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 gives?
- 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.
  - Circuits now cannot accept undecidable languages (why?).
  - The circuit family on p. 484 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 70 *$L \in P$ if and only if L has uniformly polynomial circuits.*

- One direction was proved in Proposition 69 (p. 483).
- Now suppose L has uniformly polynomial circuits.
- Decide $x \in L$ in polynomial time as follows:
  - Let $n = |\, x \,|$.
  - Build C_n in $\log n$ space, hence polynomial time.
  - Evaluate the circuit with input x in polynomial time.
- Therefore $L \in$ P.

Relation to P vs. NP

- Theorem 70 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 the P $\neq$ NP conjecture—without success so far.
  - Theorem 14 (p. 153) 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 71 (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.
  - Something exists if its probability of existence is nonzero.
- How to efficiently generate circuit C_n given 1^n is not known.
- If the construction of C_n is efficient, then $\mathrm{P} = \mathrm{BPP}$, an unlikely result.

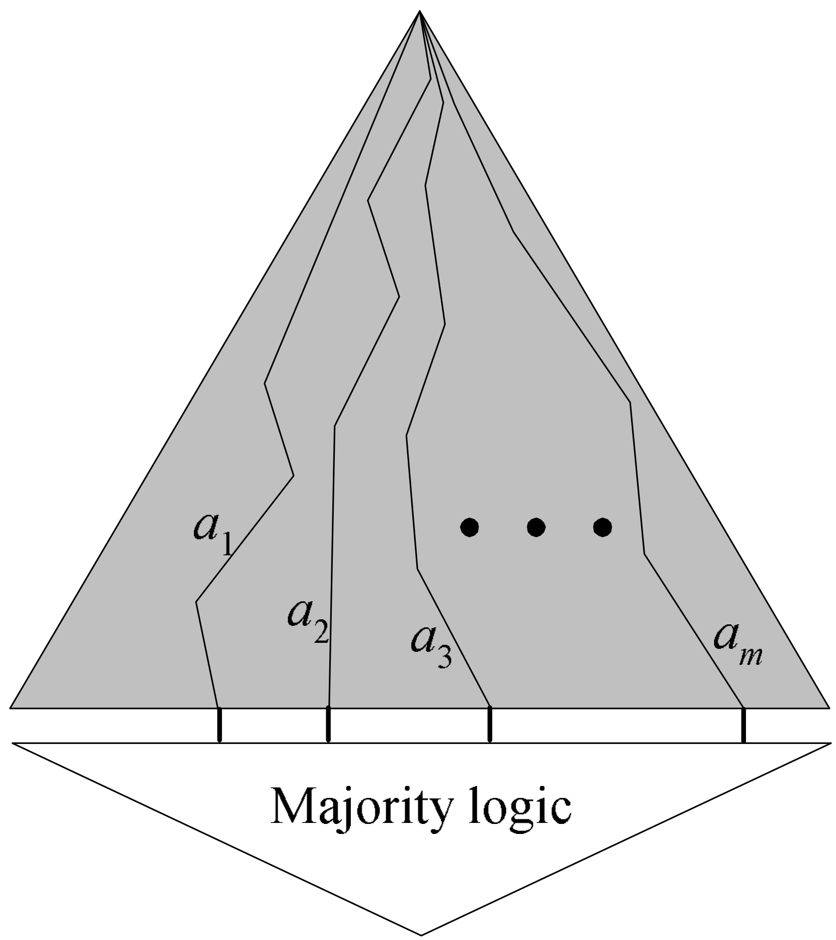
The Proof

- Let $L \in \text{BPP}$ be decided by a precise NTM N by clear majority.
- We shall prove that L has polynomial circuits $C_0, C_1, \ldots$.
- Suppose N runs in time $p(n)$, where $p(n)$ is a polynomial.
- Let $A_n = \{a_1, a_2, \ldots, a_m\}$, where $a_i \in \{0, 1\}^{p(n)}$.
- Let $m = 12(n + 1)$.
- 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 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 69 (p. 483).
- The size of C_n is therefore $O(mp(n)^2) = O(np(n)^2)$, 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 $x \in \{0, 1\}^n$, $1/4$ 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. 464), 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 (concluded)

- 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 $< 2^n 2^{-(n+1)} = 2^{-1}$.
  - $\text{prob}[\, A \cup B \cup \cdots \,] \leq \text{prob}[\, A \,] + \text{prob}[\, B \,] + \cdots$.
- 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 exists.

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

4: Bob receives y;

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

---

[a] Mauborgne and Vernam (1917), Shannon (1949); 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. 501).
- 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 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).

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 $\mathrm{P} \neq \mathrm{NP}$.
- 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 is too 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 a glass a one-way function?

UP[a]

- An NTM that has at most one accepting computation for any input is called an **unambiguous Turing machine** (**UTM**).
- UP denotes the set of languages accepted by UTMs in polynomial time.
- Obviously, P $\subseteq$ UP $\subseteq$ NP.

[a]Valiant (1976).

SAT and UP

- SAT is not expected to be in UP (so UP $\neq$ NP).
  - Suppose SAT $\in$ UP.
  - Then there is an NTM M that has a single accepting computation path for all satisfiable boolean expressions.
  - But M runs in polynomial time.
  - Hence M does not try all truth assignments for satisfiable boolean expressions.
  - At present, this seems implausible.

UP and One-Way Functions[a]

Theorem 72 *One-way functions exist if and only if $P \neq UP$.*

- Suppose there exists a one-way function f.
- Define language
  $L_f \equiv \{ (x, y) : \exists z \text{ such that } f(z) = y \text{ and } z \leq x \}$.
  - Relation "$\leq$" orders strings of $\{ 0, 1 \}^*$ first by length and then lexicographically.
  - So $\epsilon < 0 < 1 < 00 < 01 < 10 < 11 < \cdots$.

[a] Ko (1985); Grollmann and Selman (1988).

The Proof (continued)

- $L_f \in$ UP.
  - There is an UTM M that accepts L_f.
    - M on input (x, y) nondeterministically guesses a string z of length at most $|\, y \,|^k$.
    - M tests if $y = f(z)$.
    - If the answer is "yes" (this happens at most once because f is one-to-one) and $z \leq x$, M accepts.

The Proof (continued)

- $L_f \notin$ P.
  - Suppose there is a polynomial-time algorithm for L_f.
  - Then $f(x) = y$ can be inverted.
    * Given y, ask $(1^{|y|^k}, y) \in L_f$.
    * If the answer is "no," we know x does not exist as any such x must satisfy $|x| \leq |y|^k$.
    * Otherwise, ask $(1^{|y|^k-1}, y) \in L_f, (1^{|y|^k-2}, y) \in L_f, \ldots$ until we got a "no" for $(1^{\ell-1}, y) \in L_f$.
    * This means $|x| = \ell$.
  - The procedure makes $O(|y|^k)$ calls to L_f.

The Proof (continued)

- (continued)
  - * Now conduct a binary search to find each bit of x as follows.
    * If $(01^{\ell-1}, y) \in L_f$, then $x = 0 \cdots$ and we recur by asking "$(001^{\ell-2}, y) \in L_f$?"
    * If $(01^{\ell-1}, y) \notin L_f$, then $x = 1 \cdots$ and we recur by asking $(101^{\ell-2}, y) \in L_f$?"
  - The procedure makes $O(|\, y \,|^k)$ calls to L_f.
- $\mathrm{P} \neq \mathrm{UP}$ because $L_f \in \mathrm{UP} - \mathrm{P}$.

The Proof (continued)

- Now suppose $\text{P} \neq \text{UP}$ with $L \in \text{UP} - \text{P}$.
- Let L be accepted by an UTM M.
- $\text{comp}_M(y)$ denotes an accepting computation of $M(y)$.
- Define

$$f_M(x) = \begin{cases} 1y & \text{if } x = \text{comp}_M(y), \\ 0x & \text{otherwise.} \end{cases}$$

- f_M is well-defined as y is part of $\text{comp}_M(y)$ (recall p. 238) and there is at most one accepting computation for y.
- So f_M is a total function.

The Proof (concluded)

- f_M is one-way.
  - The lengths of argument and results are polynomially related as M has polynomially long computations.
  - f_M is one-to-one because $f(x) = f(x')$ means that $x = x'$ by the use of the flag and unambiguity of M.
  - f_M can be inverted on $1y$ if and only if M accepts y (i.e., if $y \in L$).
  - Were we able to invert f_M in polynomial time, then we would be able to decide L in polynomial time.

Complexity Issues

- For a language in UP, there is either 0 or 1 accepting path.
- So similar to RP, there are not likely to be UP-complete problems.
- Relating a cryptosystem with an NP-complete problem has been argued before to be not useful (p. 505).
- Theorem 72 (p. 510) shows that the relevant question is the P = UP question.
- There are stronger notions of one-way functions.

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.
- 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] But it is in NP in some sense; Grollmann and Selman (1988).
[b] Rivest, Shamir, and 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 49 (p. 359),
$$\phi(pq) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1.$$
- As $\gcd(e, \phi(pq)) = 1$, there is a d such that
$$ed \equiv 1 \bmod \phi(pq),$$
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 encryption 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 when $\gcd(x, pq) = 1$ (p. 367).

The "Security" of the RSA Function

- Factoring pq or calculating d from (e, pq) seems hard.
  - See also p. 363.
- Breaking the last bit of RSA is as hard as breaking the RSA.[a]
- Recommended RSA key sizes:
  - 1024 bits up to 2010.
  - 2048 bits up to 2030.
  - 3072 bits up to 2031 and beyond.

---

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

The "Security" of the RSA Function (concluded)

- Recall that problem A is "harder than" problem B if solving A results in solving B.
  - Factorization is "harder than" breaking the RSA.
  - Calculating Euler's phi function is "harder than" breaking the RSA.
  - Factorization is "harder than" calculating Euler's phi function (see Lemma 49 on p. 359).
- Factorization cannot be NP-hard unless NP = coNP.[a]
- So breaking the RSA is unlikely to imply P = NP.

---

[a] Brassard (1979).

The Secret-Key Agreement Problem

- Exchanging messages securely using a private-key cryptosystem requires Alice and Bob possessing the same key (p. 503).
- 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:

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

- 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.

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).

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

$$E(e, D(d, x)) = D(d, E(e, x)). \tag{7}$$

  – As $(x^d)^e = (x^e)^d$, the RSA system satisfies it.

  – Every cryptosystem guarantees $D(d, E(e, x)) = x$.

[a] Diffie and 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. (7).
- 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 decrypt a ciphertext of hers.

Mental Poker[a]

- Suppose 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.
- Assume Alice and Bob will not deviate from the protocol.

[a]Shamir, Rivest, and 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 $(x^{e_{\text{Alice}}})^{d_{\text{Alice}}} = x \bmod p$ and $(x^{e_{\text{Bob}}})^{d_{\text{Bob}}} = x \bmod p$.
- 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;

Probabilistic Encryption[a]

- 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!
- 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 and Micali (1982).

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.
- 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.
- A sequence of residues and nonresidues are sent.
- Knowing the factorization of n, Bob can efficiently test quadratic residuacity and thus read the message.

A Useful Lemma

Lemma 73 *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.
  - 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 Protocol for Alice

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

The Protocol for Bob

for $i = 1, 2, \ldots, k$ **do**
    Receive r;
    if $(r \,|\, p) = 1$ and $(r \,|\, q) = 1$ **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.
- This scheme is both polynomially secure and **semantically secure**.

What Is a Proof?

- A proof convinces a party of a certain claim.
  - "Is $x^n + y^n \neq z^n$ for all $x, y, z \in \mathbb{Z}^+$ and $n > 2$?"
  - "Is graph G Hamiltonian?"
  - "Is $x^p = x \bmod p$ for prime p and $p \not| x$?"
- 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.

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 very much like the Turing test.[a]

[a]Turing (1950).

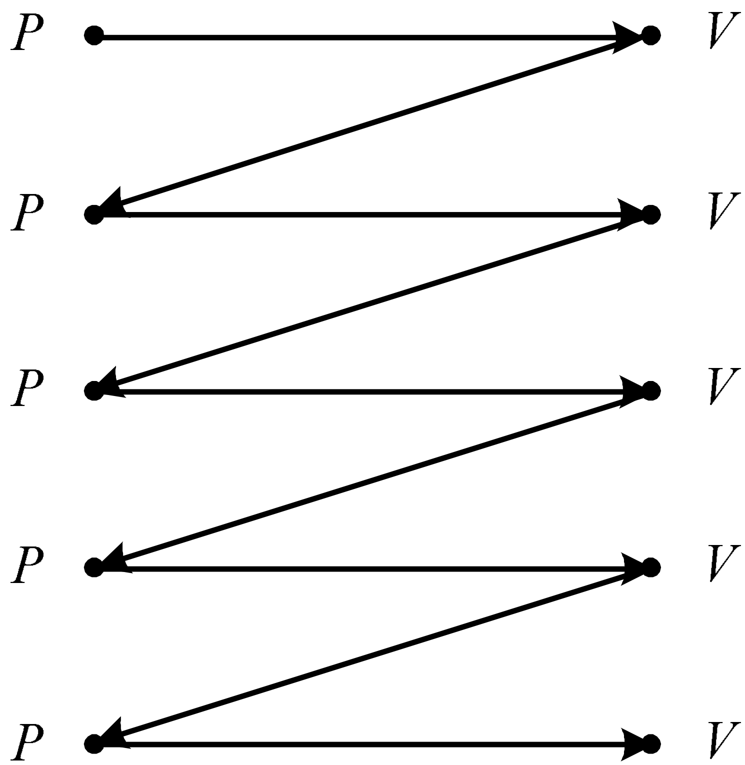
Interactive Proof Systems

- 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 based on the knowledge he acquired in the proof process whether the claim is true or false.
- The verifier must be a probabilistic polynomial-time algorithm.
- The prover runs an exponential-time algorithm.
  - If the prover is not more powerful than the verifier, no interaction is needed.

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 accepts 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, and Rackoff (1985).
[b] Goldreich, Mansour, and Sipser (1987).
[c] Goldwasser and Sipser (1989).

The Relations of IP with Other Classes

- NP $\subseteq$ IP.
  - IP becomes NP when the verifier is deterministic.
- BPP $\subseteq$ IP.
  - IP becomes BPP when the verifier ignores the prover's messages.
- IP actually coincides with PSPACE (see the textbook for a proof).[a]

[a] Shamir (1990).