Lengths of Boolean Formulas for the Threshold Function[a]

- Define the boolean function $T_k(x_1, \ldots, x_n)$ to be 1 if at least k of the x_i's are 1s, and 0 otherwise.
- Trivially, a formula of size $O(\binom{n}{k})$ exists.
  - Formula
$$T_3(x_1, x_2, \ldots, x_n) = \bigvee_{1 \leq i < j < k \leq n} (x_i \wedge x_j \wedge x_k)$$
    has size $\binom{n}{3} = \Theta(n^3)$.
- Surprisingly, for any k, there exists a constant c_k such that $T_k(x_1, \ldots, x_n)$ has formula size at most $c_k n \log_2 n$.
- The construction is again probabilistic, not constructive.

[a] Nechiporuk (1964)?

Lengths of Boolean Formulas for the Threshold Function (continued)

- We will verify the $k = 3$ case below.
- Suppose we construct the formula of the form

$$F = F_1 \vee \cdots \vee F_r.$$

- Each F_i takes the form:

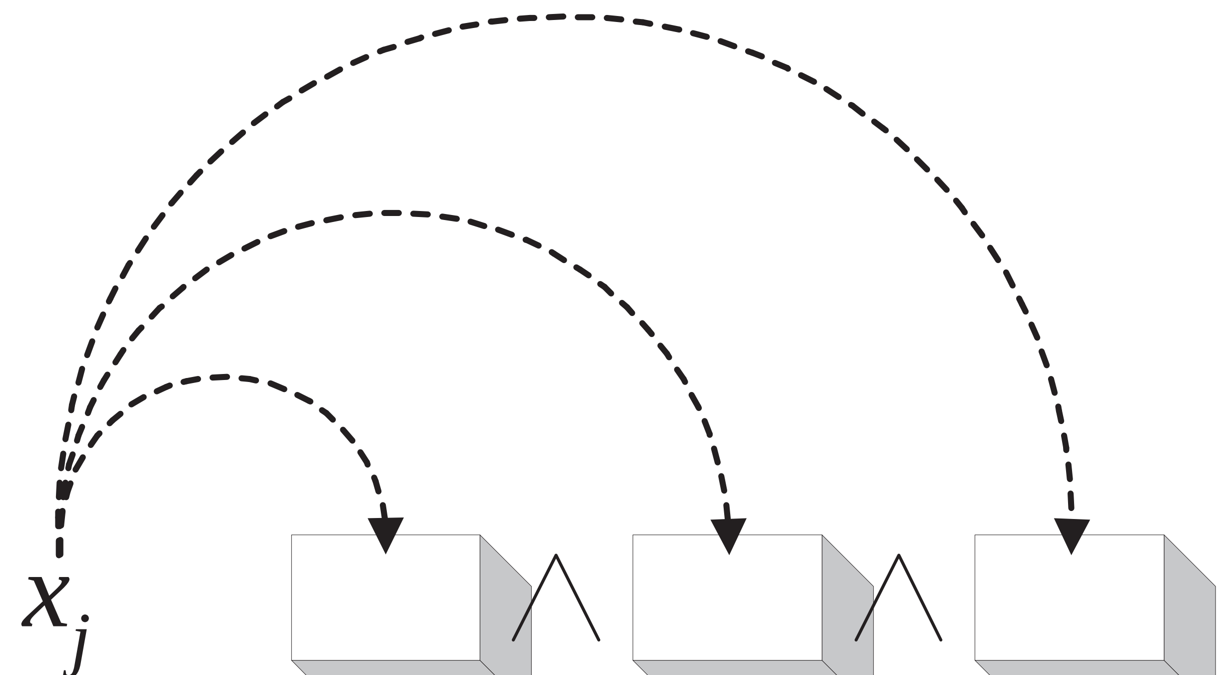
$$F_i = \overbrace{(\bigvee \cdots) \wedge (\bigvee \cdots) \wedge (\bigvee \cdots)}^{3}.$$

  – By the distribution law,

$$\begin{aligned} & (a_1 \vee a_2 \vee \cdots) \wedge (b_1 \vee b_2 \vee \cdots) \wedge (c_1 \vee c_2 \vee \cdots) \\ = \quad & (a_1 \wedge b_1 \wedge c_1) \vee (a_1 \wedge b_1 \wedge c_2) \vee \cdots . \end{aligned}$$

Lengths of Boolean Formulas for the Threshold Function (continued)

- Each x_j is placed into one of the pairs of parentheses at random.
  - E.g., $F_i = (x_1 \lor x_3 \lor x_5) \land (x_2 \lor x_4) \land (x_6 \lor x_7)$.
- So F_i has exactly n variables.
- The process is repeated for each F_i.


x_j
$\wedge$
$\wedge$

Lengths of Boolean Formulas for the Threshold Function (continued)

- Clearly, all the monomials of F are of the form $x_a \wedge x_b \wedge x_c$ for *distinct* a, b, c.
  - For example, F_i may look like

$$\begin{aligned} & (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_4) \wedge (x_6 \vee x_7) \\ = \; & (x_1 \wedge x_2 \wedge x_6) \vee (x_1 \wedge x_2 \wedge x_7) \\ & \vee \cdots \vee (x_5 \wedge x_4 \wedge x_7). \end{aligned}$$

- We know T_3 has $\binom{n}{3}$ monomials.
- We shall show, if r is large enough, all $\binom{n}{3}$ monomials will appear with high probability.

Lengths of Boolean Formulas for the Threshold Function (continued)

- The probability that any given monomial $x_a \wedge x_b \wedge x_c$ appears in a given F_i is the probability that x_a, x_b, x_c are thrown into *distinct* pairs of parentheses.
- The probability is hence equal to $(2/3)(1/3) = 2/9$.
- The probability that $x_a \wedge x_b \wedge x_c$ is not a monomial of F_i's is $(7/9)^r$.
- Therefore, the probability that at least one of the $\binom{n}{3} \leq n^3$ monomials is missing from all the F_i's is $\leq n^3 (7/9)^r$.

Lengths of Boolean Formulas for the Threshold Function (concluded)

- This probability is less than one when $n^3(7/9)^r < 1$.
- When this happens, F includes all $\binom{n}{3}$ monomials, and F has size $< rn$.
- In particular, with $r = -\log_{7/9} 2n^3$, the probability that $F \neq T_3$ is at most $1/2$.
- In other words, the probability of that $F = T_3$ is at least $1/2$.
- Hence a formula of size $O(n \log n)$ exists.

Finding Short Formulas for the Threshold Function

- Our analysis implies an expected polynomial-time randomized algorithm to find such a formula (for T_3).
- Generate F randomly as described.
- In $O(\binom{n}{3}) = O(n^3)$ time, evaluate F with every n-bit truth assignment with three 1's and check if $F = 1$.
- In $O(\binom{n}{2}) = O(n^2)$ time, evaluate F with every n-bit truth assignment with two 1's and check if $F = 0$.
- In $O(n)$ time, evaluate F with every n-bit truth assignment with one 1 and check if $F = 0$.
- Check if $F = 0$ with the all-0 truth assignment.

Finding Short Formulas for the Threshold Function (concluded)

- If F passes all the tests, return F.
  - No need to check if $F = 1$ when the truth assignment contains more than three 1's because F is monotone.[a]
- Otherwise, repeat the experiment.
- Clearly, the expected running time to find a valid formula is proportional to

$$n^3 + (1/2)\, n^3 + (1/2)^2\, n^3 + \cdots = O(n^3).$$

[a] Thanks to a lively class discussion on December 8, 2009.

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

4: Bob receives y;

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

---

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

Analysis

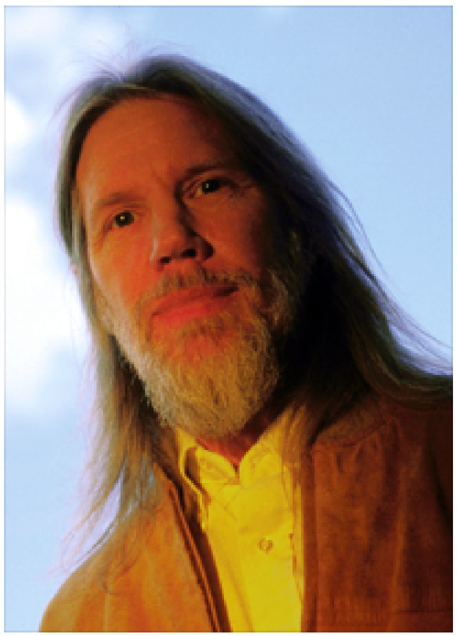
- The one-time pad uses $e = d = r$.
- This is said to be a **private-key cryptosystem**.
- Knowing x and knowing r are equivalent.
- Because r is random and private, the one-time pad achieves perfect secrecy (see also p. 567).
- The random bit string must be new for each round of communication.
  - **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).

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 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 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 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. 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]

[a] Due to Gauss.

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 51 (p. 404),

$$\phi(pq) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = pq - p - q + 1. \quad (8)$$

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

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

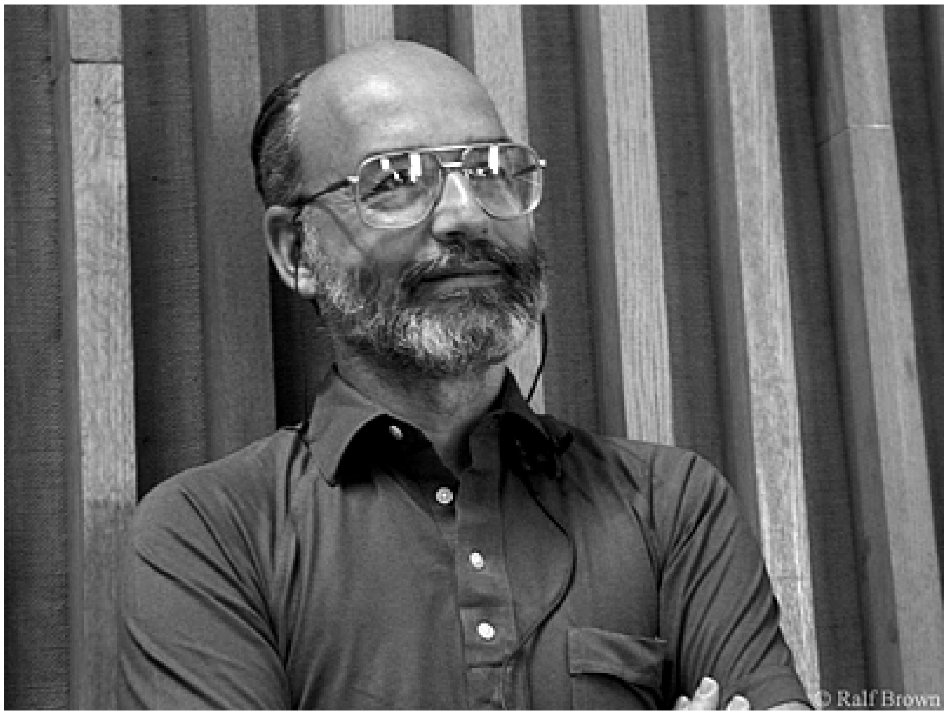
which can be found by the Euclidean algorithm.

Adi Shamir, Ron Rivest, and Leonard Adleman

Ron Rivest[a] (1947–)

[a]Turing Award (2002).

Adi Shamir[a] (1952–)



[a]Turing Award (2002).

Leonard Adleman[a] (1945–)

[a]Turing Award (2002).

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. (8) (p. 578).
  - 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. 412).

The "Security" of the RSA Function

- Factoring pq or calculating d from (e, pq) seems hard.
  - See also p. 408.
- 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, and Schnorr (1988).
[b] RSA (2003).

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 51 on p. 404).
  - So factorization is harder than calculating Euler's phi function, which is harder than breaking the RSA.
- 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. 569).
- 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

$$E(e, D(d, x)) = D(d, E(e, x)). \tag{9}$$

  - 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

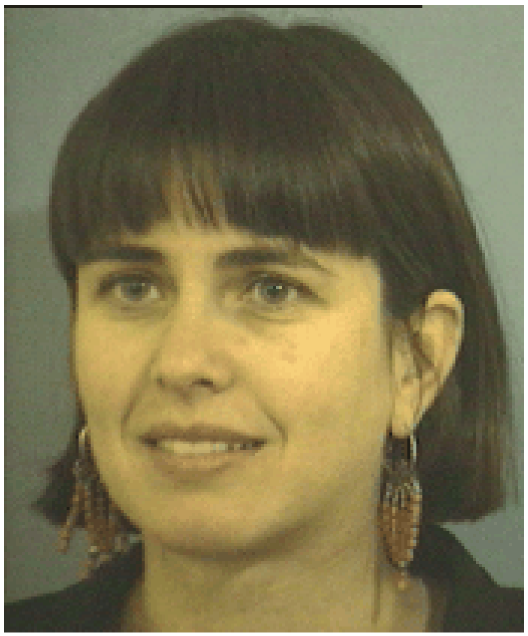
- Alice signs x as
$$(x, D(d_{\text{Alice}}, x)).$$
- Bob receives (x, y) and verifies the signature by checking
$$E(e_{\text{Alice}}, y) = E(e_{\text{Alice}}, D(d_{\text{Alice}}, x)) = x$$
based on Eq. (9).
- The claim of authenticity is founded on the difficulty of inverting E_{Alice} without knowing the key d_{Alice}.
- Warning: If Alice signs anything presented to her, she might inadvertently decrypt a ciphertext of hers.

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 and Micali (1982).

Shafi Goldwasser (1958–)



Silvio Micali (1954–)

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.
- 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 75 *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 Jacobi Symbol and Quadratic Residuacity Test

- The Legendre symbol can be used as a test for quadratic residuacity by Lemma 63 (p. 482).
- Lemma 75 (p. 596) 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 $(y \,|\, pq) = 1$ may be a quadratic nonresidue modulo n when

$$(y \,|\, p) = (y \,|\, q) = -1,$$

because $(y \,|\, pq) = (y \,|\, p)(y \,|\, q)$.

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, ..., 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.
  - "$x^n + y^n \neq z^n$ 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 very much like the Turing test.[a]

[a] Turing (1950).

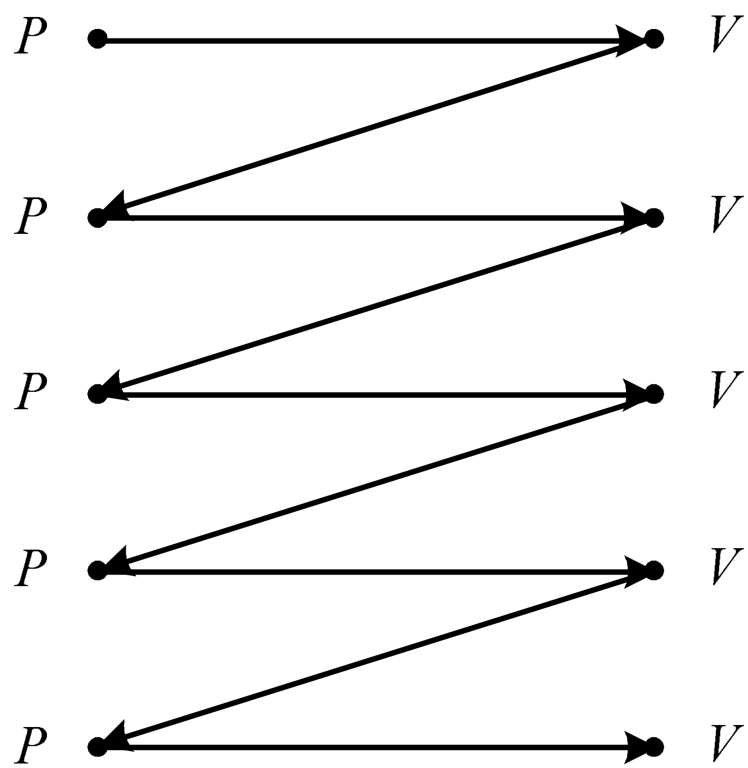
Interactive Proof Systems

- 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 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.[a]

[a]Shamir (1990).

Graph Isomorphism

- $V_1 = V_2 = \{1, 2, \ldots, n\}$.
- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a permutation π on $\{1, 2, \ldots, n\}$ so that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.
- The task is to answer if $G_1 \cong G_2$.
- No known polynomial-time algorithms.
- The problem is in NP (hence IP).
- It is not likely to be NP-complete.[a]

[a]Schöning (1987).

GRAPH NONISOMORPHISM

- $V_1 = V_2 = \{1, 2, \ldots, n\}$.
- Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **nonisomorphic** if there exist no permutations π on $\{1, 2, \ldots, n\}$ so that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.
- The task is to answer if $G_1 \not\cong G_2$.
- Again, no known polynomial-time algorithms.
  - It is in coNP, but how about NP or BPP?
  - It is not likely to be coNP-complete.
- Surprisingly, GRAPH NONISOMORPHISM $\in$ IP.[a]

[a] Goldreich, Micali, and Wigderson (1986).

A 2-Round Algorithm

Victor selects a random $i \in \{1, 2\}$;
Victor selects a random permutation π on $\{1, 2, \ldots, n\}$;
Victor applies π on graph G_i to obtain graph H;
Victor sends (G_1, H) to Peggy;
if $G_1 \cong H$ **then**
    Peggy sends $j = 1$ to Victor;
else
    Peggy sends $j = 2$ to Victor;
end if
if $j = i$ **then**
    Victor accepts;
else
    Victor rejects;
end if

Analysis

- Victor runs in probabilistic polynomial time.
- Suppose $G_1 \ncong G_2$.
  - Peggy is able to tell which G_i is isomorphic to H.
  - So Victor always accepts.
- Suppose $G_1 \cong G_2$.
  - No matter which i is picked by Victor, Peggy or any prover sees 2 identical graphs.
  - Peggy or any prover with exponential power has only probability one half of guessing i correctly.
  - So Victor erroneously accepts with probability 1/2.
- Repeat the algorithm to obtain the desired probabilities.