Uniformly Polynomial Circuits and P

Theorem 65 *$L \in P$ if and only if L has uniformly polynomial circuits.*

- One direction was proved in Proposition 64 (p. 345).
- Now suppose that L has uniformly polynomial circuits.
- Decide $x \in L$ in polynomial time as follows:
  - Build $C_{|x|}$ in $\log |x|$ space, hence polynomial time.
  - Evaluate the circuit with input x in polynomial time.
- Therefore $L \in \mathrm{P}$.

Relation to P vs. NP

- Theorem 65 implies that P $\neq$ NP if and only if NP-complete problems have no uniformly polynomial circuits.
- A stronger conjecture: NP-complete problems have no polynomial circuits, *uniform or otherwise*.
- The above is currently the preferred approach to proving the P $\neq$ NP conjecture, of course without success so far.
  - Theorem 9 (p. 110) states that there are boolean functions requiring $2^n/(2n)$ gates to compute.
  - In fact, almost all boolean functions do.

BPP's Circuit Complexity

Theorem 66 *All languages in BPP have polynomial circuits.*

- Our proof will be *nonconstructive* in that only the existence of the desired circuits is shown.
  - Something exists if its probability of existence is nonzero.
- How to efficiently generate the circuit C_n given 1^n is not clear.
- If the construction of C_n is simple, then $\mathrm{P} = \mathrm{BPP}$, a most unlikely result.

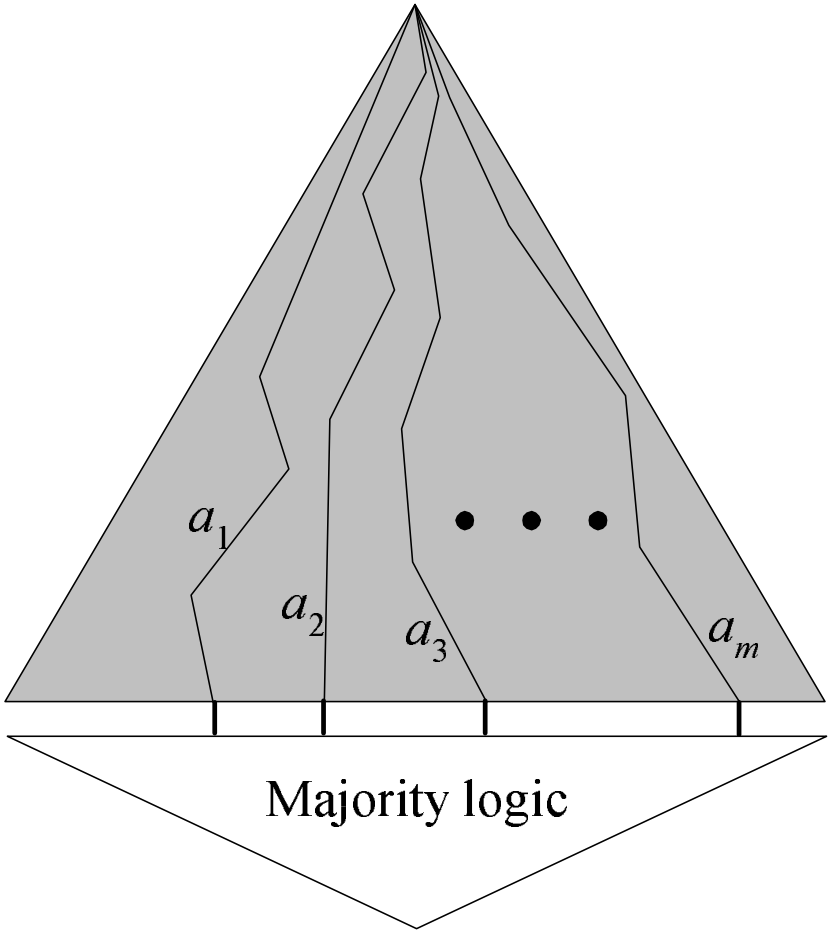
The Proof

- Let $L \in \text{BPP}$ be decided by a precise NTM N by clear majority.
- We shall prove that L has a polynomial family of circuits $C_0, C_1, \ldots$.
- Suppose that N runs in time $p(n)$, where p is a polynomial.
- Let $A_n = \{a_1, a_2, \ldots, a_m\}$, where $a_i \in \{0,1\}^{p(n)}$ and $m = 12(n+1)$.
- Each $a_i \in A_n$ represents a sequence of nondeterministic choices—i.e., a computation path—for N.

The Proof (continued)

- Let x be an input with $|x| = n$.
- Circuit C_n simply simulates N on x with each sequence of choices in A_n and then takes the majority of the m outcomes.
- Because N with a_i is a polynomial-time TM, it can be simulated by polynomial circuits of size $O(p(n)^2)$ (see the proof of Proposition 64 on p. 345).
- The size of C_n is therefore $O((mp(n))^2) = O(n^2 p(n)^2)$, a polynomial.
- We next prove the existence of A_n making C_n correct.

The Circuit



The Proof (continued)

- Call a_i **bad** if it leads N to a false positive or a false negative answer.
- Select A_n *uniformly randomly*.
- For each $x \in \{0,1\}^n$, at most $1/4$ of the computations of N are erroneous.
- Because the sequences in A_n are chosen randomly and independently, the expected number of bad a_i's is $m/4$.
- By the Chernoff bound (p. 330), the probability that the number of bad a_i's is $m/2$ or more is at most

$$e^{-m/12} < 2^{-(n+1)}.$$

The Proof (continued)

- The error probability is $< 2^{-(n+1)}$ for each $x \in \{0,1\}^n$.
- The probability that there is an x such that A_n results in an incorrect answer is $< 2^n 2^{-(n+1)} = 2^{-1}$.
  - $\text{prob}[\, E_1 \cup E_2 \cup \cdots \cup E_m \,] \leq \sum_{i=1}^{m} \text{prob}[\, E_i \,]$.
- So with probability one half, a random A_n produces a correct C_n for *all* inputs of length n.
- Because this probability exceeds 0, an A_n that makes majority vote work for all inputs of length n exists.
- Hence a correct C_n exist.

Cryptography

- **Alice** (A) wants to send a message to **Bob** (B) over a channel monitored by **Eve** (eavesdropper).
- The protocol should be such that the message is known only to Alice and Bob.
- The art and science of keeping messages secure is **cryptography**.

Encoding and Decoding

- Alice and Bob agree on two algorithms E and D—the **encoding** and the **decoding algorithms**.
- Both E and D are known to the public in the analysis.
- Alice runs E and wants to send a message x to Bob.
- Bob operates D.
- Privacy is assured in terms of two numbers e, d, the **encoding** and **decoding keys**.
- Alice sends $y = E(e, x)$ to Bob, who then performs $D(d, y) = x$ to recover x.
- x is called **plaintext**, and y is called **ciphertext**.

Some Requirements

- D should be an inverse of E given e and d.
- D and E must both run in (probabilistic) polynomial time.
- Eve should not be able to recover y from x without knowing d.
  - As D is public, d must be kept secret.
  - e may or may not be a secret.

Degrees of Security

- **Perfect secrecy**: After a ciphertext is intercepted by the enemy, the a posteriori probabilities of the clear texts that this ciphertext represents are identical to the a priori probabilities of the same clear texts before the interception.
- Such systems are said to be **informationally secure** to distinguish it from systems that are **computationally secure**.
- A system is computationally secure if to break it is theoretically possible, just computationally infeasible.

The One-Time Pad[a]

1: Alice generates a random string r as long as x;

2: Alice sends r to Bob over a secret channel;

3: Alice sends $r \oplus x$ to Bob over a public channel;

4: Bob receives y;

5: Bob recovers $x := y \oplus r$;

[a] Mauborgne, Vernam, 1917, Shannon, 1949.

Analysis

- The one-time pad uses $e = d = r$.
- This is said to be a **private-key cryptosystem**.
- Knowing x and knowing r are equivalent.
- The one-time pad therefore has perfect secrecy.
- The random bit string must be new for each round of communication.
  - **Cryptographically strong pseudorandom generators** require exchanging only the seed once.
- The assumption of a secure private channel is problematic.

Public-Key Cryptography[a]

- Suppose that only d is private to Bob, whereas e is public knowledge.
- Bob generates the (e, d) pair and publishes e.
- Anybody like Alice can send $E(e, x)$ to Bob.
- Knowing d, Bob can recover x by $D(d, E(e, x)) = x$.
- The assumptions are complexity-theoretic.
  - It is computationally difficult to compute d from e.
  - It is computationally difficult to compute x from y without knowing d.

[a] Diffie, Hellman, 1976.

Complexity Issues

- Given y and x, it is easy to verify whether $E(e, x) = y$.
- A public-key cryptosystem in some sense is within NP.
- A necessary condition for the existence of secure public-key cryptosystems is P $\neq$ NP.
- But more is needed than P $\neq$ NP.
- For example, it is not sufficient that D is hard to compute in the worst case.
- We want it to be hard to compute in "most" cases.

One-Way Functions

- We say that f is a **one-way function** if the following hold.
  - f is one-to-one.
  - For all $x \in \Sigma^*$, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some $k > 0$.
  - f can be computed in polynomial time.
  - f^{-1} cannot be computed in polynomial time.
    - Exhaustive search works, but it is too slow.
- Even if P $\neq$ NP, there is no guarantee that one-way functions exist.
- No functions have been proved to be one-way.

Candidates of One-Way Functions

- Modular exponentiation $f(x) = g^x \bmod p$.
  - **Discrete logarithm** is hard.
- The RSA[a] function $f(x) = x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - Breaking the RSA function is hard.
- Modular squaring $f(x) = x^2 \bmod pq$.
  - Determining whether a number with a Jacobi symbol 1 is a quadratic residue is hard—the **quadratic residuacity assumption** (**QRA**).

[a] Rivest, Shamir, Adleman, 1978.

The RSA Function

- Let p, q be two distinct primes.
- The RSA function is $x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
- By Lemma 45 (p. 258),

$$\phi(pq) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1.$$

- As $\gcd(e, \phi(pq)) = 1$, there is a d such that

$$ed = 1 \bmod \phi(pq),$$

  which can be found by the Euclidean algorithm.

A Public-Key Cryptosystem Based on RSA

- Bob generates p and q.
- Bob publishes pq and the encoding key e, a number relatively prime to $\phi(pq)$.
  - The encryption function is $y = x^e \bmod pq$.
- Knowing $\phi(pq)$, Bob calculates d such that $ed = 1 + k\phi(pq)$ for some $k \in \mathbb{Z}$.
  - The decryption function is $y^d \bmod pq$.
  - It works because $y^d = x^{ed} = x^{1+k\phi(pq)} = x \bmod pq$ by the Fermat-Euler theorem (p. 263).
- Factoring pq or calculating d from (e, pq) seems hard.

Probabilistic Encryption[a]

- The ability to, say, forge signatures on even a vanishingly small fraction of strings of some length could be a security weakness if those strings were the probable ones!
- What is required is a scheme that does not "leak" *partial* information.
- The first solution to the problems of skewed distribution and partial information was based on the QRA.

[a] Goldwasser, Micali, 1982.

The Setup

- Bob publishes $n = pq$, a product of two primes, and a quadratic nonresidue y with Jacobi symbol 1.
- Bob keeps secret the factorization of n.
- To send bit string $b_1 b_2 \ldots b_k$ to Bob, Alice encrypts the bits by choosing a random quadratic residue modulo n if b_i is 1 and a random quadratic nonresidue with Jacobi symbol 1 otherwise.
- A sequence of residues and nonresidues are sent.
- Knowing the factorization of n, Bob can efficiently test quadratic residuacity and thus read the message.

The Protocol for Alice

1: **for** $i = 1, 2, \ldots, k$ **do**
2: Pick $r \in Z_n^*$ randomly;
3: **if** $b_i = 1$ **then**
4: Send $r^2 \bmod n$;
5: **else**
6: Send $r^2 y \bmod n$;
7: **end if**
8: **end for**

The Protocol for Bob

1: **for** $i = 1, 2, \ldots, k$ **do**
2: Receive r;
3: **if** $(r \mid p) = 1$ and $(r \mid q) = 1$ **then**
4: $b_i := 1$;
5: **else**
6: $b_i := 0$;
7: **end if**
8: **end for**

Semantic Security

- This encryption scheme is probabilistic.
- There are a large number of different encryptions of a given message.
- One is chosen at random by the sender to represent the message.
- This scheme is both polynomially secure and **semantically secure**.

A Probabilistic Encryption Based on RSA

The Protocol for Alice.

1: **for** $i = 1, 2, \ldots, k$ **do**
2: Pick $r \in \{1, \ldots, pq/2\}$ randomly;
3: Send $(2r + b_i)^e \bmod pq$;
4: **end for**

The Protocol for Bob.

1: **for** $i = 1, 2, \ldots, k$ **do**
2: Receive y;
3: $b_i := (y^d \bmod pq) \bmod 2$;
4: **end for**

Digital Signatures[a]

- Alice wants to send Bob a *signed* document x.
- The signature must unmistakably identifies the sender.
- Both Alice and Bob have public and private keys

$$e_{\text{Alice}}, e_{\text{Bob}}, d_{\text{Alice}}, d_{\text{Bob}}.$$

- Assume the cryptosystem satisfies the commutative property

$$E(e, D(d, x)) = D(d, E(e, x)). \tag{4}$$

  - As $(x^d)^e = (x^e)^d$, the RSA system satisfies it.
  - Any cryptosystem guarantees $D(d, E(e, x)) = x$.

[a] Diffie, Hellman, 1976.

Digital Signatures Based on Public-Key Systems

- Alice signs x as

$$(x, D(d_{\text{Alice}}, x)).$$

- Bob receives (x, y) and verifies the signature by checking

$$E(e_{\text{Alice}}, y) = E(e_{\text{Alice}}, D(d_{\text{Alice}}, x)) = x$$

  based on Eq. (4).

- The claim of authenticity is founded on the difficulty of inverting E_{Alice} without knowing the key d_{Alice}.
- Warning: If Alice signs anything presented to her, she might inadvertently decrypts a ciphertext of hers.

Mental Poker[a]

- Suppose that Alice and Bob have agreed on 3 n-bit numbers $a < b < c$, the cards.
- They want to randomly choose one card each, so that:
  - Their cards are different.
  - All 6 pairs of distinct cards are equiprobable.
  - Alice's (Bob's) card is known to Alice (Bob) but not to Bob (Alice), until Alice (Bob) announces it.
  - The person with the highest card wins the game.
  - The outcome is indisputable.
- Alice and Bob will not deviate from the protocol.

[a]Shamir, Rivest, Adleman, 1981.

The Setup

- Alice and Bob agree on a large prime p;
- Each has two *secret* keys $e_{\text{Alice}}, e_{\text{Bob}}, d_{\text{Alice}}, d_{\text{Bob}}$ such that $e_{\text{Alice}} d_{\text{Alice}} = e_{\text{Bob}} d_{\text{Bob}} = 1 \bmod p - 1$;
  - This ensures that $(x^{e_{\text{Alice}}})^{d_{\text{Alice}}} = x$ and $(x^{e_{\text{Bob}}})^{d_{\text{Bob}}} = x$.
- The protocol lets Bob pick Alice's card and Alice pick Bob's card.
- Cryptographic techniques make it plausible that Alice's and Bob's choices are practically random, for lack of time to break the system.

The Protocol

1: Alice encrypts the cards

$$a^{e_{\text{Alice}}} \bmod p, b^{e_{\text{Alice}}} \bmod p, c^{e_{\text{Alice}}} \bmod p$$

and sends them in random order to Bob;

1: Bob picks one of the messages $x^{e_{\text{Alice}}}$ to send to Alice;

2: Alice decodes it $(x^{e_{\text{Alice}}})^{d_{\text{Alice}}} = x \bmod p$ for her card;

3: Bob encrypts the two remaining cards $(x^{e_{\text{Alice}}})^{e_{\text{Bob}}} \bmod p, (y^{e_{\text{Alice}}})^{e_{\text{Bob}}} \bmod p$ and sends them in random order to Alice;

4: Alice picks one of the messages, $(z^{e_{\text{Alice}}})^{e_{\text{Bob}}}$, encrypts it $((z^{e_{\text{Alice}}})^{e_{\text{Bob}}})^{d_{\text{Alice}}} \bmod p$, and sends it to Bob;

5: Bob decrypts the message $(((z^{e_{\text{Alice}}})^{e_{\text{Bob}}})^{d_{\text{Alice}}})^{d_{\text{Bob}}} = z \bmod p$ for his card;

What Is a Proof?

- A proof convinces a party of a certain claim.
- In mathematics, a proof is a fixed sequence of theorems.
  - Think of a written examination.
- We will extend a proof to cover a proof *process* by which the validity of the assertion is established.
  - Think of a job interview or an oral examination.