The Circuit Complexity of P

Proposition 80 *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. 319), there is a circuit with $O(p(n)^2)$ gates that accepts $L \cap \{0, 1\}^n$.
- The size of that circuit depends only on L and the length of the input.
- The size of that circuit is polynomial in n.

Polynomial Circuits vs. P

- Is the converse of Proposition 80 true?
  - Do polynomial circuits accept only languages in P?
- No.
- Polynomial circuits can accept *undecidable* languages![a]

[a] See p. 268 of the textbook.

BPP's Circuit Complexity: Adleman's Theorem

Theorem 81 (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 16 (p. 212).
  - 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

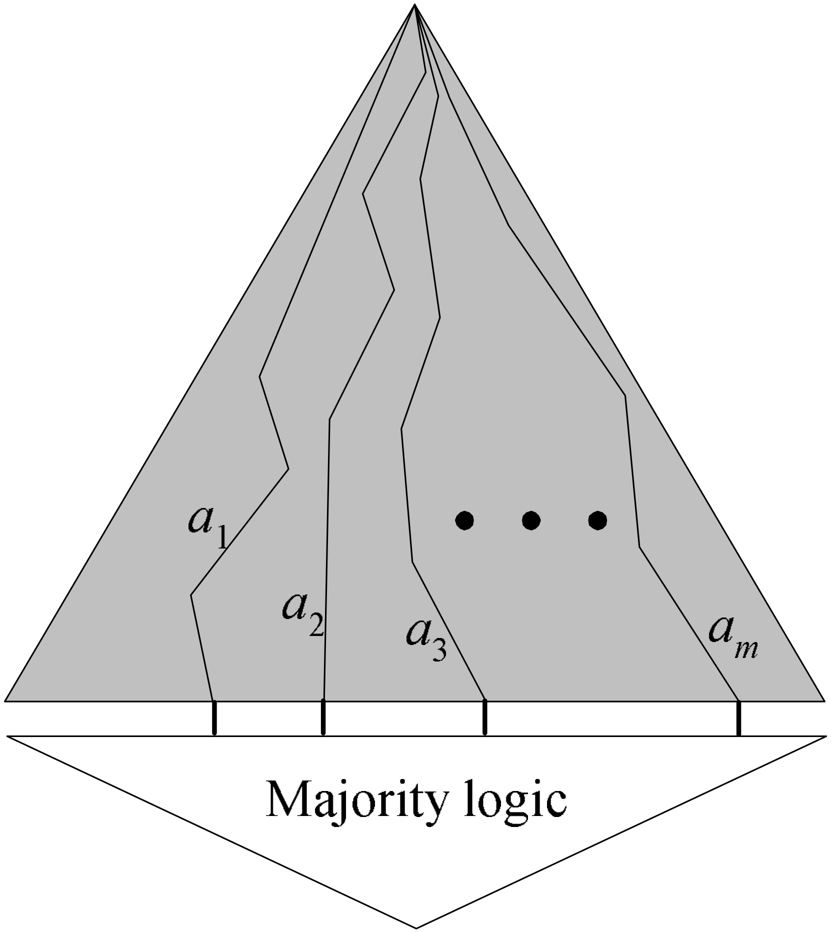
- Let $L \in$ BPP be decided by a precise polynomial-time NTM N by clear majority.
- We shall prove that L has polynomial circuits $C_0, C_1, \ldots$.
  - These *deterministic* circuits do not err.
- Suppose N runs in time $p(n)$, where $p(n)$ is a polynomial.
- Let $A_n = \{ a_1, a_2, \ldots, a_m \}$, 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 80 (p. 626).

[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]
- Also note after fixing the input x, the circuit is a function of the random bits.

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

The Proof (continued)

- By the Chernoff bound (p. 605), 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 $x \in \{ 0, 1 \}^n$.

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

- We just showed that at least half of the random A_n 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)

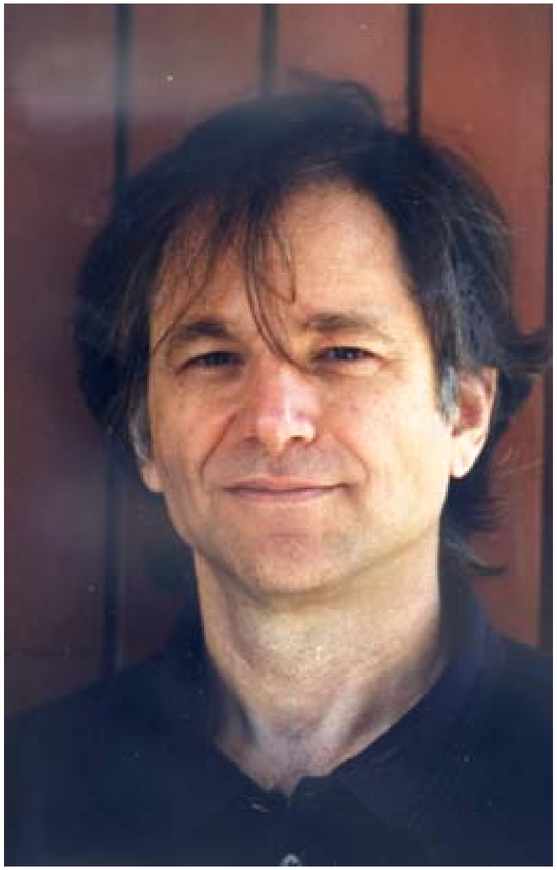
- Because this probability exceeds 0, an A_n that makes majority vote work for *all* inputs of length n exists.
- Hence a correct C_n exists.[a]
- We have used the **probabilistic method**[b] popularized by Erdős (1947).[c]
- This result answers the question on p. 537 with a "yes."

[a] Quine (1948), "To be is to be the value of a bound variable."

[b] A counting argument in the probabilistic language.

[c] Szele (1943) and Turán (1934) were earlier.

Leonard Adleman[a] (1945–)



[a] Turing Award (2002).

Paul Erdős (1913–1996)

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**.

$$\text{Alice} \xrightarrow{\text{Eve}} \text{Bob}$$

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.

Encryption and Decryption (concluded)

- 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.

Degree 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 $\mathcal{P}$.

Degree of Security (concluded)

- 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]

```
Alice generates a random string $r$ as long as $x$;
Alice sends $r$ to Bob over a secret channel;
Alice sends $x \oplus r$ to Bob over a public channel;
Bob receives $y$;
Bob recovers $x := y \oplus r$;
```

[a] Mauborgne & Vernam (1917); Shannon (1949). It was allegedly used for the hotline between Russia and the 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.[a]
- The random bit string must be new for each round of communication.
- But the assumption of a private channel is problematic.

[a] See p. 647.

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 via

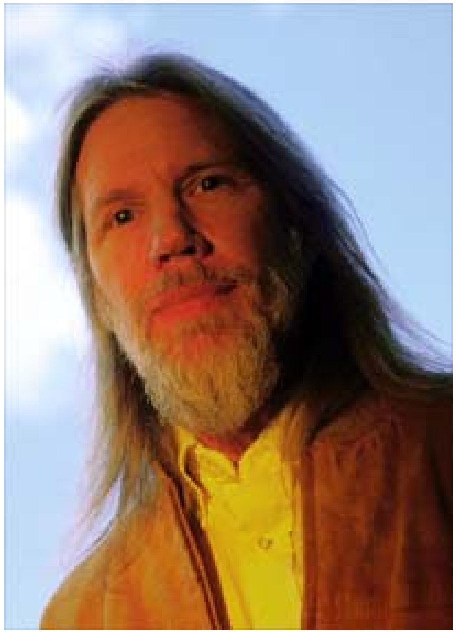
$$D(d, E(e, x)) = x.$$

[a] Diffie & Hellman (1976).

Public-Key Cryptography (concluded)

- 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.

Whitfield Diffie[a] (1944–)



[a] Turing Award (2016).

Martin Hellman[a] (1945–)

[a] Turing Award (2016).

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 P $\neq$ NP.
- But more is needed than P $\neq$ 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 & Hellman (1976); Boppana & Lagarias (1986); Grollmann & Selman (1988); Ko (1985); Ko, Long, & Du (1986); Watanabe (1985); Young (1983).

Existence of One-Way Functions (OWFs)

- 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 & Selman, 1988).

[b] Rivest, Shamir, & 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 have the *same* key.[a]
  - An example is the r in the one-time pad.[b]
- 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.

[a] See p. 649.

[b] See p. 648.

The Diffie-Hellman Secret-Key Agreement Protocol

```
Alice and Bob agree on a large prime $p$ and a primitive root $g$ of $p$; {$p$ and $g$ are public.}
Alice chooses a large number $a$ at random;
Alice computes $\alpha = g^a \bmod p$;
Bob chooses a large number $b$ at random;
Bob computes $\beta = g^b \bmod p$;
Alice sends $\alpha$ to Bob, and Bob sends $\beta$ to Alice;
Alice computes her key $\beta^a \bmod p$;
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.[a]
- 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.

[a] This is the **computational Diffie-Hellman assumption** (CDH).

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 59 (p. 490),
$$\phi(pq) = pq \left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1. \quad (16)$$
- 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. (16) (p. 662).
  - Bob then calculates d such that $ed = 1 + k\phi(pq)$ for some $k \in \mathbb{Z}$.

A Public-Key Cryptosystem Based on RSA (continued)

- 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. 495).

A Public-Key Cryptosystem Based on RSA (continued)

- What if x is not relatively prime to pq?[a]
- As $\phi(pq) = (p-1)(q-1)$,

$$ed = 1 + k(p-1)(q-1).$$

- Say $x \equiv 0 \bmod p$.
- Then

$$y^d \equiv x^{ed} \equiv 0 \equiv x \bmod p.$$

[a] Of course, one would be unlucky here.

A Public-Key Cryptosystem Based on RSA (continued)

- On the other hand, either $x \not\equiv 0 \bmod q$ or $x \equiv 0 \bmod q$.
- If $x \not\equiv 0 \bmod q$, then

$$\begin{aligned} y^d &\equiv x^{ed} \equiv x^{ed-1} x \equiv x^{k(p-1)(q-1)} x \equiv \left(x^{q-1} \right)^{k(p-1)} x \\ &\equiv x \bmod q. \end{aligned}$$

by Fermat's "little" theorem (p. 493).

- If $x \equiv 0 \bmod q$, then

$$y^d \equiv x^{ed} \equiv 0 \equiv x \bmod q.$$

A Public-Key Cryptosystem Based on RSA (concluded)

- By the Chinese remainder theorem (p. 492),

$$y^d \equiv x^{ed} \equiv 0 \equiv x \bmod pq,$$

  even when x is not relatively prime to p.

- When x is not relatively prime to q, the same conclusion holds.

The "Security" of the RSA Function

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

[a] Alexi, Chor, Goldreich, & Schnorr (1988).

[b] 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 59 on p. 490).
  - 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]

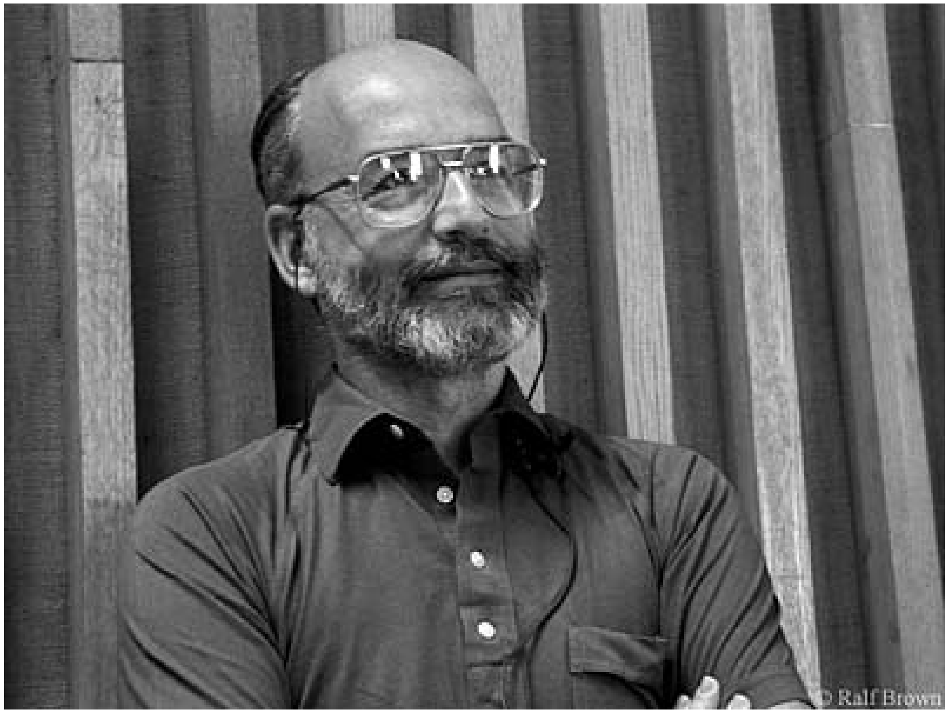
[a] Brassard (1979).
[b] Shor (1994).
[c] Menn (2013).

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.
- In 1973, the RSA public-key cryptosystem was invented in Britain before the Diffie-Hellman secret-key agreement scheme.[a]

[a] 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)

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

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

- Every cryptosystem guarantees $D(d, E(e, x)) = x$.
- Assume the cryptosystem also satisfies the commutative property

$$E(e, D(d, x)) = D(d, E(e, x)). \tag{17}$$

  - E.g., the RSA system satisfies it as $(x^d)^e = (x^e)^d$.

[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. (17).
- The claim of authenticity is founded on the difficulty of inverting E_{Alice} without knowing the key d_{Alice}.

Blind Signatures[a]

- There are applications where the document author (Alice) and the signer (Bob) are *different* parties.
- Sender privacy: We do not want Bob to see the document.
  - Anonymous electronic voting systems, digital cash schemes, anonymous payments, etc.
- Idea: The document is **blinded** by Alice before it is signed by Bob.
- The resulting blind signature can be publicly verified against the original, unblinded document x as before.

[a] Chaum (1983).

Blind Signatures Based on RSA

Blinding by Alice:

```
1: Pick $r \in Z_n^*$ randomly;
2: Send
       $$x' = xr^e \bmod n$$
   to Bob; {$x$ is blinded by $r^e$.}
```

- Note that $r \to r^e \bmod n$ is a one-to-one correspondence.
- Hence $r^e \bmod n$ is a random number, too.
- As a result, x' is random and leaks no information, even if x has any structure.

Blind Signatures Based on RSA (continued)

Signing by Bob with his private decryption key d:

1: Send the blinded signature

$$s' = (x')^d \bmod n$$

to Alice;

Blind Signatures Based on RSA (continued)

The RSA signature of Alice:

1: Alice obtains the signature $s = s' r^{-1} \bmod n$;

- This works because

  $$s \equiv s' r^{-1} \equiv (x')^d r^{-1} \equiv (x r^e)^d r^{-1} \equiv x^d r^{ed-1} \equiv x^d \bmod n$$

  by the properties of the RSA function.

- Note that only Alice knows r.

Blind Signatures Based on RSA (concluded)

- Anyone can verify the document was signed by Bob by checking with Bob's encryption key e the following:

$$s^e \equiv x \bmod n.$$

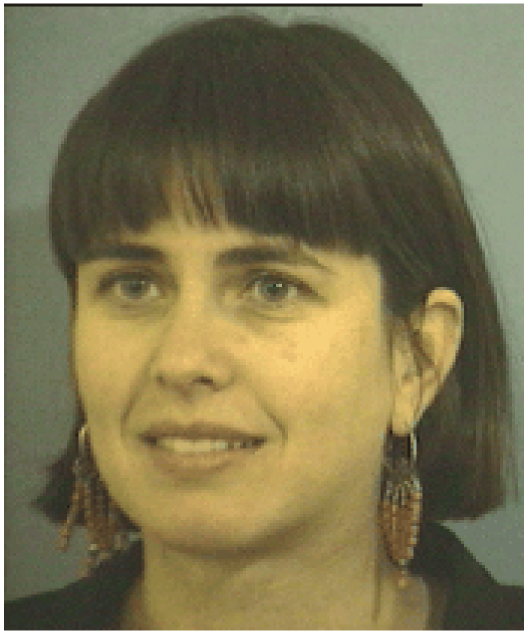
- But Bob does not know s is related to x' (thus Alice).

Probabilistic Encryption[a]

- A deterministic cryptosystem can be broken if the plaintext has a distribution that favors the "easy" cases.
- The ability to forge signatures on even a vanishingly small fraction of strings of some length is a security weakness if those strings were the probable ones!
- A scheme may also "leak" *partial* information.
  - Parity of the plaintext, e.g.
- The first solution to the problems of skewed distribution and partial information was based on the QRA.

[a] Goldwasser & Micali (1982). This paper "laid the framework for modern cryptography" (2013).

Shafi Goldwasser[a] (1958–)



[a] Turing Award (2013).

Silvio Micali[a] (1954–)

[a] Turing Award (2013).

Goldwasser and Micali

A Useful Lemma

Lemma 82 *Let $n = pq$ be a product of two distinct primes. Then a number $y \in Z_n^*$ is a quadratic residue modulo n if and only if $(y \,|\, p) = (y \,|\, q) = 1$.*

- The "only if" part:
  - Let x be a solution to $x^2 = y \bmod pq$.
  - Then $x^2 = y \bmod p$ and $x^2 = y \bmod q$ also hold.
  - Hence y is a quadratic modulo p and a quadratic residue modulo q.

The Proof (concluded)

- The "if" part:
  - Let $a_1^2 = y \bmod p$ and $a_2^2 = y \bmod q$.
  - Solve

    $$\begin{aligned} x &= a_1 \bmod p, \\ x &= a_2 \bmod q, \end{aligned}$$

    for x with the Chinese remainder theorem (p. 492).
  - As $x^2 = y \bmod p$, $x^2 = y \bmod q$, and $\gcd(p, q) = 1$, we must have $x^2 = y \bmod pq$.

The Jacobi Symbol and Quadratic Residuacity Test

- The Legendre symbol can be used as a test for quadratic residuacity by Lemma 69 (p. 560).
- Lemma 82 (p. 687) says this is *not* the case with the Jacobi symbol in general.
- Suppose $n = pq$ is a product of two distinct primes.
- A number $y \in Z_n^*$ with Jacobi symbol $(y \,|\, pq) = 1$ is a quadratic *nonresidue* modulo n when
$$(y \,|\, p) = (y \,|\, q) = -1,$$
because $(y \,|\, pq) = (y \,|\, p)(y \,|\, q)$.

The Setup

- Bob publishes $n = pq$, a product of two distinct primes, and a quadratic nonresidue y with Jacobi symbol 1.
- Bob keeps secret the factorization of n.
- Alice wants to send bit string $b_1 b_2 \cdots b_k$ 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.
- So 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

```
for $i = 1, 2, \ldots, k$ do
    Pick $r \in Z_n^*$ randomly;
    if $b_i = 1$ then
        Send $r^2 \bmod n$; {Jacobi symbol is 1.}
    else
        Send $r^2 y \bmod n$; {Jacobi symbol is still 1.}
    end if
end for
```

The Protocol for Bob

```
for $i = 1, 2, \ldots, k$ do
    Receive $r$;
    if $(r \mid p) = 1$ and $(r \mid q) = 1$ then
        $b_i := 1$;
    else
        $b_i := 0$;
    end if
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.
  - Encryption is a *one-to-many* mapping.
- This scheme is both polynomially secure and **semantically secure**.

What then do you call proof?
— Henry James (1843–1916),
The Wings of the Dove (1902)

Leibniz knew what a proof is.
Descartes did not.
— Ian Hacking (1973)

What Is a Proof?

- A proof convinces a party of a certain claim.
  - "$x^n + y^n \neq z^n$ for all $x, y, z \in \mathbb{Z}^+$ and $n > 2$."
  - "Graph G is Hamiltonian."
  - "$x^p = x \bmod p$ for prime p and $p \not| x$."
- In mathematics, a proof is a fixed sequence of theorems.
  - Think of it as a written examination.
- We will extend a proof to cover a proof *process* by which the validity of the assertion is established.
  - Recall a job interview or an oral examination.

Prover and Verifier

- There are two parties to a proof.
  - The **prover** (**Peggy**).
  - The **verifier** (**Victor**).
- Given an assertion, the prover's goal is to convince the verifier of its validity (**completeness**).
- The verifier's objective is to accept only correct assertions (**soundness**).
- The verifier usually has an easier job than the prover.
- The setup is similar to the Turing test.[a]

[a] Turing (1950).

Interactive Proof Systems

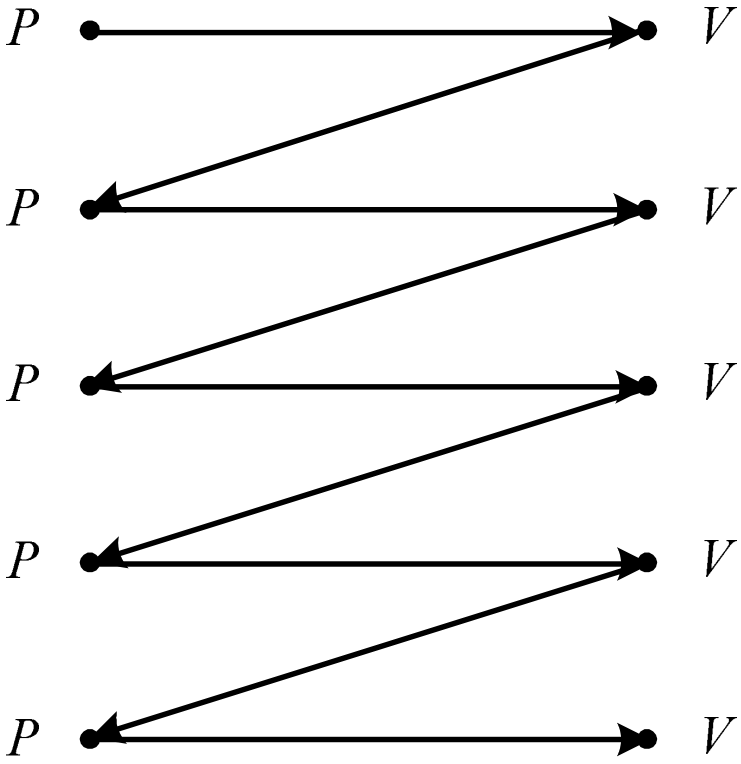
- An **interactive proof** for a language L is a sequence of questions and answers between the two parties.
- At the end of the interaction, the verifier decides whether the claim is true or false.
- The verifier must be a probabilistic polynomial-time algorithm.
- The prover runs an exponential-time algorithm.[a]
  - If the prover is not more powerful than the verifier, no interaction is needed!

[a] See the problem to Note 12.3.7 on p. 296 and Proposition 19.1 on p. 475, both of the textbook, about alternative complexity assumptions without affecting the definition. Contributed by Mr. Young-San Lin (B97902055) and Mr. Chao-Fu Yang (B97902052) on December 18, 2012.

Interactive Proof Systems (concluded)

- The system decides L if the following two conditions hold for any common input x.
  - If $x \in L$, then the probability that x is accepted by the verifier is at least $1 - 2^{-|\,x\,|}$.
  - If $x \notin L$, then the probability that x is accepted by the verifier with *any* prover replacing the original prover is at most $2^{-|\,x\,|}$.
- Neither the number of rounds nor the lengths of the messages can be more than a polynomial of $|\,x\,|$.

An Interactive Proof



IP[a]

- **IP** is the class of all languages decided by an interactive proof system.
- When $x \in L$, the completeness condition can be modified to require that the verifier accept with certainty without affecting IP.[b]
- Similar things cannot be said of the soundness condition when $x \notin L$.
- Verifier's coin flips can be public.[c]

[a] Goldwasser, Micali, & Rackoff (1985).
[b] Goldreich, Mansour, & Sipser (1987).
[c] Goldwasser & Sipser (1989).

The Relations of IP with Other Classes

- NP $\subseteq$ IP.
  - IP becomes NP when the verifier is deterministic and there is only one round of interaction.[a]
- BPP $\subseteq$ IP.
  - IP becomes BPP when the verifier ignores the prover's messages.
- IP = PSPACE.[b]

[a] Recall Proposition 41 on p. 335.

[b] Shamir (1990).