computation:

## Definition (Hard-on-Average Search Problem)

A search problem is considered hard if any Probabilistic Turing Machine can solve a **random** problem instance in polynomial time only with a **negligible probability**

## Definition (Hard-on-Average Decision Problem)

A decision problem is considered hard if any Probabilistic Turing Machine can solve a **random** problem instance in polynomial time only with a **negligible advantage** with respect to a random guess

# Bounded Computational Power

The previous considerations motivate the following definitions for infeasible computation:

## Definition (Hard-on-Average Search Problem)

A search problem is considered hard if any Probabilistic Turing Machine can solve a **random** problem instance in polynomial time only with a **negligible probability**

## Definition (Hard-on-Average Decision Problem)

A decision problem is considered hard if any Probabilistic Turing Machine can solve a **random** problem instance in polynomial time only with a **negligible advantage** with respect to a random guess

**NOTE:** feasible  $\neq$  efficient

# Codes and Cryptography

Jorge L. Villar

MAMME, Fall 2015

**END OF PART X**

# A Sample Turing Machine

Given the binary encoding of a nonnegative integer

... 

--

$x_0$	$x_1$
-------	-------

 ... 

1	
---	--

 ...

compute the binary encoding of  $y = x + 1$

... 

--

$y_0$	$y_1$
-------	-------

 ... 

1	
---	--

 ...

IDEA: flip all bits until the first 0 (included), then rewind and halt.

# A Sample Turing Machine

Given the binary encoding of a nonnegative integer

...   x<sub>0</sub> x<sub>1</sub> ... 1   ...

compute the binary encoding of  $y = x + 1$

...   y<sub>0</sub> y<sub>1</sub> ... 1   ...

**IDEA:** flip all bits until the first 0 (included), then rewind and halt.

**States:**  $S = \{\text{rewind}\}$

**Transition function table:**

$(\text{init}, 0) \mapsto (\text{rewind}, \text{write}_1, \text{move\_left})$

$(\text{init}, 1) \mapsto (\text{init}, \text{write}_0, \text{move\_right})$

$(\text{init}, \text{blank}) \mapsto (\text{rewind}, \text{write}_1, \text{move\_left})$

$(\text{rewind}, 0) \mapsto (\text{rewind}, \text{write}_0, \text{move\_left})$

$(\text{rewind}, 1) \mapsto (\text{rewind}, \text{write}_1, \text{move\_left})$

$(\text{rewind}, \text{blank}) \mapsto (\text{halt}, \text{erase}, \text{move\_right})$

◀ go back...

# Probability Amplification: Details

Fix  $x \in \{0, 1\}^\ell \cap L$  and define  $V_i = TM(x)$  for the  $i$ -th run.  
 $V_i \in \{0, 1\}$  and  $\Pr[V_i = 1] = 1/2 + \epsilon_x$  for some  $\epsilon_x \in (\alpha(\ell), 1/2]$ .  
 Moreover,  $V_1, \dots, V_n$  are independent. Let  $V = V_1 + \dots + V_n$ .  
 For any  $t > 0$ , by Markov's inequality

$$\begin{aligned} \Pr[V \leq n/2] &= \Pr[e^{t(n/2-V)} \geq 1] \leq \mathbb{E}[e^{t(n/2-V)}] = \mathbb{E}[e^{t(1/2-V_1)}]^n = \\ &= \left( \left( \frac{1}{2} - \epsilon_x \right) e^{t/2} + \left( \frac{1}{2} + \epsilon_x \right) e^{-t/2} \right)^n \end{aligned}$$

$$\text{Therefore } \Pr[V \leq n/2] \leq \inf_{t>0} \left( \frac{1-2\epsilon_x}{2} e^{t/2} + \frac{1+2\epsilon_x}{2} e^{-t/2} \right)^n$$

The minimum value is attained when  $e^t = \frac{1+2\epsilon_x}{1-2\epsilon_x}$  and then

$$\Pr[V > n/2] \geq 1 - \left( \sqrt{1 - 4\epsilon_x^2} \right)^n \geq 1 - \left( \sqrt{1 - 4\alpha(\ell)^2} \right)^n$$