BPP[a] (Bounded Probabilistic Polynomial)

- The class **BPP** contains all languages L for which there is a precise polynomial-time NTM N such that:
  - If $x \in L$, then at least $3/4$ of the computation paths of N on x lead to "yes."
  - If $x \notin L$, then at least $3/4$ of the computation paths of N on x lead to "no."
- So N accepts or rejects by a *clear* majority.

[a] Gill (1977).

Magic 3/4?

- The number 3/4 bounds the probability (ratio) of a right answer away from 1/2.
- Any constant *strictly* between 1/2 and 1 can be used without affecting the class BPP.
- In fact, as with RP,

$$\frac{1}{2} + \frac{1}{q(n)}$$

  for any polynomial $q(n)$ can replace 3/4 (p. 589).
- The next algorithm shows why.

The Majority Vote Algorithm

Suppose L is decided by N by majority $(1/2) + \epsilon$.

1: **for** $i = 1, 2, \ldots, 2k + 1$ **do**
2: Run N on input x;
3: **end for**
4: **if** "yes" is the majority answer **then**
5: "yes";
6: **else**
7: "no";
8: **end if**

Analysis

- The running time remains polynomial: $2k + 1$ times N's running time.
- By Corollary 75 (p. 600), the probability of a false answer is at most $e^{-\epsilon^2 k}$.
- By taking $k = \lceil 2/\epsilon^2 \rceil$, the error probability is at most $1/4$.
- Even if ϵ is any inverse polynomial, k remains a polynomial in n.

Aspects of BPP

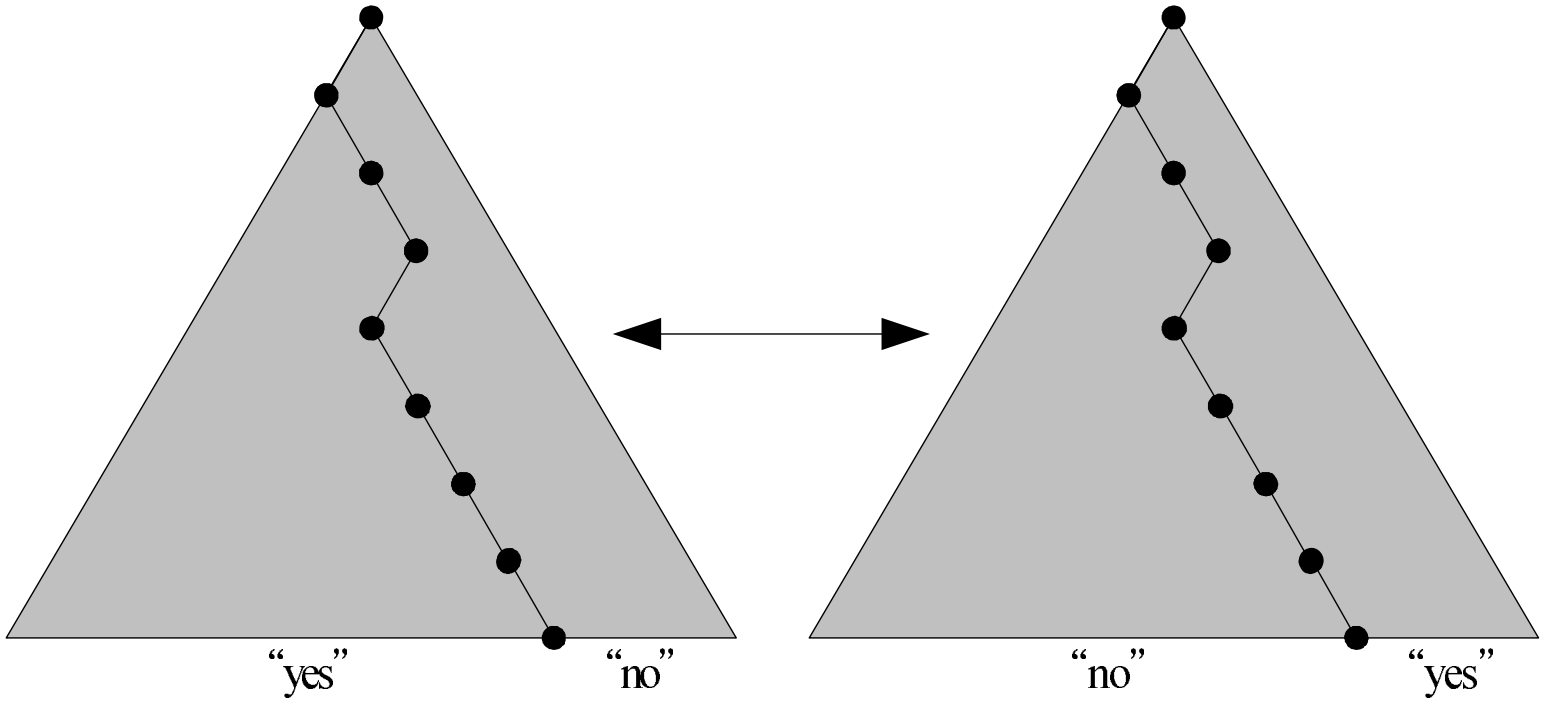
- BPP is the most comprehensive yet plausible notion of efficient computation.
  - If a problem is in BPP, we take it to mean that the problem can be solved efficiently.
  - In this aspect, BPP has effectively replaced P.
- $(\text{RP} \cup \text{coRP}) \subseteq (\text{NP} \cup \text{coNP})$.
- $(\text{RP} \cup \text{coRP}) \subseteq \text{BPP}$.
- Whether $\text{BPP} \subseteq (\text{NP} \cup \text{coNP})$ is unknown.
- But it is unlikely that $\text{NP} \subseteq \text{BPP}$ (see p. 622 and p. 623).

coBPP

- The definition of BPP is symmetric: acceptance by clear majority and rejection by clear majority.
- An algorithm for $L \in \text{BPP}$ becomes one for $\bar{L}$ by reversing the answer.
- So $\bar{L} \in \text{BPP}$ and $\text{BPP} \subseteq \text{coBPP}$.
- Similarly $\text{coBPP} \subseteq \text{BPP}$.
- Hence $\text{BPP} = \text{coBPP}$.
- This approach does not work for RP.[a]

[a] It did not work for NP either.

BPP and coBPP

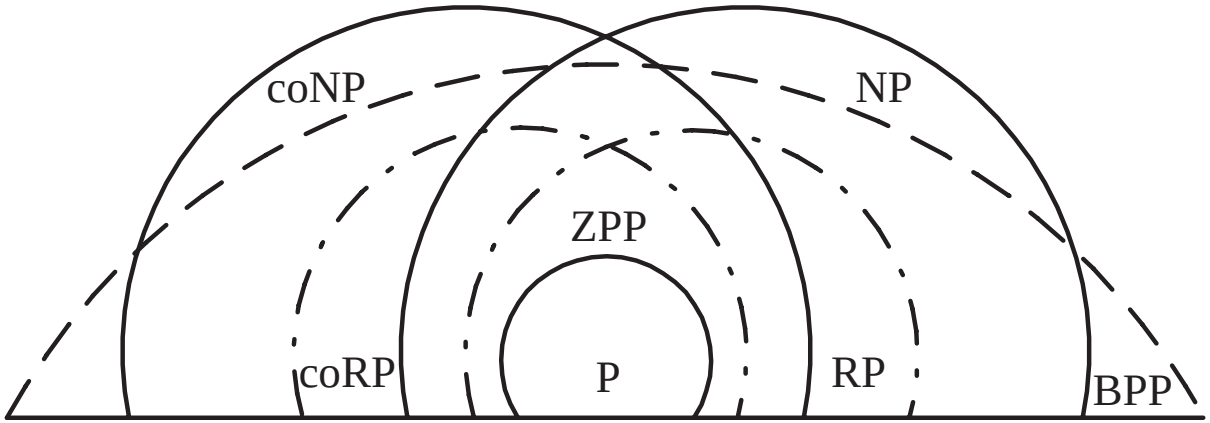


BPP and P: Derandomization

Theorem 76 (Nisan and Wigderson (1994)) *If every language in BPP only needs a pseudorandom generator which stretches a random seed of logarithmic length, then BPP = P.*

- We only need to show BPP $\subseteq$ P.
- Run the BPP algorithm for each of the seeds.
  - There are only $2^{O(\log n)} = O(n^c)$ seeds, a polynomial
- Accept if and only if at least 3/4 of the outcomes is a "yes."
- The running time is deterministically polynomial.

"The Good, the Bad, and the Ugly"



Circuit Complexity

- Circuit complexity is based on boolean circuits instead of Turing machines.
- A boolean circuit with n inputs computes a boolean function of n variables.
- Now, identify `true`/1 with "yes" and `false`/0 with "no."
- Then a boolean circuit with n inputs accepts certain strings in $\{ 0, 1 \}^n$.
- To relate circuits with an arbitrary language, we need one circuit for each possible input length n.

Formal Definitions

- The **size** of a circuit is the number of *gates* in it.
- A **family of circuits** is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n boolean inputs.
- For input $x \in \{\,0,1\,\}^*$, $C_{|\,x\,|}$ outputs 1 if and only if $x \in L$.
- In other words,

$$C_n \text{ accepts } L \cap \{\,0,1\,\}^n.$$

Formal Definitions (concluded)

- $L \subseteq \{ 0, 1 \}^*$ has **polynomial circuits** if there is a family of circuits $\mathcal{C}$ such that:
  - The size of C_n is at most $p(n)$ for some fixed polynomial p.
  - C_n accepts $L \cap \{ 0, 1 \}^n$.

Exponential Circuits Suffice for All Languages

- Theorem 18 (p. 214) implies that there are languages that cannot be solved by circuits of size $2^n/(2n)$.
- But surprisingly, circuits of size 2^{n+2} can solve *all* problems, decidable or otherwise!

Exponential Circuits Suffice for All Languages (continued)

Proposition 77 *All decision problems (decidable or otherwise) can be solved by a circuit of size* 2^{n+2}.

- We will show that for any language $L \subseteq \{0,1\}^*$, $L \cap \{0,1\}^n$ can be decided by a circuit of size 2^{n+2}.
- Define boolean function $f : \{0,1\}^n \to \{0,1\}$, where

$$f(x_1 x_2 \cdots x_n) = \begin{cases} 1 & x_1 x_2 \cdots x_n \in L, \\ 0 & x_1 x_2 \cdots x_n \notin L. \end{cases}$$

The Proof (concluded)

- Clearly, any circuit that implements f decides $L \cap \{\, 0, 1 \,\}^n$.
- Now,

$$f(x_1 x_2 \cdots x_n) = (x_1 \wedge f(1 x_2 \cdots x_n)) \vee (\neg x_1 \wedge f(0 x_2 \cdots x_n)).$$

- The circuit size $s(n)$ for $f(x_1 x_2 \cdots x_n)$ hence satisfies

$$s(n) = 4 + 2s(n-1)$$

  with $s(1) = 1$.
- Solve it to obtain $s(n) = 5 \times 2^{n-1} - 4 \leq 2^{n+2}$.

The Circuit Complexity of P

Proposition 78 *All languages in P have polynomial circuits.*

- Let $L \in \mathrm{P}$ be decided by a TM in time $p(n)$.
- By Corollary 35 (p. 315), there is a circuit with $O(p(n)^2)$ gates that accepts $L \cap \{\, 0, 1 \,\}^n$.
- The size of the circuit depends only on L and the length of the input.
- The size of the circuit is polynomial in n.

Polynomial Circuits vs. P

- Is the converse of Proposition 78 true?
  - Do polynomial circuits accept only languages in P?
- No.
- Polynomial circuits can accept *undecidable* languages!

Languages That Polynomial Circuits Accept

- Let $L \subseteq \{ 0, 1 \}^*$ be an undecidable language.
- Let $U = \{ 1^n : \text{the binary expansion of } n \text{ is in } L \}$.[a]
  - For example, $11111_1 \in U$ if $101_2 \in L$.
- U is also undecidable (prove it).
- $U \cap \{ 1 \}^n$ is accepted by the trivial circuit C_n that outputs 1 if $1^n \in U$ and outputs 0 if $1^n \notin U$.[b]
- The family of circuits $(C_0, C_1, \ldots)$ is polynomial in size.

[a] Assume n's leading bit is always 1 without loss of generality.

[b] We may not know which is the case for *general* n.

A Patch

- Despite the simplicity of a circuit, the previous discussions imply the following:
  - Circuits are *not* a realistic model of computation.
  - Polynomial circuits are *not* a plausible notion of efficient computation.
- What is missing?
- The *effective and efficient constructibility* of

$$C_0, C_1, \ldots.$$

Uniformity

- A family $(C_0, C_1, \ldots)$ of circuits is **uniform** if there is a $\log n$-space bounded TM which on input 1^n outputs C_n.
  - Note that n is the length of the input to C_n.
  - Circuits now cannot accept undecidable languages (why?).
  - The circuit family on p. 618 is not constructible by a *single* Turing machine (algorithm).
- A language has **uniformly polynomial circuits** if there is a *uniform* family of polynomial circuits that decide it.

Uniformly Polynomial Circuits and P

Theorem 79 *$L \in P$ if and only if L has uniformly polynomial circuits.*

- One direction was proved in Proposition 78 (p. 616).
- Now suppose L has uniformly polynomial circuits.
- A TM decides $x \in L$ in polynomial time as follows:
  - Calculate $n = |\, x \,|$.
  - Generate C_n in $\log n$ space, hence polynomial time.
  - Evaluate the circuit with input x in polynomial time.
- Therefore $L \in \mathrm{P}$.

Relation to P vs. NP

- Theorem 79 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, *uniformly or not*.
- The above is currently the preferred approach to proving P $\neq$ NP—without success so far.

BPP's Circuit Complexity

Theorem 80 (Adleman (1978)) *All languages in BPP have polynomial circuits.*

- Our proof will be *nonconstructive* in that only the existence of the desired circuits is shown.
  - Recall our proof of Theorem 18 (p. 214).
  - Something exists if its probability of existence is nonzero.
- It is not known how to efficiently generate circuit C_n.
  - If the construction of C_n can be made efficient, then P = BPP, an unlikely result.

The Proof

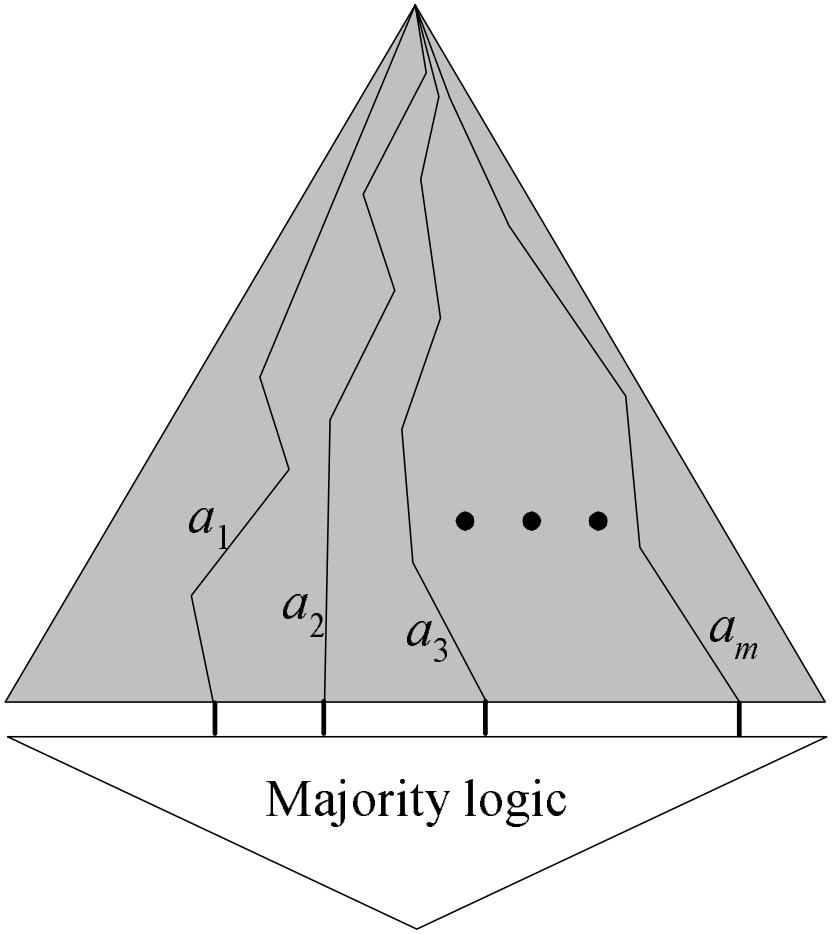
- Let $L \in \text{BPP}$ be decided by a precise polynomial-time NTM N by clear majority.
- We shall prove that L has polynomial circuits $C_0, C_1, \ldots$.
  - These *deterministic* circuits do not err.
- Suppose N runs in time $p(n)$, where $p(n)$ is a polynomial.
- Let $A_n = \{ a_1, a_2, \ldots, a_m \}$, where $a_i \in \{ 0, 1 \}^{p(n)}$.
- Each $a_i \in A_n$ represents a sequence of nondeterministic choices (i.e., a computation path) for N.
- Pick $m = 12(n + 1)$.

The Proof (continued)

- Let x be an input with $|\,x\,| = n$.
- Circuit C_n simulates N on x with all sequences of choices in A_n and then takes the majority of the m outcomes.[a]
  - Note that each A_n yields a circuit.
- As N with a_i is a polynomial-time deterministic TM, it can be simulated by polynomial circuits of size $O(p(n)^2)$.
  - See the proof of Proposition 78 (p. 616).

[a] As m is even, there may be no clear majority. Still, the probability of that happening is very small and does not materially affect our general conclusion. Thanks to a lively class discussion on December 14, 2010.

The Circuit



The Proof (continued)

- The size of C_n is therefore $O(mp(n)^2) = O(np(n)^2)$.
  - This is a polynomial.
- We now confirm the existence of an A_n making C_n correct on *all* n-bit inputs.
- Call a_i **bad** if it leads N to an error (a false positive or a false negative) for x.
- Select A_n uniformly randomly.

The Proof (continued)

- For each $x \in \{ 0, 1 \}^n$, 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 $m/4$.[a]
- By the Chernoff bound (p. 595), 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 error probability of using the majority rule is thus $< 2^{-(n+1)}$ for each $x \in \{ 0, 1 \}^n$.

[a] So the proof will not work for NP. Contributed by Mr. Ching-Hua Yu (D00921025) on December 11, 2012.

The Proof (continued)

- The probability that there is an x such that A_n results in an incorrect answer is

$$< 2^n 2^{-(n+1)} = 2^{-1}.$$

  - Recall the union bound:
    $\text{prob}[\, A \cup B \cup \cdots \,] \leq \text{prob}[\, A \,] + \text{prob}[\, B \,] + \cdots$
    (**Boole's inequality**).

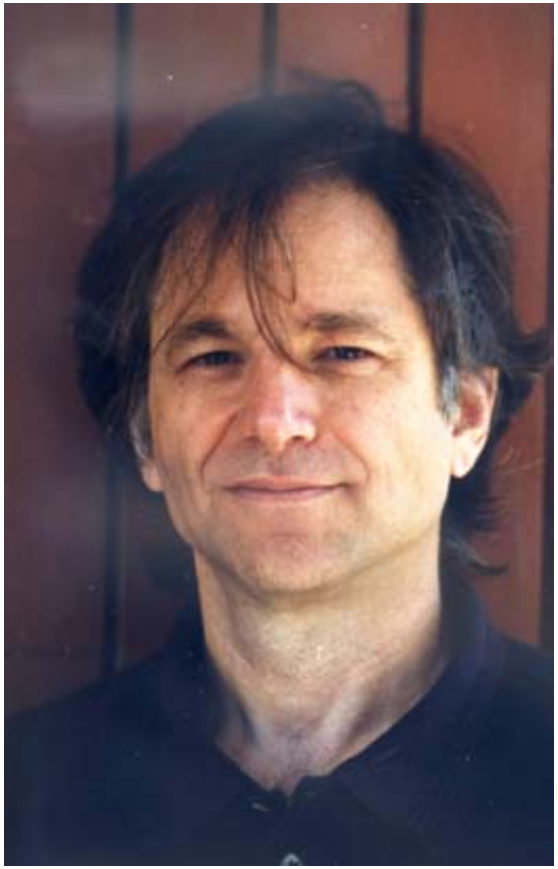
- We just showed that at least half of them are correct.
- So with probability ≥ 0.5, a random A_n produces a correct C_n for *all* inputs of length n.
  - Of course, verifying this fact may take a long time.

The Proof (concluded)

- 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 exists.[a]
- We have used the **probabilistic method**.[b]
- This result answers the question on p. 525 with a "yes."

[a] Quine (1948), "To be is to be the value of a bound variable."
[b] A counting argument in the probabilistic language.

Leonard Adleman[a] (1945–)



[a]Turing Award (2002).

Cryptography

Whoever wishes to keep a secret
must hide the fact that he possesses one.
— Johann Wolfgang von Goethe (1749–1832)

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**.

Encryption and Decryption

- Alice and Bob agree on two algorithms E and D—the **encryption** and the **decryption 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 **encryption** and **decryption 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**.[a]

[a] Both "zero" and "cipher" come from the same Arab word.

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 x from y 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 plaintext that this ciphertext represents are identical to the a priori probabilities of the same plaintext before the interception.
  - The probability that plaintext $\mathcal{P}$ occurs is independent of the ciphertext $\mathcal{C}$ being observed.
  - So knowing $\mathcal{C}$ yields no advantage in recovering $\mathcal{P}$.
- Such systems are said to be **informationally secure**.
- A system is **computationally secure** if breaking it is theoretically possible but computationally infeasible.

Conditions for Perfect Secrecy[a]

- Consider a cryptosystem where:
  - The space of ciphertext is as large as that of keys.
  - Every plaintext has a nonzero probability of being used.
- It is **perfectly secure** if and only if the following hold.
  - A key is chosen with uniform distribution.
  - For each plaintext x and ciphertext y, there exists a unique key e such that $E(e, x) = y$.

[a]Shannon (1949).

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 $x \oplus r$ to Bob over a public channel;

4: Bob receives y;

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

---

[a]Mauborgne and Vernam (1917); Shannon (1949). It was allegedly used for the hotline between Russia and U.S.

Analysis

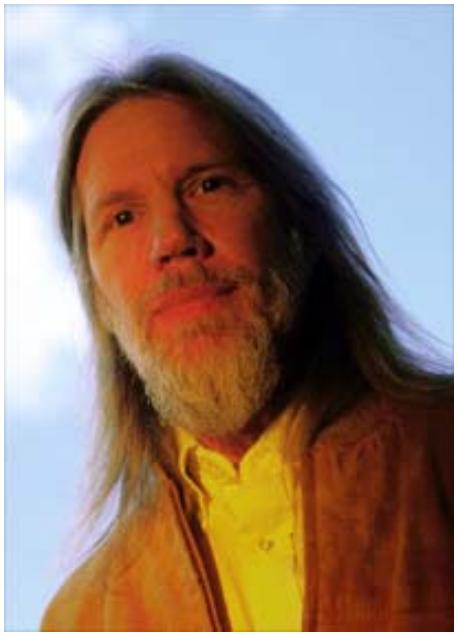
- The one-time pad uses $e = d = r$.
- This is said to be a **private-key cryptosystem**.
- Knowing x and knowing r are equivalent.
- Because r is random and private, the one-time pad achieves perfect secrecy (see also p. 638).
- The random bit string must be new for each round of communication.
  - **Cryptographically strong pseudorandom generators** require exchanging only the seed once.
- But the assumption of a private channel is problematic.

Public-Key Cryptography[a]

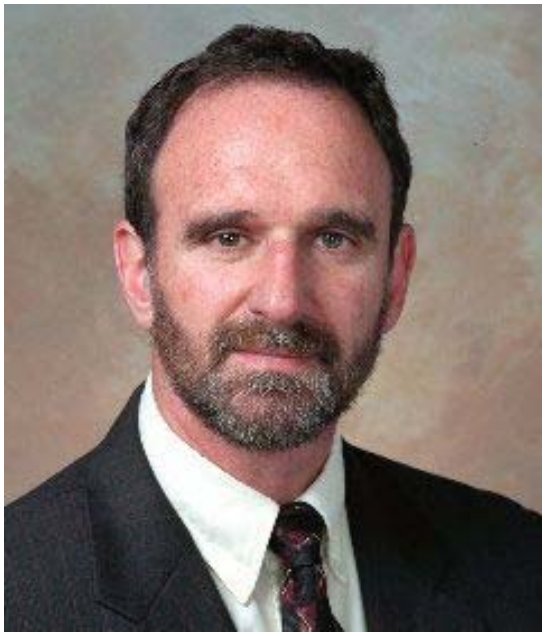
- Suppose only d is private to Bob, whereas e is public knowledge.
- Bob generates the (e, d) pair and publishes e.
- Anybody like Alice can send $E(e, x)$ to Bob.
- Knowing d, Bob can recover x by $D(d, E(e, x)) = x$.
- The assumptions are complexity-theoretic.
  - It is computationally difficult to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

[a]Diffie and Hellman (1976).

Whitfield Diffie (1944–)



Martin Hellman (1945–)



Complexity Issues

- Given y and x, it is easy to verify whether $E(e, x) = y$.
- Hence one can always guess an x and verify.
- Cracking a public-key cryptosystem is thus in NP.
- A *necessary* condition for the existence of secure public-key cryptosystems is $\mathrm{P} \neq \mathrm{NP}$.
- But more is needed than $\mathrm{P} \neq \mathrm{NP}$.
- For instance, it is not sufficient that D is hard to compute in the *worst* case.
- It should be hard in "most" or "average" cases.

One-Way Functions

A function f is a **one-way function** if the following hold.[a]

1. f is one-to-one.
2. For all $x \in \Sigma^*$, $|\, x \,|^{1/k} \leq |f(x)| \leq |\, x \,|^k$ for some $k > 0$.
   - f is said to be **honest**.
3. f can be computed in polynomial time.
4. f^{-1} cannot be computed in polynomial time.
   - Exhaustive search works, but it must be slow.

[a] Diffie and Hellman (1976); Boppana and Lagarias (1986); Grollmann and Selman (1988); Ko (1985); Ko, Long, and Du (1986); Watanabe (1985); Young (1983).

Existence of One-Way Functions

- Even if P $\neq$ NP, there is no guarantee that one-way functions exist.
- No functions have been proved to be one-way.
- Is breaking glass a one-way function?

Candidates of One-Way Functions

- Modular exponentiation $f(x) = g^x \bmod p$, where g is a primitive root of p.
  - **Discrete logarithm** is hard.[a]
- The RSA[b] function $f(x) = x^e \bmod pq$ for an odd e relatively prime to $\phi(pq)$.
  - Breaking the RSA function is hard.

---

[a] Conjectured to be 2^{n^ϵ} for some $\epsilon > 0$ in both the worst-case sense and average sense. Doable in time $n^{O(\log n)}$ for finite fields of small characteristic (Barbulescu, et al., 2013). It is in NP in some sense (Grollmann and Selman, 1988).

[b] Rivest, Shamir, and Adleman (1978).

Candidates of One-Way Functions (concluded)

- Modular squaring $f(x) = x^2 \bmod pq$.
  - Determining if a number with a Jacobi symbol 1 is a quadratic residue is hard—the **quadratic residuacity assumption** (**QRA**).[a]
  - Breaking it is as hard as factorization when $p \equiv q \equiv 3 \bmod 4$.[b]

[a] Due to Gauss.
[b] Rabin (1979).

The Secret-Key Agreement Problem

- Exchanging messages securely using a private-key cryptosystem requires Alice and Bob possessing the same key (p. 640).
  - An example is the r in the one-time pad (p. 639).
- How can they agree on the same secret key when the channel is insecure?
- This is called the **secret-key agreement problem**.
- It was solved by Diffie and Hellman (1976) using one-way functions.

The Diffie-Hellman Secret-Key Agreement Protocol

1: Alice and Bob agree on a large prime p and a primitive root g of p; {p and g are public.}
2: Alice chooses a large number a at random;
3: Alice computes $\alpha = g^a \bmod p$;
4: Bob chooses a large number b at random;
5: Bob computes $\beta = g^b \bmod p$;
6: Alice sends α to Bob, and Bob sends β to Alice;
7: Alice computes her key $\beta^a \bmod p$;
8: Bob computes his key $\alpha^b \bmod p$;

Analysis

- The keys computed by Alice and Bob are identical as

$$\beta^a = g^{ba} = g^{ab} = \alpha^b \bmod p.$$

- To compute the common key from p, g, α, β is known as the **Diffie-Hellman problem**.
- It is conjectured to be hard.
- If discrete logarithm is easy, then one can solve the Diffie-Hellman problem.
  - Because a and b can then be obtained by Eve.
- But the other direction is still open.

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 56 (p. 471),

$$\phi(pq) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1. \quad (15)$$

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

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

  which can be found by the Euclidean algorithm.[a]

[a] One can think of d as e^{-1}.

A Public-Key Cryptosystem Based on RSA

- Bob generates p and q.
- Bob publishes pq and the encryption key e, a number relatively prime to $\phi(pq)$.
  - The encryption function is $y = x^e \bmod pq$.
  - Bob calculates $\phi(pq)$ by Eq. (15) (p. 652).
  - Bob then 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 when $\gcd(x, pq) = 1$ (p. 482).

The "Security" of the RSA Function

- Factoring pq or calculating d from (e, pq) seems hard.[a]
- Breaking the last bit of RSA is as hard as breaking the RSA.[b]
- Recommended RSA key sizes:[c]
  - 1024 bits up to 2010.
  - 2048 bits up to 2030.
  - 3072 bits up to 2031 and beyond.

---

[a] See also p. 478.

[b] Alexi, Chor, Goldreich, and Schnorr (1988).

[c] RSA (2003). RSA was acquired by EMC in 2006 for 2.1 billion US dollars.

The "Security" of the RSA Function (continued)

- Recall that problem A is "harder than" problem B if solving A results in solving B.
  - Factorization is "harder than" breaking the RSA.
  - It is not hard to show that calculating Euler's phi function[a] is "harder than" breaking the RSA.
  - Factorization is "harder than" calculating Euler's phi function (see Lemma 56 on p. 471).
  - So factorization is harder than calculating Euler's phi function, which is harder than breaking the RSA.

[a] When the input is not factorized!

The "Security" of the RSA Function (concluded)

- Factorization cannot be NP-hard unless NP = coNP.[a]
- So breaking the RSA is unlikely to imply P = NP.
- But numbers can be factorized efficiently by quantum computers.[b]
- RSA was alleged to have received 10 million US dollars from the government to promote unsecure p and q![c]

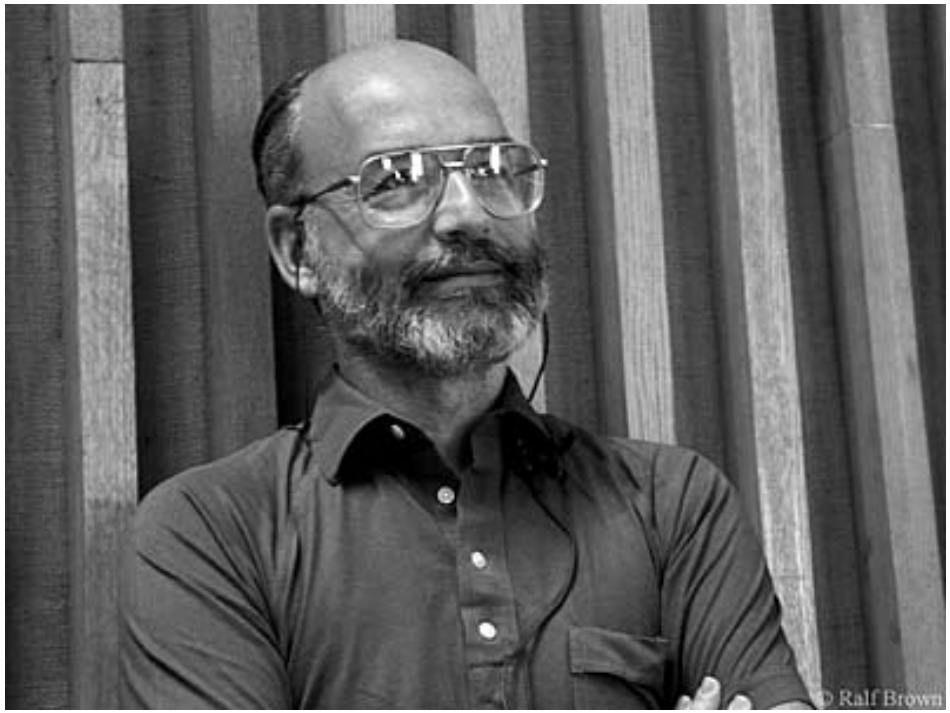
[a] Brassard (1979).
[b] Shor (1994).
[c] Menn (2013).

Adi Shamir, Ron Rivest, and Leonard Adleman

Ron Rivest[a] (1947–)

[a]Turing Award (2002).

Adi Shamir[a] (1952–)



[a]Turing Award (2002).

A Parallel History

- Diffie and Hellman's solution to the secret-key agreement problem led to public-key cryptography.
- At around the same time (or earlier) in Britain, the RSA public-key cryptosystem was invented first before the Diffie-Hellman secret-key agreement scheme was.
  - Ellis, Cocks, and Williamson of the Communications Electronics Security Group of the British Government Communications Head Quarters (GCHQ).

Is a forged signature the same sort of thing
as a genuine signature,
or is it a different sort of thing?
— Gilbert Ryle (1900–1976),
The Concept of Mind (1949)

"Katherine, I gave him the code.
He verified the code."
"But did you verify him?"
— *The Numbers Station* (2013)