Complexity-Theoretic Cryptography

Stefan Neukamm
stefan.neukamm@mytum.de

Joint Advanced Student School '05

Outline

Information Theoretic Approach

plaintext m → **encryption** → ciphertext c

adversary:

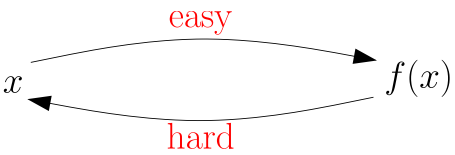
- Is there plaintext information left in the ciphertext?
- I have unlimited computational power!

Complexity Theoretic Approach

plaintext m → **encryption** → ciphertext c

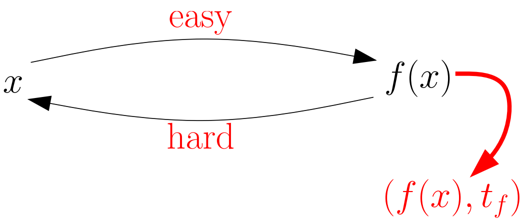
adversary:

- Can I efficiently extract plaintext information?
- I only have limited computational ressources!

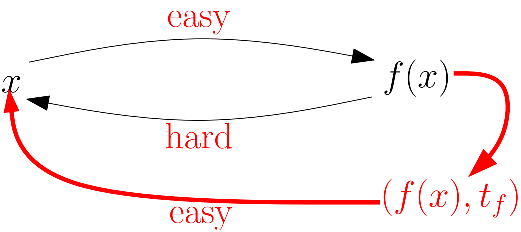


easy
x
f(x)
hard

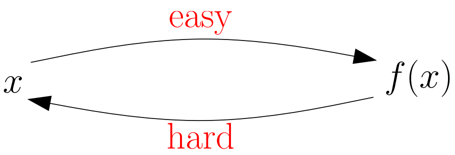
One-Way Function
Informal Definition.



One-Way Function
Informal Definition.



One-Way Function
Informal Definition.



Definition

A function *f* is called one-way, if *f* is easy to compute but hard to invert.

One-Way Function
Road Map to Formalize the Definition.

- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

One-Way Function
Road Map to Formalize the Definition.

- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

One-Way Function
Road Map to Formalize the Definition.

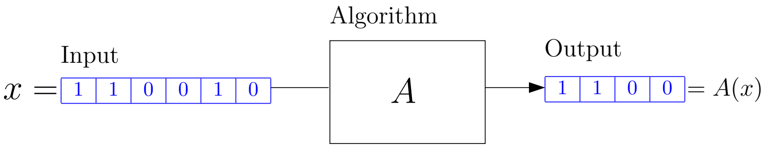
- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

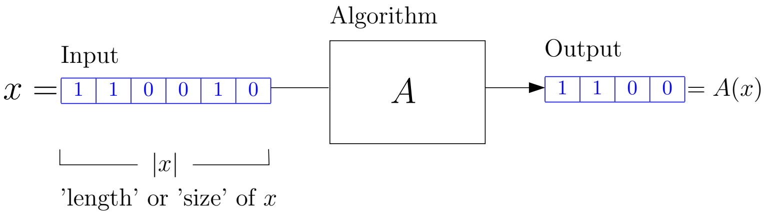
- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.

One-Way Function
Road Map to Formalize the Definition.

- Find proper definitions of easy and hard.
- Use computational complexity theory:
  - Classify problems according to their computational difficulty.
  - Classify problems according to needed resources (like time, storage space,...).
  - Our focus: `time complexity`.
  - Computational models: Turing machine, boolean circuits,...
- Basic definitions of complexity theory.



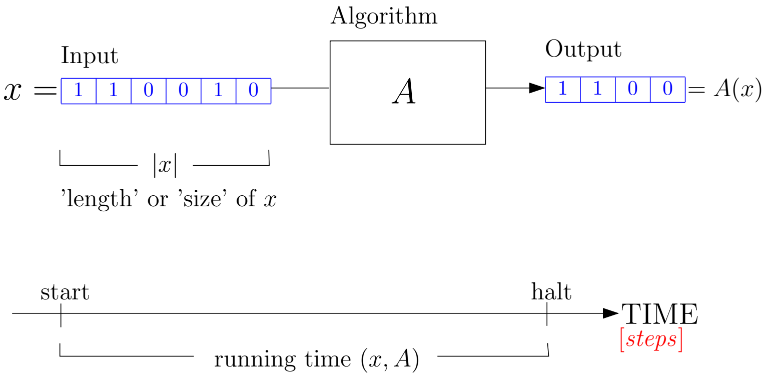
Algorithm
Input
$x =$ 1 1 0 0 1 0
A
Output
1 1 0 0 $= A(x)$



Algorithm
Input
$x = $ 1 1 0 0 1 0
A
Output
1 1 0 0 $= A(x)$
$|x|$
'length' or 'size' of x

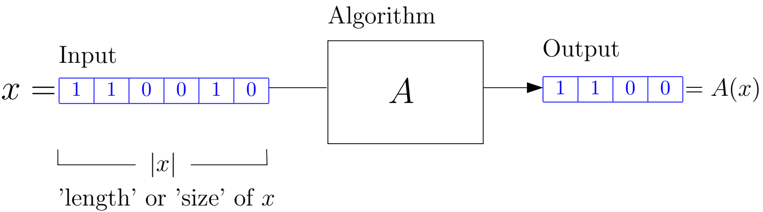
Complexity Theory - Basic Definitions

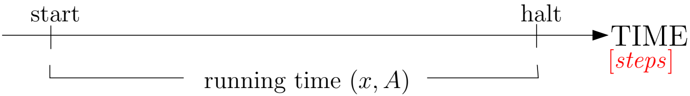
Algorithm; Running Time.



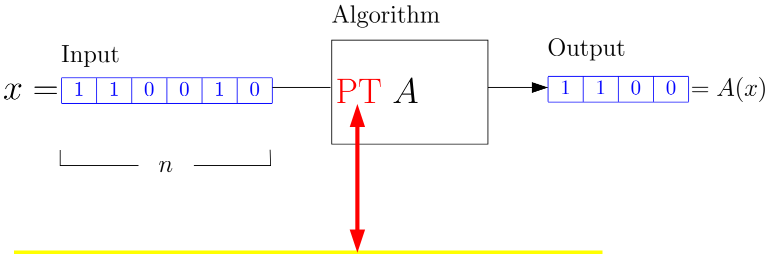
Complexity Theory - Basic Definitions

Algorithm; Running Time.

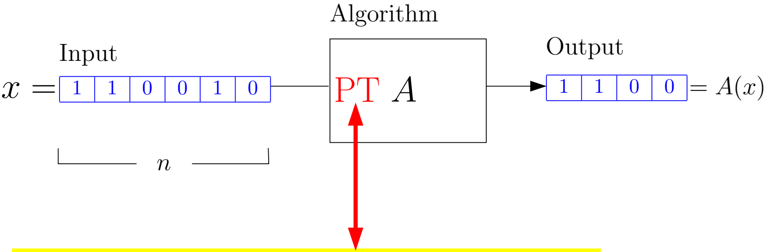




$$\text{worst case running time } (n) \geq \text{running time } (x, A) \qquad \forall x : |x| \leq n$$



$$\text{worst case running time } (n) \leq \text{poly}(n) \qquad \forall n$$



worst case running time $(n) \leq \text{poly}(n) \quad \forall n$

Otherwise: Exponential time algorithm

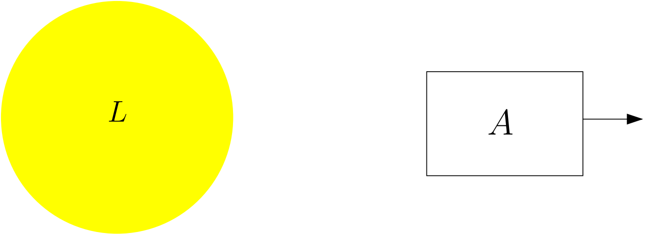
growing of poly., sub-exp., exp. functions

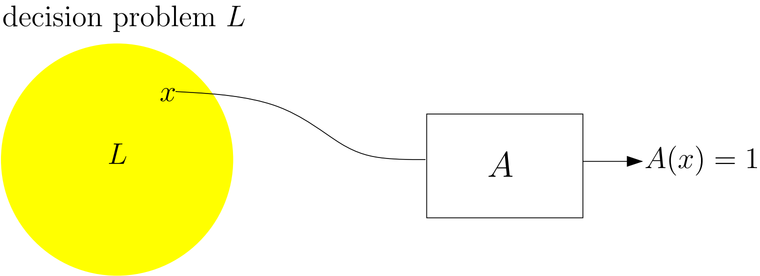
$f(x)$ / x	n^2	n^3	$\exp(\sqrt{n \ln n})$	2^n
10	10^2	10^3	$1.2 \cdot 10^2$	10^3
50	$2.5 \cdot 10^3$	$1.2 \cdot 10^5$	10^6	10^{15}
100	10^4	10^6	$2 \cdot 10^9$	10^{30}

Notes

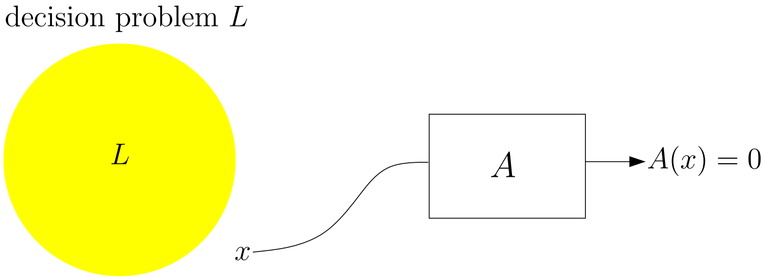
- polynomial time algorithm $\Leftrightarrow$ efficient
- exponential time algorithm $\Leftrightarrow$ inefficient

decision problem L

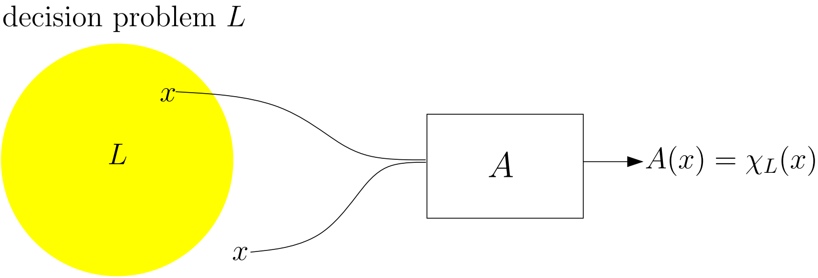


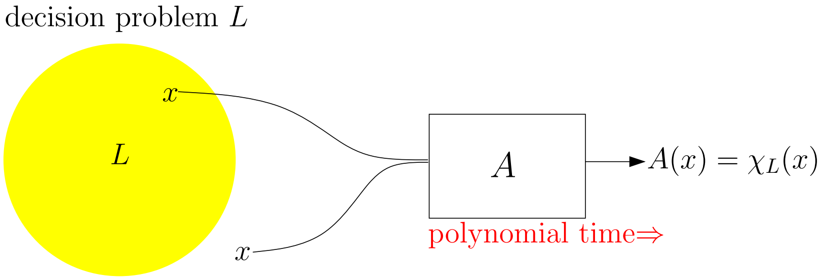


decision problem L
x
L
A
$A(x) = 1$

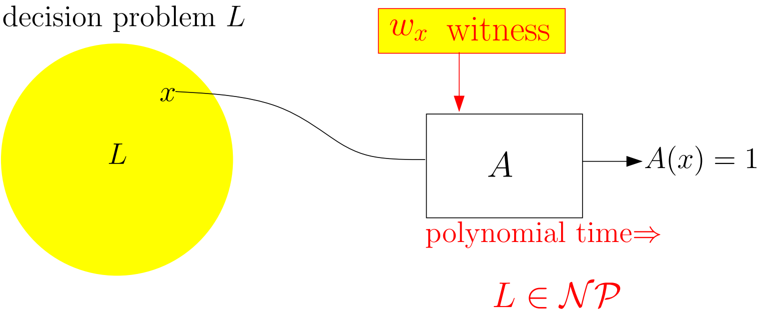


decision problem L
L
x
A
$A(x) = 0$


decision problem L
L
x
x
A
$A(x) = \chi_L(x)$



$L \in \mathcal{P}$


decision problem L
w_x witness
x
L
A
$A(x) = 1$
polynomial time⇒
$L \in \mathcal{NP}$

Fact

- $\mathcal{P} \subseteq \mathcal{NP}$

Examples

- PRIMES$\in\mathcal{P}$
- 3-Coloring-Problem: It is widely assumed that
  3COL $:= \{G : G$ is 3-colorable finite Graph$\} \notin \mathcal{P}$
  But $\forall G \in$ 3COL exists a **PT C** that makes G 3-colored $\Rightarrow$ 3COL $\in \mathcal{NP}$.

Definition (temporary)

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- f is easy to compute
- f is hard to invert.

Definition (temporary)

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- f is easy to compute
- f is hard to invert.

Definition (temporary)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PT A**: $\mathbf{A}(x) = f(x) \quad \forall x \in \{0,1\}^*$
- f is hard to invert.

Definition (temporary)

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PT A**: $\mathbf{A}(x) = f(x) \qquad \forall x \in \{0,1\}^*$
- f is hard to invert.

Definition (temporary)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PT A**: $\mathbf{A}(x) = f(x) \qquad \forall x \in \{0,1\}^*$
- $\nexists$**PT A′**: $\mathbf{A}'(f(x)) = x'$ with $f(x') = f(x) \qquad \forall x \in \{0,1\}^n$

Definition (temporary)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PT A**: $\mathbf{A}(x) = f(x) \qquad \forall x \in \{0,1\}^*$
- $\nexists$**PT A′**: $\mathbf{A}'(f(x)) = x'$ with $f(x') = f(x) \qquad \forall x \in \{0,1\}^n$

Example (Factoring)

Let $f_{mult}(p,q) := pq, \quad p,q$ primes.
Assumption: Factoring $\notin \mathcal{P} \Rightarrow f_{mult}$ is one-way (according to the above definition)

Observation of f_{mult}

- for $p, q \in \textsc{Primes} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

- for $p, q \in \text{PRIMES} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

- for $p, q \in \textsc{Primes} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

- for $p, q \in \text{PRIMES} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

- for $p, q \in$ PRIMES : $|p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

- for $p, q \in \textsc{Primes} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

Observation of f_{mult}

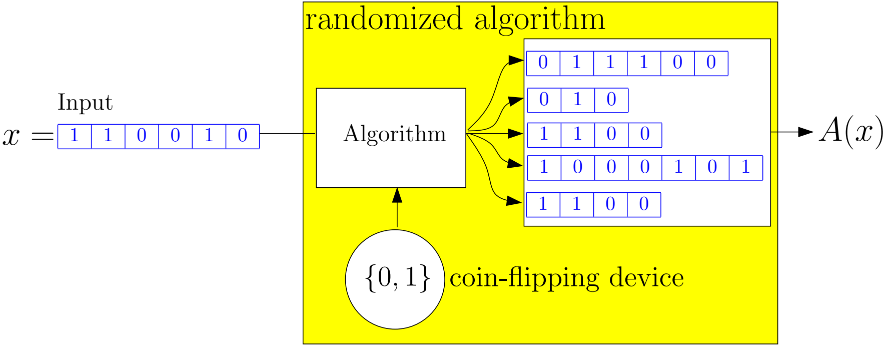
- for $p, q \in \textsc{Primes} : |p| \approx |q|$ huge, inverting $f_{mult}(p, q)$ is indeed hard
- But for half of the integers, finding an inverse of $n := f_{mult}(p, q)$ is very easy:

$$f_{mult}(n/2, 2) \in f_{mult}^{-1}(n)$$

$\Rightarrow$ Definition has to be improved.

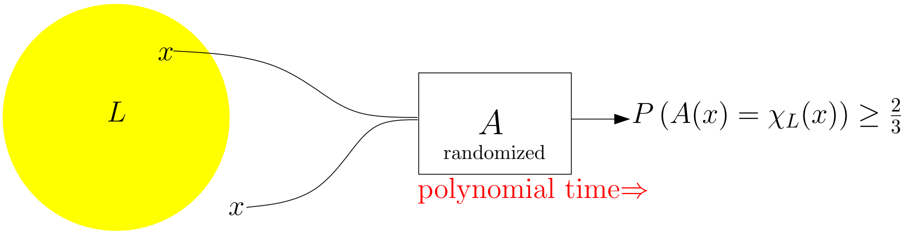
- Substitute: *worst-case complexity* $\Rightarrow$ average-case complexity
- success probability of an inverting algorithm should be negligible

$\Rightarrow$ randomized algorithms

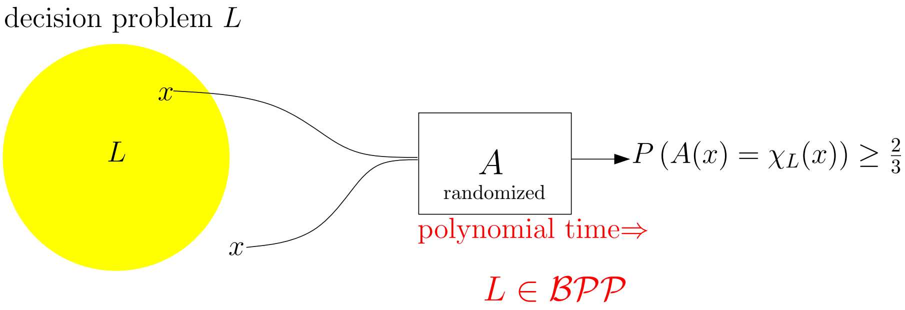


probabistic polynomial time, if worst case running time $(n) \leq \text{poly}(n)\ \forall n$

decision problem L



$$L \in \mathcal{BPP}$$



Notes

- $\mathcal{BPP}$ remains same with $\mathbf{P}(\mathbf{A}(x) = \chi_L(x)) \geq \frac{1}{2} + \frac{1}{p(|x|)}, p$ `polynomial` instead.
- $\mathcal{BPP} \Leftrightarrow$ 'efficiently' computable.

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- f is hard to invert.

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- f is hard to invert.

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** : $\forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A′** : **P** (**A′** successful) is negligible

One-Way Function
Definition.

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** : $\forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A′** : **P** (**A′** successful) $< \frac{1}{p(n)}$
  for all polynomials p and sufficiently large integers n

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** : $\forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A′** : $\mathbf{P}\,(\mathbf{A}'\text{successful}) < \frac{1}{p(n)} \quad \forall p$ poly., $\forall n \geq N_p$

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** : $\forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A′** : $\mathbf{P}\left(\mathbf{A}'(f(x)) \in f^{-1}(f(x))\right) < \frac{1}{p(n)} \quad \forall p \text{ poly.}, \forall n \geq N_p$

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A**$'$ $: \mathbf{P}\left(\mathbf{A}'(f(U_n)) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)} \quad \forall p$ poly.$, \forall n \geq N_p$

One-Way Function
Definition.

Definition

A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** : $\forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A**$'$: $\mathbf{P}\left(\mathbf{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)} \quad \forall p$ poly., $\forall n \geq N_p$

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A'** $: \mathbf{P}\left(\mathbf{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)} \quad \forall p$ poly.$, \forall n \geq N_p$

Notes

- Adversary is not unable to invert f, but has low probability to do so.

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A**$'$ $: \mathbf{P}\left(\mathbf{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)} \quad \forall p$ poly.$, \forall n \geq N_p$

Notes

- Adversary is not unable to invert f, but has low probability to do so.
- Definition works with asymptotic complexity: A sufficiently large *security parameter n* makes inversion infeasible.

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called one-way if the following two conditions hold

- $\exists$**PPT A** $: \forall x \in \{0,1\}^* : \mathbf{A}(x) = f(x)$
- $\forall$**PPT A**$'$ $: \mathbf{P}\left(\mathbf{A}'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)} \quad \forall p$ poly.$, \forall n \geq N_p$

Notes

- Adversary is not unable to invert f, but has low probability to do so.
- Definition works with asymptotic complexity: A sufficiently large *security parameter n* makes inversion infeasible.
- If f is $1-1$ then $f^{-1}(f(x)) = x$.

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called length preserving if

$$\forall x \in \{0,1\}^* : |f(x)| = |x|$$

A permutation is a length-preserving function f which is 1-1.

Lemma (Length-preserving)

If there exists a one-way function, then we can construct a length-preserving one-way function f:

$$\forall x \in \{0,1\}^* : |f(x)| = |x|$$

`Proof by reducibility arguments.`

FACTORING-problem

FACTORING	
Instance:	positive integer n
Question:	Find the prime factorization $n = \prod_i p_i^{e_i}$

Algorithms

- NUMBER FIELD SIEVE (1990)
  sub-exponential expected running time $\exp(1.9(\log n)^{1/3}(\log\log n)^{2/3}))$
- Special-purpose algorithms, like POLLARD´S $p - 1$

RSA function

$\mathrm{RSA}_{n,e}$	where $n = pq$, $\lvert p\rvert = \lvert q\rvert$ primes, $\gcd(e, \varphi(n)) = 1$
input:	x positive integer
output:	$\mathrm{RSA}_{n,e}(x) := x^e \mod n$

- $\mathrm{RSA}_{n,e}$ assumed to be one-way

Fact (Factoring vs. Inverting-RSA)

If n can be factored by a **PPT**$\Rightarrow$ $\mathrm{RSA}_{n,e}$ *can be inverted by a* **PPT**

INVERTING-RSA$\leq_P$FACTORING

Open Problem - Factoring vs. Inverting-RSA

Are FACTORING and INVERTING-RSA computationally equivalent?

Candidates Based on Factoring.

The SQUARE-Function by Rabin

Rabin´s SQUARE function

SQUARE_n	where $n = pq,\ p, q$ primes and $\lvert p\rvert = \lvert q\rvert$
input:	$x \in \mathbb{Z}_n^*$
output:	$\text{SQUARE}_n(x) := x^2 \mod n$

- SQUARE_n is not 1-1
- But SQUARE_n restricted to Q_n is a permutation, if
  $n \in \{pq : p, q \text{ distinct odd primes}, \lvert p\rvert = \lvert q\rvert,\ p \equiv q \equiv 3 \mod 4\}$
  $Q_n := \{x : x \in \mathbb{Z}_p^*,\ \exists y \in \mathbb{Z} : y^2 \equiv x \mod n\}$ quadratic-residues

Fact (FACTORING VS. INVERTING-SQUARE)

FACTORING*(n) and* INVERTING-SQUARE$_n$ *are computationally equivalent!*

DLP - discrete logarithm problem

DLP	
Instance:	a finite cyclic Group G of order n
	a generator α of G
	an element $\beta \in G$
Question:	Find the integer $x, 0 \leq x \leq n-1$: $\alpha^x = \beta$

- Given the prime factorization $n = \prod_i p_i^{e_i}$ the DLP in G can be reduced to **DLP**´s in the groups $\mathbb{Z}_{p_i}^*$

Algorithms

- Best randomized algorithms in sub-exponential running time.

EXP function

$\mathrm{EXP}_{p,\alpha}$	where p prime and α generator of $\mathbb{Z}_p^*$
input:	$x \in \mathbb{Z}_p^*$
output:	$\mathrm{EXP}_{p,\alpha}(x) := \alpha^x \mod p$

- EXP is one-way, *assuming* DLP is hard

Assumptions for concrete candidates:

FACTORING efficiently computable $\Rightarrow$ RSA not one-way
FACTORING efficiently computable $\Leftrightarrow$ SQUARING not one-way
DLP efficiently computable $\Leftrightarrow$ EXP not one-way

Traditional assumption. *hard to break in worst case*

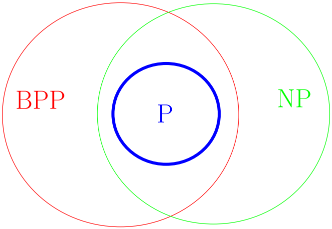
f computable by **PT** $\Rightarrow$ inverse under f computable by *non-det.* **PT**:
$\hookrightarrow \mathcal{P} = \mathcal{NP} \Rightarrow$ One-Way Function not exist.

Intractability assumption. *hard to break in average*

We assume the adversary uses a **PPT**
$\hookrightarrow \mathcal{NP} \subseteq \mathcal{BPP} \Rightarrow$ One-Way Function not exist. ($\mathcal{NP} \nsubseteq \mathcal{BPP} \Rightarrow \mathcal{P} \neq \mathcal{NP}$)

One-Way Function

Existence of One-Way Function cannot be proved yet.



Problem

- *Tradtional assumption and Intractability assumption are only necessary but not sufficient conditions.*
- *Existence of One-Way Functions not provable yet.*
- *Implementation based on reasonable 'intractability assumptions', like* FACTORING, DLP*.*

One-way function - up to now...

$$f : \{0,1\}^* \rightarrow \{0,1\}^*$$

One-way function - up to now...

$$f : \{0,1\}^* \rightarrow \{0,1\}^*$$

infinite domain

One-way function - up to now...

$$f : \underbrace{\{0,1\}^*}_{\text{infinite domain}} \rightarrow \{0,1\}^*$$

- Suitable for abstract discussion

One-way function - up to now...

$$f : \{0,1\}^* \rightarrow \{0,1\}^*$$

infinite domain

- Suitable for abstract discussion
- ..but not for natural candidates:

One-way function - up to now...

$$f : \{0,1\}^* \to \{0,1\}^*$$

infinite domain

- Suitable for abstract discussion
- ..but not for natural candidates:

$$EXP_{p,\alpha} : \{1, ..., p-2\} \to \{0,1\}^*$$

finite domain

Collection Of One-Way Functions

A larger View: Collection

$$f_i : D_i \rightarrow \{0,1\}^*$$

Collection Of One-Way Functions

A larger View: Collection

$$f_i : D_i \rightarrow \{0,1\}^*$$

finite domain

Collection Of One-Way Functions

A larger View: Collection

$$F := \{f_i : D_i \rightarrow \{0,1\}^*\}_{i \in I}$$

finite domain

Collection Of One-Way Functions

A larger View: Collection

$$F := \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$$

infinite set

Collection Of One-Way Functions

A larger View: Collection

$$F := \{f_i : D_i \rightarrow \{0,1\}^*\}_{i \in I}$$

infinite set

- The f_i sharing a common Index Sampler **S_I**

Collection Of One-Way Functions

A larger View: Collection

$$F := \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$$

infinite set

- The f_i sharing a common Index Sampler $\mathbf{S_I}$
- The f_i sharing a common Domain Sampler $\mathbf{S_D}$

Collection Of One-Way Functions

$F := \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$

Security parameter

$n \in \mathbb{N}$

Security parameter

$n \in \mathbb{N}$

PPT S_I Index sampler

$i \in I \cap \{0,1\}^n$

Collection Of One-Way Functions

$F := \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$

Security parameter

$n \in \mathbb{N}$

↓

PPT S_I Index sampler

↓

$i \in I \cap \{0,1\}^n$

↓

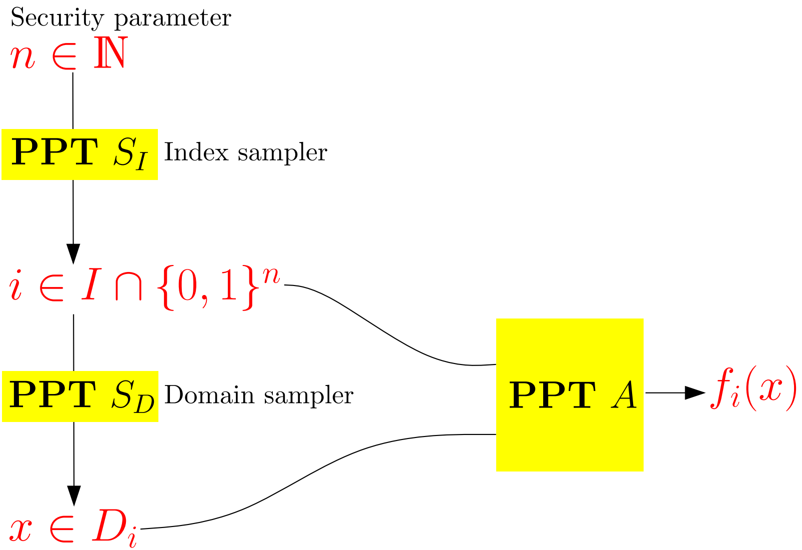
PPT S_D Domain sampler

↓

$x \in D_i$

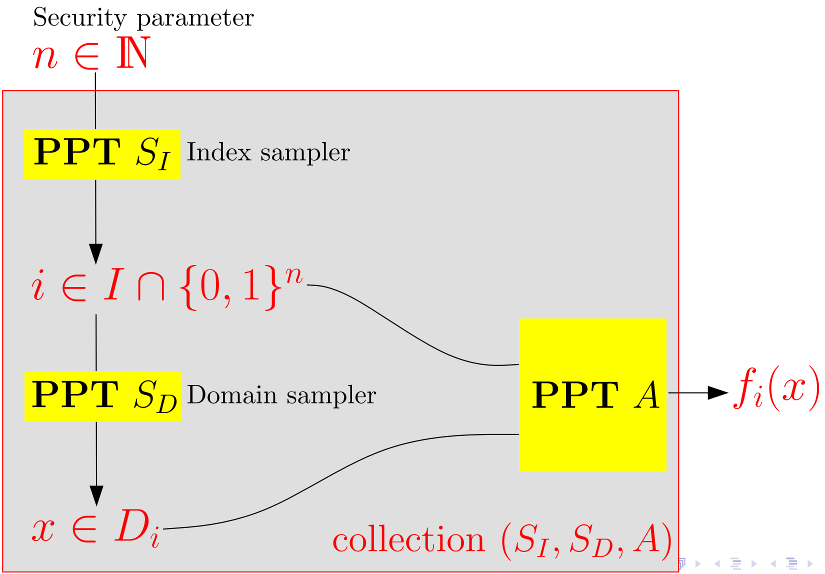
Collection Of One-Way Functions

$F := \{f_i : D_i \rightarrow \{0,1\}^*\}_{i \in I}$



Collection Of One-Way Functions

$F := \{f_i : D_i \to \{0,1\}^*\}_{i \in I}$



Definition

Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$.
A collection of one-way functions is a set

$$F = \{f_i : D_i \to \{0,1\}^*\}$$

satisfying the following two conditions

1. There exists tree **PPT $\mathbf{S_I}$, $\mathbf{S_D}$, A**, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - **A** on input $i \in I$ and $x \in D_i$ it holds that $A(i,x) = f_i(x)$

Definition

Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$.
A collection of one-way functions is a set

$$F = \{f_i : D_i \to \{0,1\}^*\}$$

satisfying the following two conditions

1. There exists tree **PPT** $\mathbf{S_I}$, $\mathbf{S_D}$, **A**, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - **A** on input $i \in I$ and $x \in D_i$ it holds that $A(i,x) = f_i(x)$
2. The probability of finding an inverse for every **PPT** given i and an element in range is negligible, if we consider the distribution induced by $\mathbf{S_I}$, $\mathbf{S_D}$.

Definition

Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$.
A collection of one-way functions is a set

$$F = \{f_i : D_i \to \{0,1\}^*\}$$

satisfying the following two conditions

1. There exists tree **PPT** $\mathbf{S_I}$, $\mathbf{S_D}$, **A**, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - **A** on input $i \in I$ and $x \in D_i$ it holds that $A(i,x) = f_i(x)$
2. The probability of finding an inverse for every **PPT** given i and an element in range is negligible, if we consider the distribution induced by $\mathbf{S_I}$, $\mathbf{S_D}$.
   For every **PPT** $\mathbf{A}'$, every polynomial $p(\cdot)$ and sufficiently large n:
   $\mathbf{P}\left(\mathbf{A}'(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1}(f_{I_n}(X_n))\right) < \frac{1}{p(n)}$

Definition

Let I be a set of indices and $D_i \subset \{0,1\}^*$ finite $\forall i \in I$.
A collection of one-way functions is a set

$$F = \{f_i : D_i \rightarrow \{0,1\}^*\}$$

satisfying the following two conditions

1. There exists tree **PPT** $\mathbf{S_I}$, $\mathbf{S_D}$, **A**, such that
   - $\mathbf{S_I}$ on input 1^n outputs an $i \in \{0,1\}^n \cap I$
   - $\mathbf{S_D}$ on input $i \in I$ outputs an $x \in D_i$
   - **A** on input $i \in I$ and $x \in D_i$ it holds that $A(i,x) = f_i(x)$
2. The probability of finding an inverse for every **PPT** given i and an element in range is negligible, if we consider the distribution induced by $\mathbf{S_I}$, $\mathbf{S_D}$.

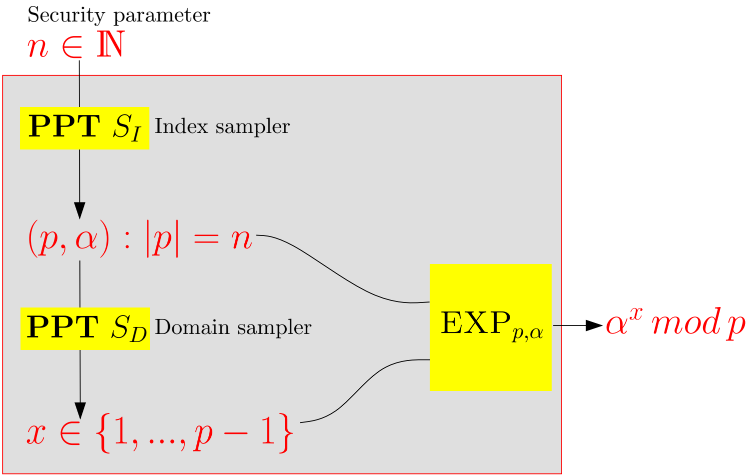
   For every **PPT** $\mathbf{A}'$, every polynomial $p(\cdot)$ and sufficiently large n:

   $$\mathbf{P}\left(\mathbf{A}'(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1}(f_{I_n}(X_n))\right) < \frac{1}{p(n)}$$

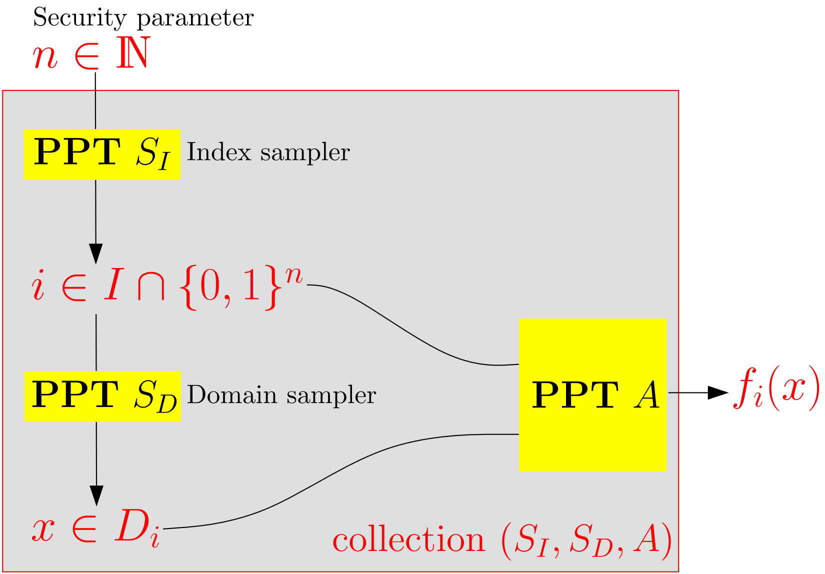
   I_n, X_n random variable describing output distribution of $\mathbf{S_I}$, $\mathbf{S_D}$

Collection Of One-Way Functions

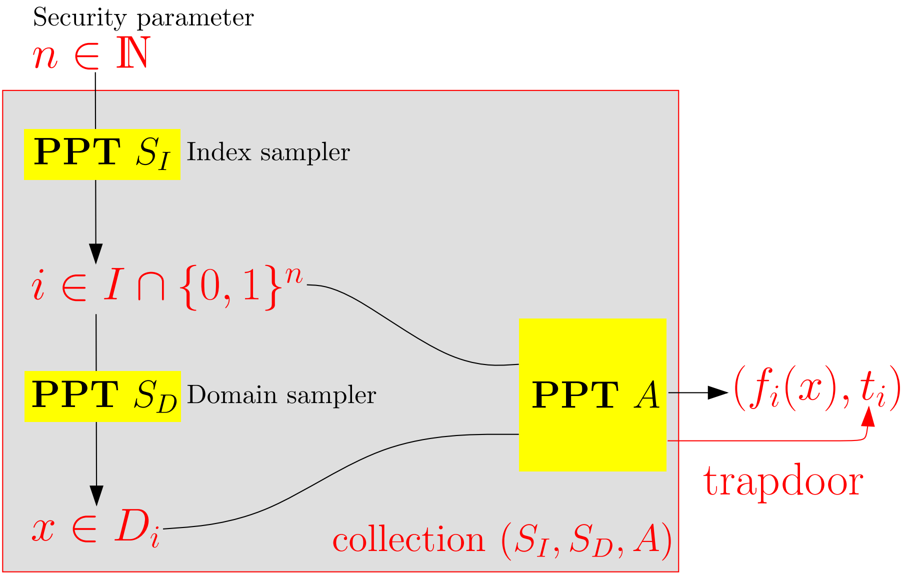
$EXP := \{EXP_{p,\alpha} : \mathbb{Z}_{p-1} \to \{0,1\}^*\}$



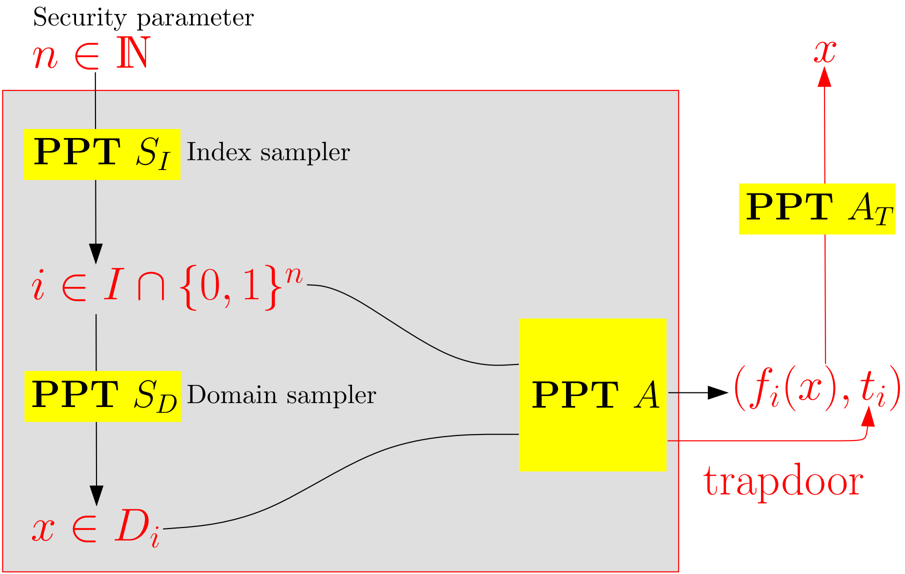
Collection Of Trapdoor Functions



Collection Of Trapdoor Functions



Collection Of Trapdoor Functions



Hard-Core Predicate - Motivation

Bit-Security of EXP

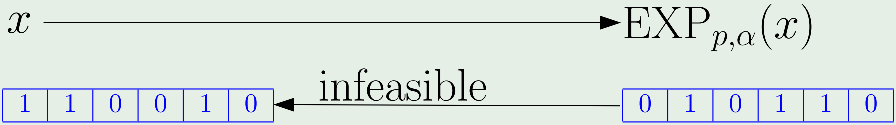
How secure is *EXP*?

$$x \longrightarrow \mathrm{EXP}_{p,\alpha}(x)$$

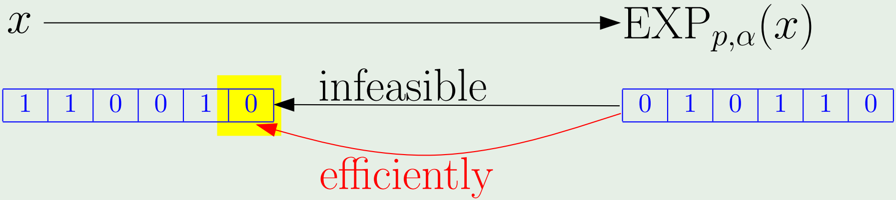
1	1	0	0	1	0

0	1	0	1	1	0

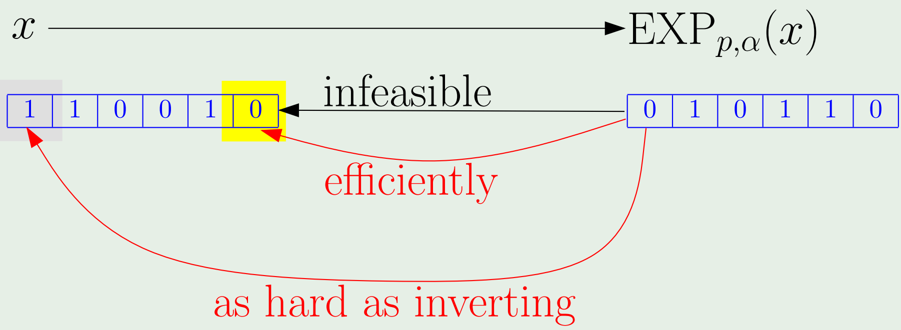
How secure is *EXP*?



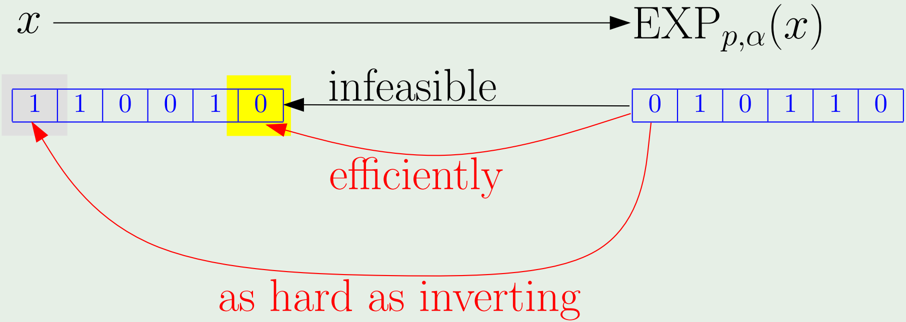
How secure is *EXP*?



How secure is *EXP*?

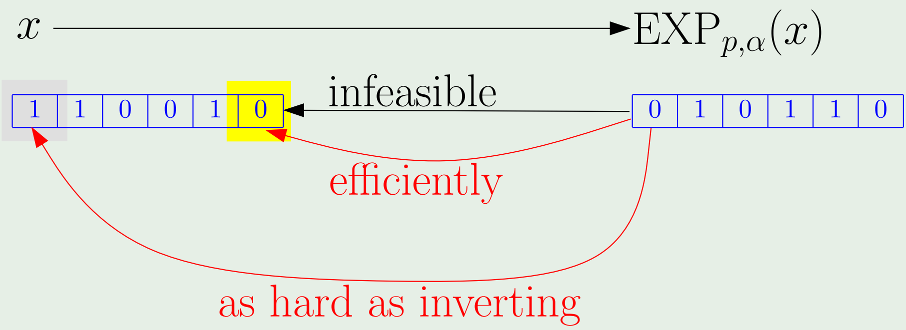


How secure is *EXP*?



- A one-way function doesn't hide partial information

How secure is *EXP*?



- A one-way function doesn't hide partial information
- But at least one Bit of information is hard to guess

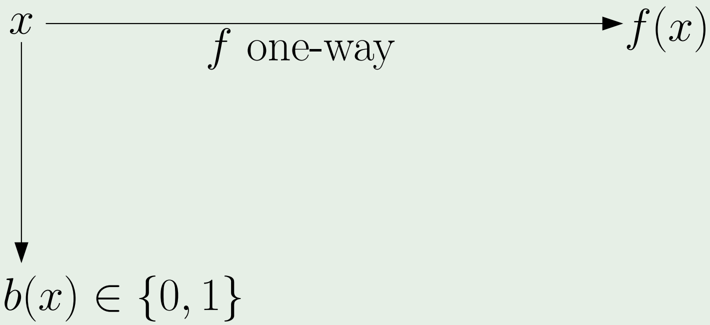
Hard-Core Predicate - Definition

Idea of hard-core predicate.

$$x \xrightarrow[f \text{ one-way}]{} f(x)$$

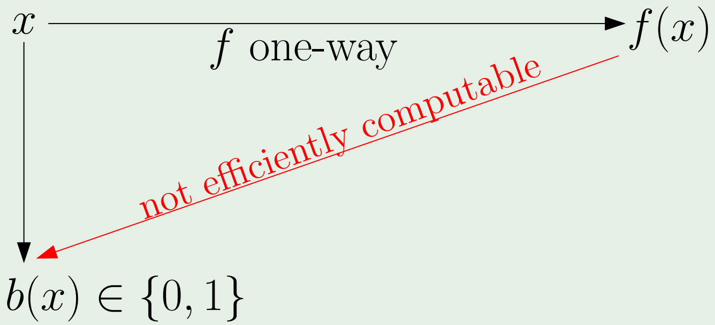
Hard-Core Predicate - Definition

Idea of hard-core predicate.



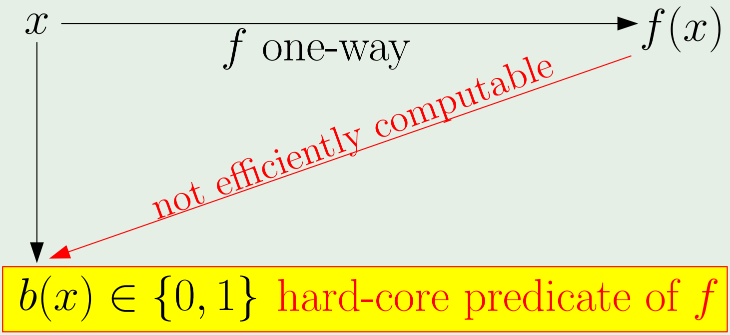
Hard-Core Predicate - Definition

Idea of hard-core predicate.



Hard-Core Predicate - Definition

Idea of hard-core predicate.



Hard-Core Predicate
Definition.

Instance

- a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$
- a predicate $b : \{0,1\}^* \rightarrow \{0,1\}$

Definition

b is a hard-core predicate of f, iff

- $\exists$**PPT A**, such that $\forall x : \mathbf{A}(x) = b(x)$

Hard-Core Predicate
Definition.

Instance

- a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$
- a predicate $b : \{0,1\}^* \rightarrow \{0,1\}$

Definition

b is a hard-core predicate of f, iff

- $\exists$**PPT A**, such that $\forall x : \mathbf{A}(x) = b(x)$
- Every efficient algorithm given $f(x)$ can guess $b(x)$ only with success probability negligible better than $\frac{1}{2}$

Instance

- a function $f : \{0,1\}^* \to \{0,1\}^*$
- a predicate $b : \{0,1\}^* \to \{0,1\}$

Definition

b is a hard-core predicate of f, iff

- $\exists$**PPT A**, such that $\forall x : \mathbf{A}(x) = b(x)$
- $\forall$**PPT G**, $\forall p$ polynomial and sufficiently large n:

$$\mathbf{P}\left(G(f(U_n)) = b(U_n)\right) < \frac{1}{2} + \frac{1}{p(n)}$$

Instance

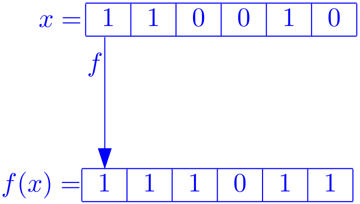
- $f : \{0,1\}^* \rightarrow \{0,1\}^*$length preserving
- $g(x,r) := (f(x),r)$, where $|x| = |r|$
- $b(x,r) :=< x,r >_{mod2} := \sum_i (x_i r_i \mod 2)$

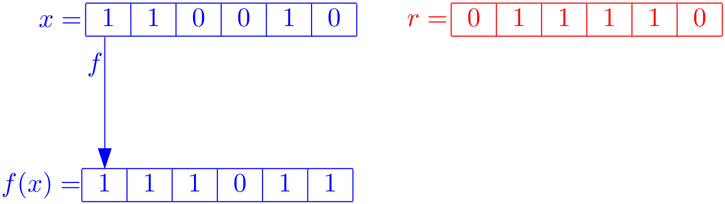
Theorem

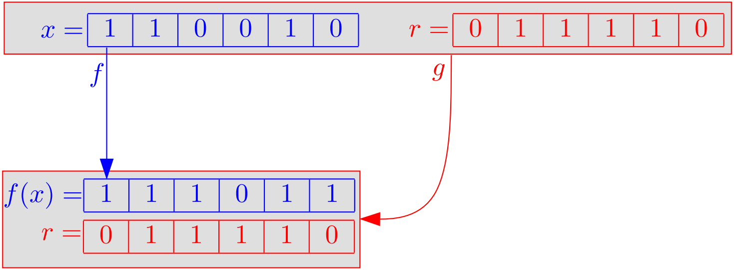
Let f be a length-preserving one-way function, and let g, b defined like above. Then b is a hard-core predicate of the function g.

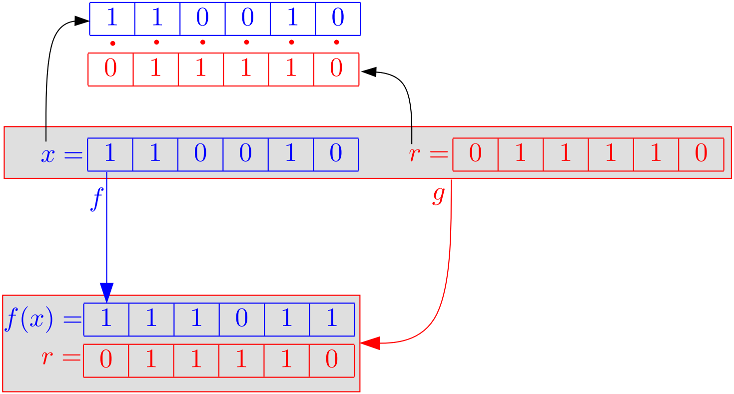
Notes

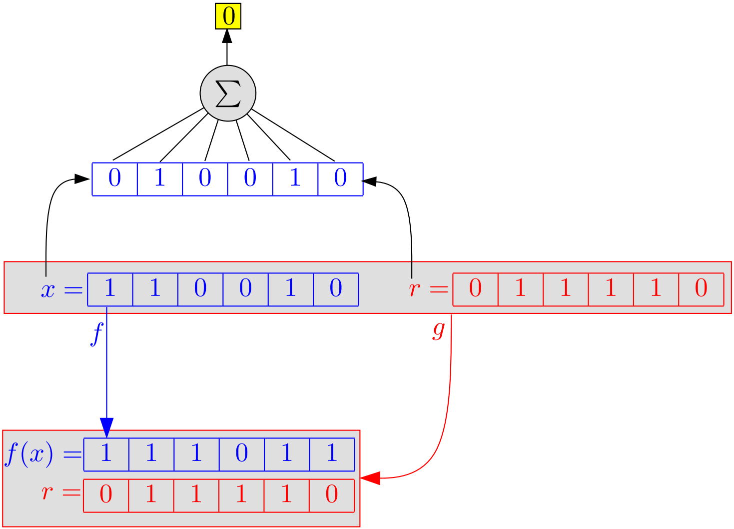
It means: it is infeasible to guess the exclusive-or of a random subset of the bits of x, when given $f(x)$ and the subset itself, denoted by r.

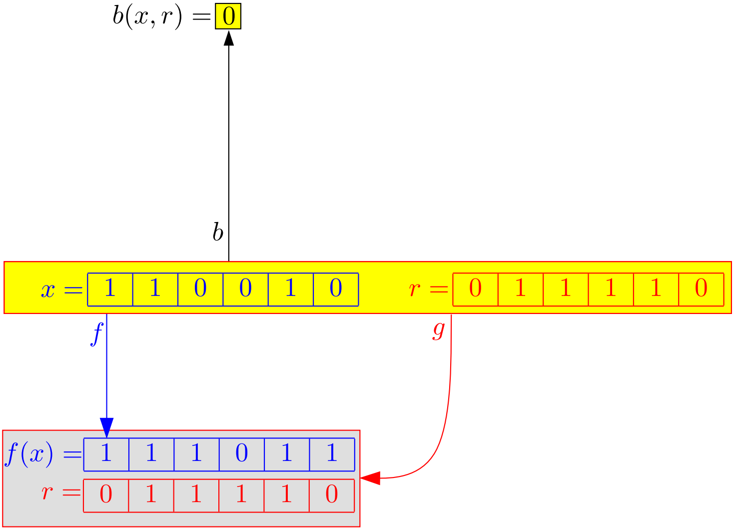

x =
1 1 0 0 1 0
f
f(x) =
1 1 1 0 1 1


x = 1 1 0 0 1 0
r = 0 1 1 1 1 0
f
f(x) = 1 1 1 0 1 1

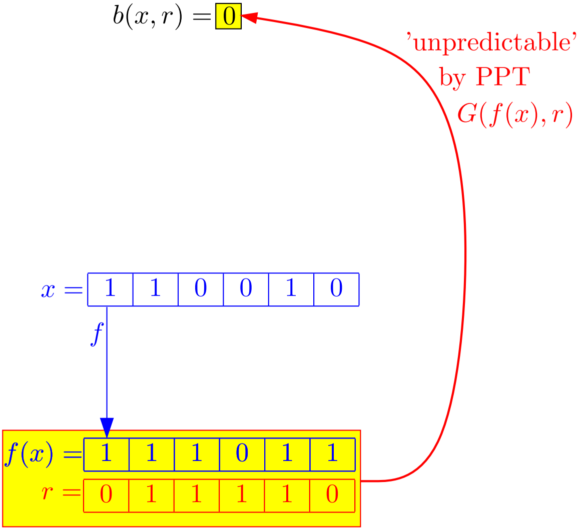

$x = $ 1 1 0 0 1 0
$r = $ 0 1 1 1 1 0
f
g
$f(x) = $ 1 1 1 0 1 1
$r = $ 0 1 1 1 1 0

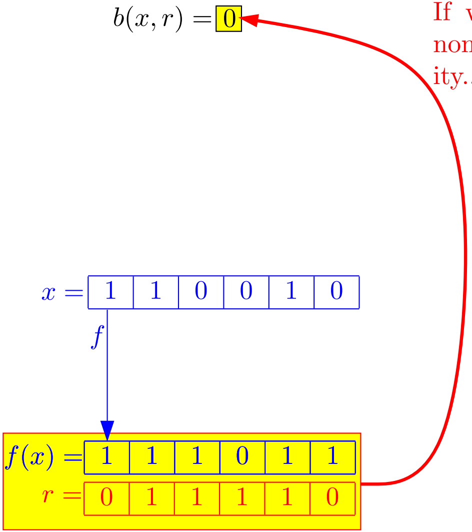

1 1 0 0 1 0
0 1 1 1 1 0
x = 1 1 0 0 1 0
r = 0 1 1 1 1 0
f
g
f(x) = 1 1 1 0 1 1
r = 0 1 1 1 1 0


0
Σ
0 1 0 0 1 0
x = 1 1 0 0 1 0
r = 0 1 1 1 1 0
f
g
f(x) = 1 1 1 0 1 1
r = 0 1 1 1 1 0



$b(x,r) = 0$
b
$x =$ 1 1 0 0 1 0
$r =$ 0 1 1 1 1 0
f
g
$f(x) =$ 1 1 1 0 1 1
$r =$ 0 1 1 1 1 0


$b(x, r) = 0$
'unpredictable'
by PPT
$G(f(x), r)$
$x =$ 1 1 0 0 1 0
f
$f(x) =$ 1 1 1 0 1 1
$r =$ 0 1 1 1 1 0



If we can predict b with non-negligible probability...

by PPT

$G(f(x), r)$

$b(x, r) = \boxed{0}$

If we can predict b with non-negligible probability...
by PPT
$G(f(x), r)$

...,then we can find an inverse of $f(x)$ with non-negligible probability!

$x =$ 1 1 0 0 1 0

f

$f(x) =$ 1 1 1 0 1 1
$r =$ 0 1 1 1 1 0

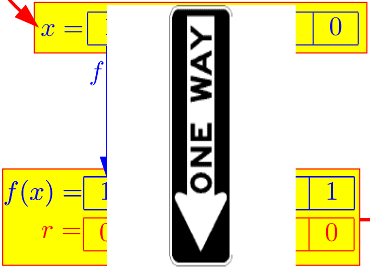
$b(x, r) = \boxed{0}$

If we can predict b with non-negligible probability...

by PPT

$G(f(x), r)$

...,then we can find an inverse of $f(x)$ with non-negligible probability!



A Generic Hard-Core Predicate - Proof
Sketch.

Proof sketch.

We use a 'reducibility argument' and proof by contradiction:

1. Suppose: b is not hard-core predicate of g
   Then there exists an *efficient* algorithm **G**, that can guess b with non-negligible probability better $\frac{1}{2}$:

   $\Rightarrow \exists$**PPT G**, $\exists p$ polynomial:
   $$\varepsilon(n) := \mathbf{P}\left(\mathbf{G}(f(X_n, R_n) = b(X_n, R_n)\right) - \frac{1}{2} > \frac{1}{p(n)}$$
2. Construct an *efficient* algorithm **A** (using **G**), which inverts f on input $(f(x), r)$ with non-negligible probability
3. Conclude:
   $\exists \mathbf{G} \Rightarrow \exists \mathbf{A} \Rightarrow f$ not one-way
   $\Rightarrow$contradiction to f one-way.

□

Proof - Inverting Algorithm **A**

Idea I - a mental experiment

Important Observation

$b(x,\alpha) \oplus b(x,\beta) = b(x,\alpha \oplus \beta)$
$x_i = b(x,\alpha) \oplus b(x,\alpha \oplus e_i)$

Mental Experiment

Suppose: Guessing by **G** works very good for a subset $S_n \subseteq \{0,1\}^n$:

- $\mathbf{P}(\mathbf{G} \text{ correct guess}) = \mathbf{P}(\mathbf{G}(f(x),r) = b(x,r)) > \frac{3}{4} + \frac{1}{2p(n)}$
- for all inputs $f(x)$ with $x \in S_n$
- for all sufficiently large $n \in \mathbb{N}$

Algorithm **A** (guessing the i^{th} bit of the inverse):

1. Randomly select $r \in \{0,1\}^n$
2. Compute $z_i := \mathbf{G}(f(x),r) \oplus \mathbf{G}(f(x),r \oplus e_i)$

Success probability: $\mathbf{P}\left(\mathbf{A}(f(x)) \in f^{-1}(f(x))\right) > \frac{1}{2} + \frac{3}{4p(n)}$

$\hookrightarrow$ Repetition and rule by majority $\Rightarrow$ efficiently computes x_i

Notice: $b(x,\alpha) \oplus b(x,\alpha \oplus e_i) = x_i \qquad \forall x,\alpha,i$

Idea to construct **A** inverting $f(x)$ for all $x \in S_n$

- Select a special subset S_n, where **G** works sufficiently successful.
- Use **G** to guess $b(x, r \oplus e_i)$
- Make own guess ρ for $b(x,r)$
- Both guess correct: $x_i = \rho \oplus \mathbf{G}(f(x), r \oplus e_i)$

Claim I (S_n, where **G** guesses sufficiently good)

If b not hard-core, n sufficiently large, then there exists a subset $S_n \subseteq \{0,1\}^n$, such that

- 'Large enough': $|S_n| \geq \frac{\varepsilon(n)}{2} 2^n$
- 'Succesful enough': $\forall x \in S_n : \pi(x) := \mathbf{P}(\mathbf{G}(x,R_n) = b(x,R_n)) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}$

Proof - Inverting Algorithm **A**

Idea II - ρ_J our own guess

Our guess

- Randomly select k strings $s_1, ..., s_k \in \{0,1\}^n$ and k predicates $\sigma_1, ..., \sigma_k \in \{0,1\}$ (by Laplace-Experiment)
- for every (non empty) index-subset $J \subseteq \{1, ..., k\}$:

$$r_J := \bigoplus_{j \in J} s_j$$

$$\Rightarrow b(x, r_J) = b(x, \bigoplus_{j \in J} s_j) = \bigoplus_{j \in J} b(x, s_j)$$

$$\Rightarrow \rho_J := \bigoplus_{j \in J} \sigma_j \text{ our guess of } b(x, r_J)$$

- Probability that $\rho_J = b(x, r_J)$ for all subsets $J \in \{1, ..., k\}$ is 2^{-k}

Algorithm (guesses i^{th} bit)

Let **A** be the following **PPT** algorithm:

1. Set $k := \lceil \log_2(2n \cdot p(n)^2 + 1 \rceil$
2. Uniformly and Independent select $s_1, ..., s_k \in \{0,1\}^n$, $\sigma_1, ..., \sigma_k \in \{0,1\}$
3. $\forall J \subseteq \{1, ..., k\}, J$ non-empty compute:
   - $r_J \leftarrow \bigoplus_{j \in J} s_j$
   - $\rho_J \leftarrow \bigoplus_{j \in J} \sigma_j$
   - $z_J \leftarrow \rho_J \oplus \mathbf{G}(f(x), r_J \oplus e_i)$
4. Output z the majority value of the z_J

Proof - Inverting Algorithm **A**
Observing Events.

$r_J \leftarrow \bigoplus_{j \in J} s_j$ — $s_j \in \{0,1\}^n$ randomly chosen

$\rho_J \leftarrow \bigoplus_{j \in J} \sigma_j$ — $\sigma_j \in \{0,1\}$ randomly chosen

$z_J \leftarrow \rho_J \oplus \mathbf{G}(f(x), r_J \oplus e_i)$ — compare: $x_i = b(x, r_J) \oplus b(x, r_J \oplus e_i)$

Events of interest

- Event $\mathcal{E}$: **G** guessing correct for majority of subsets $J \subseteq \{1, ..., k\}$:
  $\mathcal{E}$: $|\{J : \mathbf{G}(f(x), r_J \oplus e_i) = b(x, r_J \oplus e_i)\}| > \frac{1}{2}(2^k - 1)$
- Event $\mathcal{F}$: our guess correct for all subsets:
  $\mathcal{F}$: $\rho_J = b(x, r_J) \qquad \forall J \subseteq \{1, ..., k\}$

Probabilities

- Event $\mathcal{E}$:
  $\mathbf{P}(\mathcal{E} | x \in S_n) > \frac{1}{2}$ *(this we have to prove!)*
- Event $\mathcal{F}$:
  $\mathbf{P}(\mathcal{F} | x \in S_n) = \mathbf{P}(\forall J : \sigma_J = b(x, s_J) | x \in S_n) = 2^{-k}$ (Bernoulli)

Proof - Inverting Algorithm **A**
Success Probability

$z_J \leftarrow \rho_J \oplus \mathbf{G}(f(x), r_J \oplus e_i)$

$\mathbf{P}(\mathcal{E}|x \in S_n) > \frac{1}{2}$ — $\mathcal{E}$: **G** correct for the majority of all J's

$\mathbf{P}(\mathcal{F}|x \in S_n) = 2^{-k}$ — $\mathcal{F}$: ρ_J correct for all J's

$|S_n| > \frac{\epsilon}{2} \cdot 2^n \geq \frac{1}{2p(n)} 2^n$ — $k := \lceil \log_2(2n \cdot p(n)^2 + 1) \rceil$

Success Probability of Algorithm

$$
\begin{aligned}
&\mathbf{P}\left(\mathbf{A}(f(x)) \text{ outputs } i^{th} \text{ bit of an inverse of } f(x)\right)\\
=&\mathbf{P}(\text{For majority of all } J\text{'s: } z_J = x_i) = \mathbf{P}(\mathcal{E} \wedge \mathcal{F}|x \in S_n)\\
=&\mathbf{P}(\mathcal{E}) \cdot \mathbf{P}(\mathcal{F}) \cdot \mathbf{P}(x \in S_n) \ \textit{(Independence to be proved!)}\\
>&\frac{1}{2} \cdot 2^{-k} \cdot \frac{|S_n|}{2^n} = \frac{1}{8np(n)^3 + p(n)} = \frac{1}{poly(n)} \text{ not negligible!!!}
\end{aligned}
$$

$\hookrightarrow$ By repeating for all bits: we can efficiently compute x.

$\hookrightarrow$ Contradiction to 'f is one-way' $\Rightarrow$ b is hard-core Predicate

Claim I: There existst S_n, where **G** guesses sufficiently good

If b not hard-core, n sufficiently large, then there exists a subset $S_n \subseteq \{0,1\}^n$, such that

- 'Large enough': $|S_n| \geq \frac{\varepsilon(n)}{2} 2^n$
- 'Succesful enough': $\forall x \in S_n : \pi(x) := \mathbf{P}\left(\mathbf{G}(x, R_n) = b(x, R_n)\right) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}$

Claim II: $\mathbf{P}\left(\mathcal{E} | x \in S_n\right) > \frac{1}{2}$

For every $x \in S_n$:
$\mathbf{P}\left(|\{J : \mathbf{G}(f(x), r_J \oplus e_i) = b(x, r_J \oplus e_i)\}| > \frac{1}{2}(2^k - 1)\right) > 1 - \frac{1}{2p(n)}$

one-way functions are important primitives.

Formalizing and abstracting

The concept of one-way functions abstracts the central idea of many common cryptosystems:

- RSA
- RABIN-SQUARE
- ELGAMAL

As a basis

The introduced concept is a basis for more applicable theories:

- public key cryptosystems
- pseudorandom sequences
- hash functions
- ...

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like 'intractability assumption'
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like 'intractability assumption'
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like ’intractability assumption’
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like 'intractability assumption'
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like 'intractability assumption'
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Summary

- Basic definitions of computational complexity theory
- Formalized the definition of one-way function
- Discussed necessary conditions, like ’intractability assumption’
- Introduced the concept of one-way collections and trapdoor-collection
- Defined the hard-core predicate
- Proved the existence of a generic hard-core predicate

Literature

- O.Goldreich
  Foundations of cryptography
  2001, available online
- S. Goldwasser, M. Bellare
  Lecture notes on cryptography
  2001, available online
- A. Menezes, P. van Oorschot, S. Vanstone
  Handbook of Applied Cryptography
  CRC Press, 1969, `www.cacr.math.uwaterloo.co/hac`

Literature

- O.Goldreich
  Foundations of cryptography
  2001, available online
- S. Goldwasser, M. Bellare
  Lecture notes on cryptography
  2001, available online
- A. Menezes, P. van Oorschot, S. Vanstone
  Handbook of Applied Cryptography
  CRC Press, 1969, `www.cacr.math.uwaterloo.co/hac`

Literature

- O.Goldreich
  Foundations of cryptography
  2001, available online
- S. Goldwasser, M. Bellare
  Lecture notes on cryptography
  2001, available online
- A. Menezes, P. van Oorschot, S. Vanstone
  Handbook of Applied Cryptography
  CRC Press, 1969, `www.cacr.math.uwaterloo.co/hac`