

CS549:
Cryptography and Network
Security

© by Xiang-Yang Li

Department of Computer Science,
IIT

Notice©

This lecture note (Cryptography and Network Security) is prepared by Xiang-Yang Li. This lecture note has benefited from numerous textbooks and online materials. Especially the "Cryptography and Network Security" 2nd edition by William Stallings and the "Cryptography: Theory and Practice" by Douglas Stinson.

You may not modify, publish, or sell, reproduce, create derivative works from, distribute, perform, display, or in any way exploit any of the content, in whole or in part, except as otherwise expressly permitted by the author.

The author has used his best efforts in preparing this lecture note. The author makes no warranty of any kind, expressed or implied, with regard to the programs, protocols contained in this lecture note. The author shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these.

Cryptography and Network Security

Secret Sharing

Xiang-Yang Li

Threshold Scheme

- A (t, w) -threshold scheme
 - Sharing key K among a set of w users
 - Any t users can recover the key
 - Any $t-1$ users can not do so
- Schemes
 - Shamir's scheme
 - Geometric techniques
 - Matroid theory

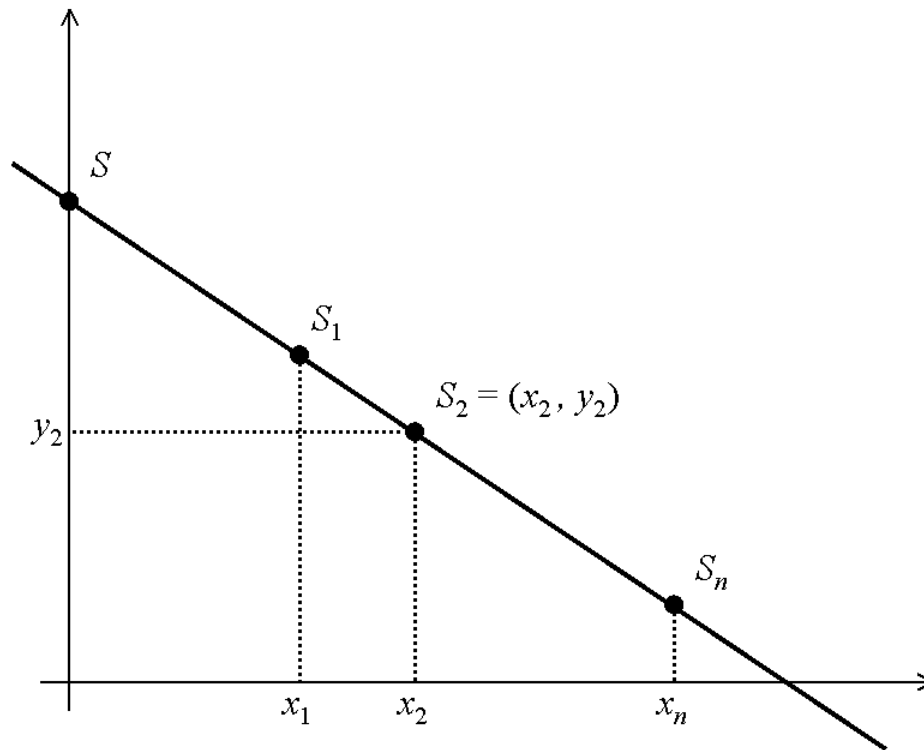
Information Theory

- The secret sharing is as large as the original secret
 - This result is based in information theory, but can be understood intuitively. Given $t-1$ shares, no information whatsoever can be determined about the secret. Thus, the final share must contain as much information as the secret itself.
- All secret sharing schemes use random bits.
 - To distribute a one-bit secret among threshold t people, $t-1$ random bits are necessary. The final share contains as much information as the secret, but the other $t-1$ shares still provide relevant information individually. This information cannot be the secret, so it must be random.

Shamir's Scheme

- Initialization phase
 - Dealer chooses a large prime number p
 - Dealer chooses w distinct x_i from Z_p
 - Gives value x_i to person p_i
- Share distribution of key k from Z_p
 - Dealer choose $t-1$ random number a_j
 - Dealer computes $y_i=f(x_i)$
 - Here $f(x)=k+\sum a_j x^j \text{ mod } p$
 - Dealer gives share y_i to person p_i

Geometry View



Simple (t,t) Sharing

➤ Procedure

- D secretly chooses $t-1$ random elements y_i from Z_n
- D computes
 - Value $y_t = K - \sum y_j \text{ mod } n$
- D distributes y_i to person p_i for all i

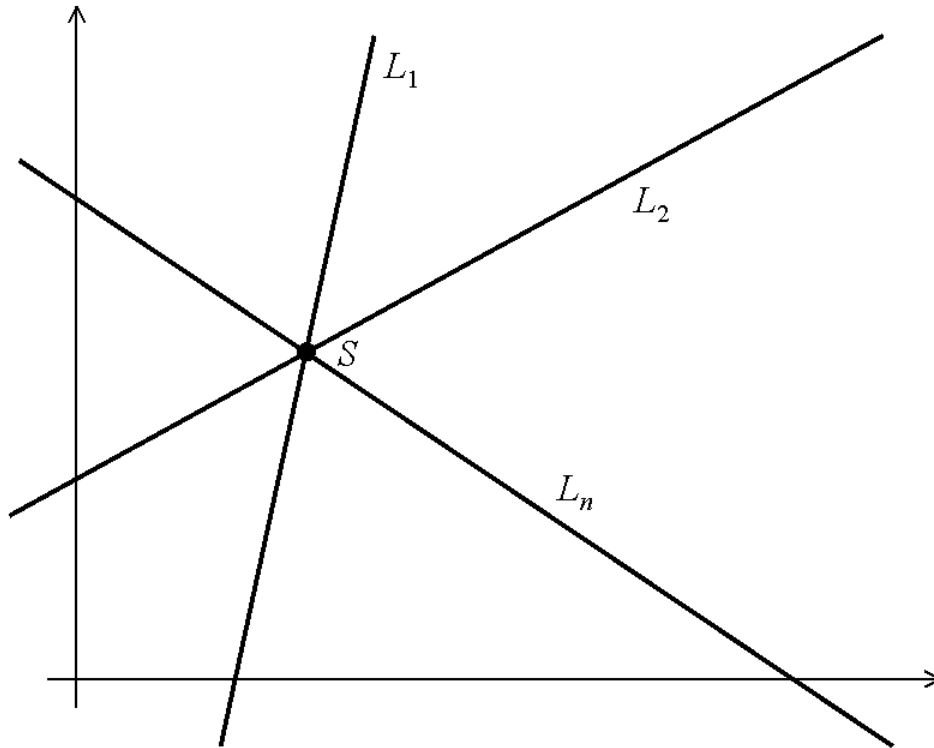
➤ It is secure and easy

- Number n can be any number
- Easy to recover the key
- Only t persons together can do so, assume y_i random

Blakley's Scheme

- Secret is a point in an t -dimensional space
- Dealer gives each user a hyper-plane passing the secret point
- Any t users can recover the common point

Geometry View



Avoid Cheating

- Two major distinct weaknesses
 - Bogus values are undetectable.
 - Participants need not reveal their true share.
- Even if a bogus value was detected, it would not necessarily give any information about the true value
- One participant did not reveal its true value after get the true values from other one

Verifiable Secret Sharing

- How can I know whether a share is correct or not?
 - Note that the correctness of a share can be verified using t other shares.
 - However, we can't ask other parties to reveal t shares.
- So each share should have a commitment which is public.
 - The correctness of shares can be verified using commitments.
 - This is called Verifiable Secret Sharing (VSS).

Ben-Or/Rabin Solution

- Using Checking Vectors
- For any two participants A and B
 - Dealer gives A (S_A, Y_{AB})
 - Dealer gives B (B_{AB}, C_{AB})
 - Here $C_{AB} = B_{AB} Y_{AB} + S_A \pmod{p}$
 - S_A is the secret share of A
 - A and B keep their values secret
 - B can use (B_{AB}, C_{AB}) to verify the value (S_A, Y_{AB}) of A

Avoid Cheating

- Participant B can send A bogus value after receive A's value
- Solution: bit transfer
 - Dealer gives A (S_{Ai}, Y_{ABi})
 - Dealer gives B (B_{ABi}, C_{ABi})
 - Here $C_{ABi} = B_{ABi} Y_{ABi} + S_{Ai} \pmod{p}$
 - S_{Ai} is the i th bit of the secret share of A

Cont.

➤ Protocol

- Participant A gives its value (S_{Ai}, Y_{ABi}) to B
- B verifies: $C_{ABi} = B_{ABi} Y_{ABi} + S_{Ai} \pmod p$
- B then sends its value (S_{Bi}, Y_{BAi}) to A
- A verifies: $C_{BAi} = B_{BAi} Y_{BAi} + S_{Bi} \pmod p$
- The protocol terminates whenever
 - One side detects cheating, or
 - All values transferred

Chinese Remainder Theorem

- Given a number $m < n$, and $n = n_1 n_2 \dots n_k$
 - Numbers n_i and n_j are coprimes
 - Let $a_i = m \bmod n_i$
 - Number n is public
 - Dealer delivers a_i and n_i to the i th participant
 - Then all k users can recover the number m
- Why it is not a good secret sharing scheme?
 - Is it computationally for any $k-1$ users to recover the key if n is large?

Recover method

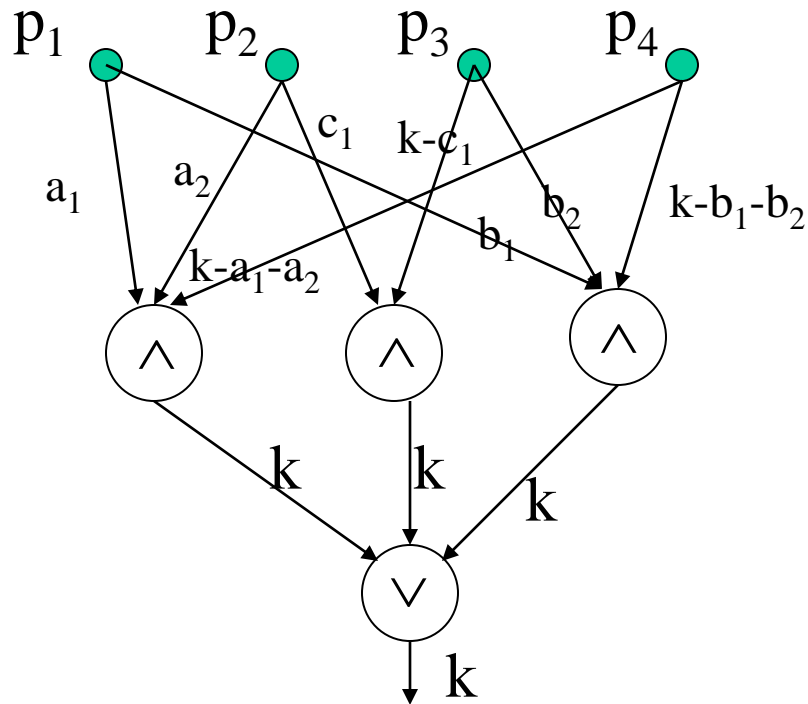
- Each user pre-computes
 - $N_i = n/n_i$
 - Inverse of N_i : $y_i = N_i \text{ mod } n_i$
 - Compute the product $s_i = a_i N_i y_i \text{ mod } n$
- Recover the secret m
 - Each user submits s_i
 - Computes $s_1 + s_2 + \dots + s_k \text{ mod } n$

Access Structure

- Threshold scheme allows any t users to recover key!
- Access structure allows some subsets to recover the key!
 - Example: $\{\{p_1, p_2, p_4\}, \{p_1, p_3, p_4\}, \{p_2, p_3\}\}$ among p_1, p_2, p_3, p_4, p_5 able to recover the key
 - Assume the accessing subset is minimized
 - No subset of any accessing subset is able to recover

Monotone Circuit

- Assign sharing for each accessing subset



Cont.

➤ Distribution

- (a_1, b_1) to p_1
- (a_2, c_1) to p_2
- $(k - c_1, b_2)$ to p_3
- $(k - a_1 - a_2, k - b_1 - b_2)$ to p_4

➤ The sharer needs know

- The circuit used by dealer
- Which shares corresponding to which wires
 - The shared value is secret

Visual Cryptography:
Secret Sharing without a
Computer

By Ricardo Martin

Visual Secret Sharing

- There is a secret picture to be shared among n participants.
 - The picture is divided into n transparencies (shares) such that
 - if any m transparencies are placed together, the picture becomes visible
 - but if fewer than m transparencies are placed together, nothing can be seen.

Visual Secret Sharing

- Such a scheme is constructed by viewing the secret picture as a set of black and white pixels and handling each pixel separately.
 - The schemes are perfectly secure and easily implemented without any cryptographic computation.
- A further improvement allows each transparency (share) to be an innocent picture
 - For example, a picture of a landscape or a picture of a building
 - thus concealing the fact of secret sharing

Secret Sharing

➤ (2,2)-Secret Sharing: Any share by itself does not provide any information, but together they reveal the secret.

• An example:

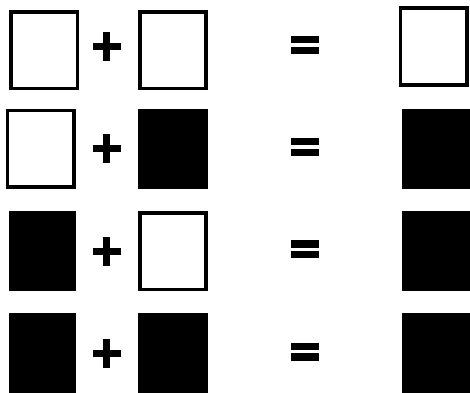
One-time pad: the secret binary string

$\mathbf{k} = k_1 k_2 k_3 \dots k_n$ can be shared as

$\{\mathbf{x} = x_1 x_2 \dots x_n ; \mathbf{y} = y_1 y_2 \dots y_n \}$, where x_i is random and $y_i = k_i \text{ XOR } x_i$

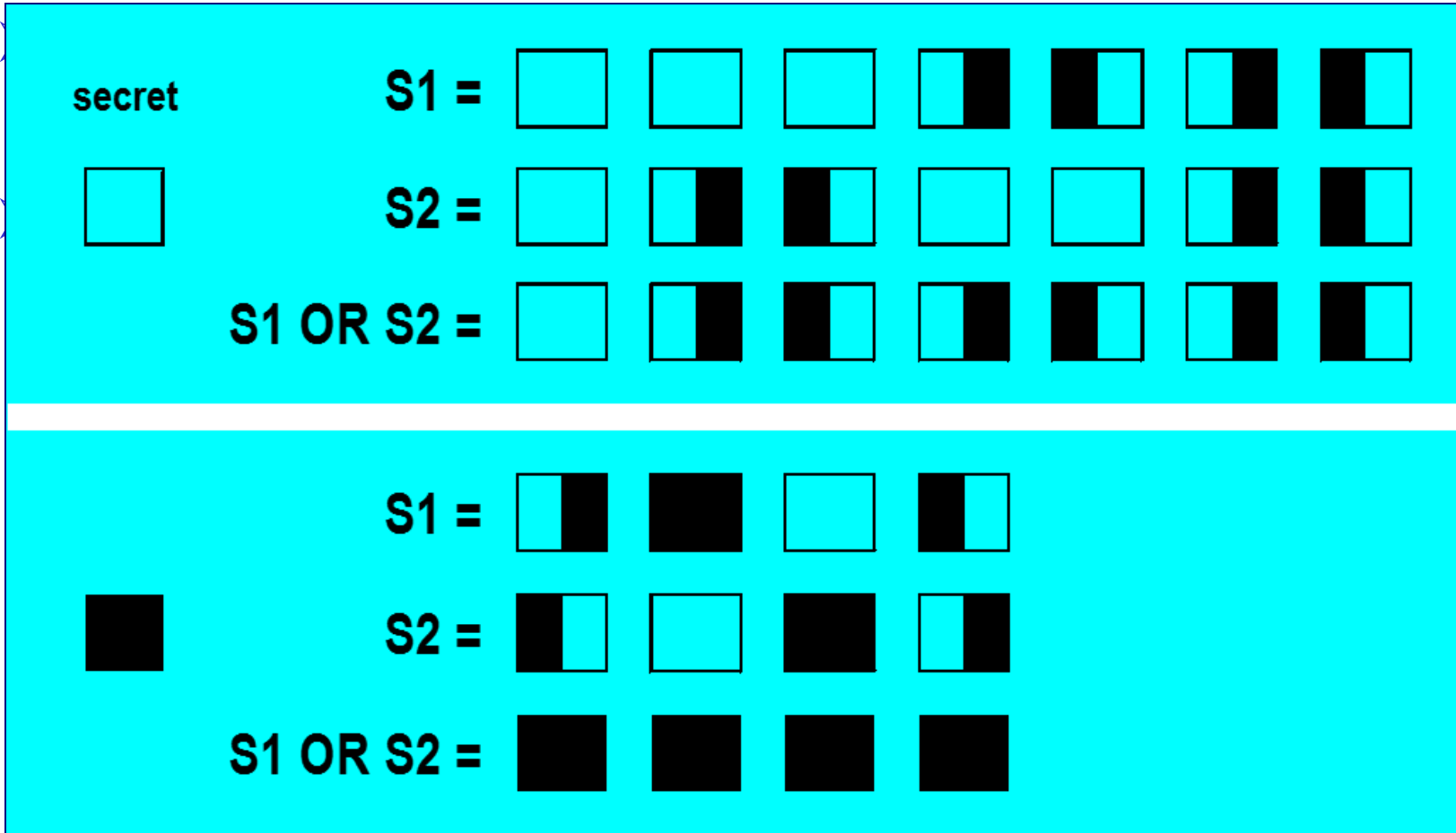
Visual Secret Sharing

- Shares are **images** printed on transparencies. The secret is reconstructed **by the eye not a computer.**
- Decryption by superimposing the proper transparencies
 - bits of the shares are combined as x_i **OR** y_i .

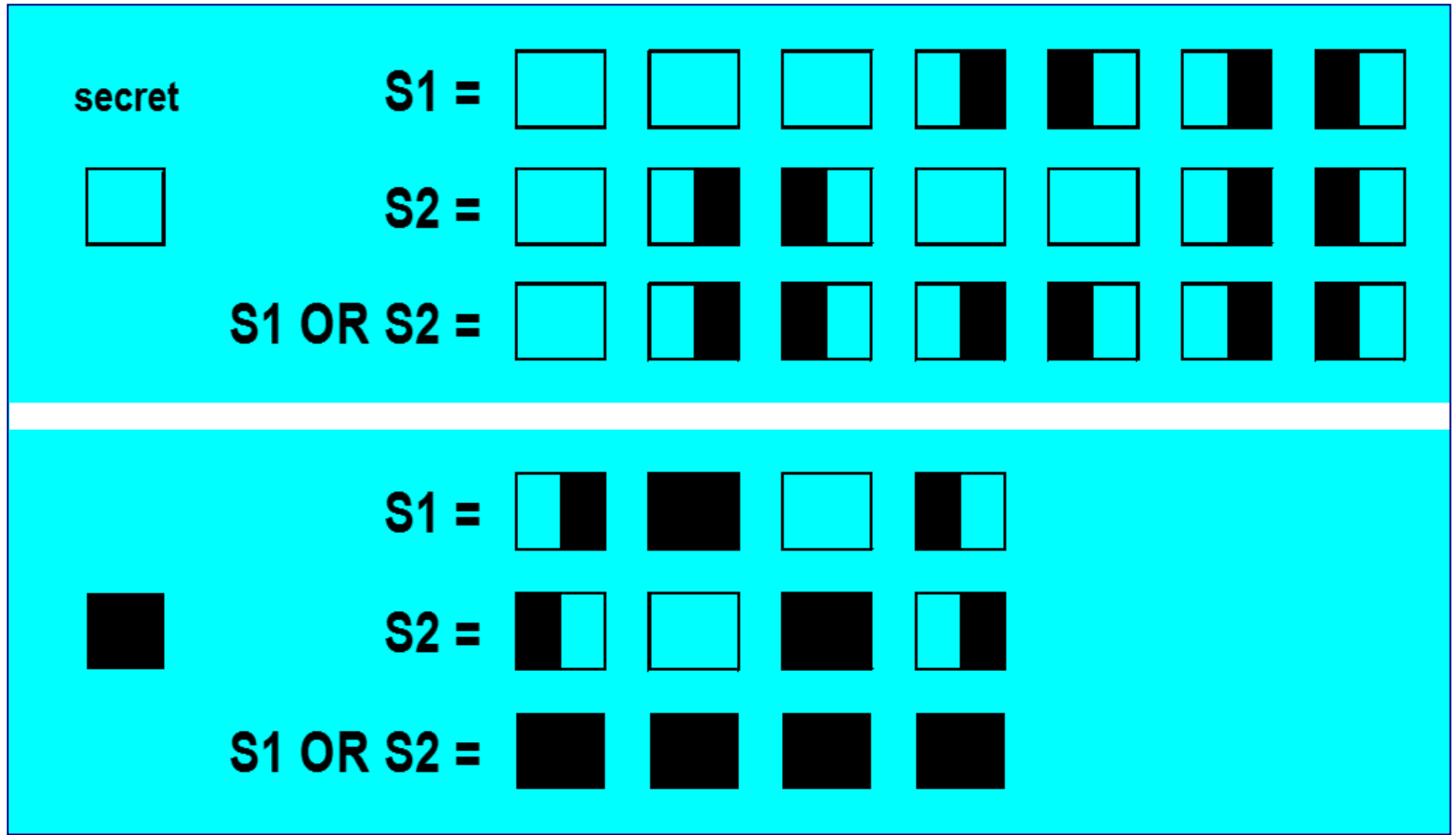


Since $(\{0,1\}, \text{OR})$ is not a group we need to introduce redundancy.

An example

















But is this secure?



Now it passes Shannon test: $\Pr(k/s_i) = \Pr(k)$ as
 $\text{Prob}(s_i = '10'/0) = \text{Prob}(s_i = '10'/1) = .5$ and $\text{Prob}(s_i = '01'/0) = \text{Prob}(s_i = '01'/1) = .5$

Sharing Matrix representation

| Secret | p = .5 | p = .5 | Superposition |
|---|---|---|--|
|  | Share # 1 =   |   |   |
|  | Share # 1 =   |   |   |

$$S_0 = \left\{ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \right\}$$

$$S_1 = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right\}$$

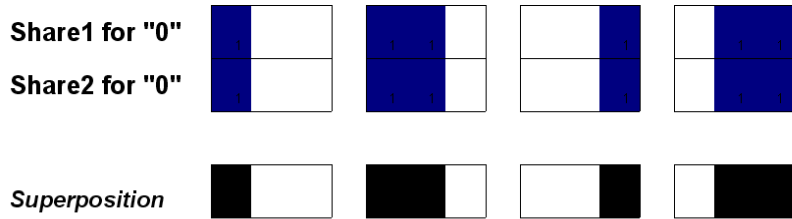
- $S = [S_{ij}]$ a boolean matrix with:
 - a row for each share, a column for each subpixels
 - $S_{ij} = 1$ iff the j^{th} subpixel of the i^{th} share is dark.
 - one set of matrices for “0” and one for “1” (or one for each grey-level in secret image)
 - “normally” each set is the column permutations of base matrix
- for each pixel, choose a random matrix in the corresponding set (“normally” with equal probabilities)

Properties of Sharing Matrices

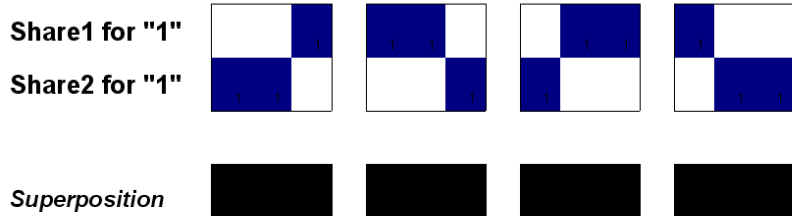
For Contrast: sum of rows for shares in a *decrypting group* should be bigger for darker pixels.

For Secrecy: sums of rows in any *non-decrypting group* should have same probability distribution for the number of 1's in s_0 and in S_1 .

Another 2-of-2 example (m=3)



$$S_0 = \left\{ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right\}$$



$$S_1 = \left\{ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \right\}$$

- Each matrix selected with equal probability (0.25)
 - the set of different column permutations of the first two matrices in each set. each with prob=1/6, would work as well.
- Sum of sum of rows is 1 or 2 in S_0 , while it is 3 in S_1
- Each share has one or two dark subpixels with equal probabilities (0.5) in both sets.

Naor-Shamir, 1994

(k,n) secret sharing: an N -bits secret shared among n participants, using m subpixels per secret bit (n strings of mN), so that any k can decrypt the secret:

Contrast: There are $d < m$ and $0 < \alpha < 1$:

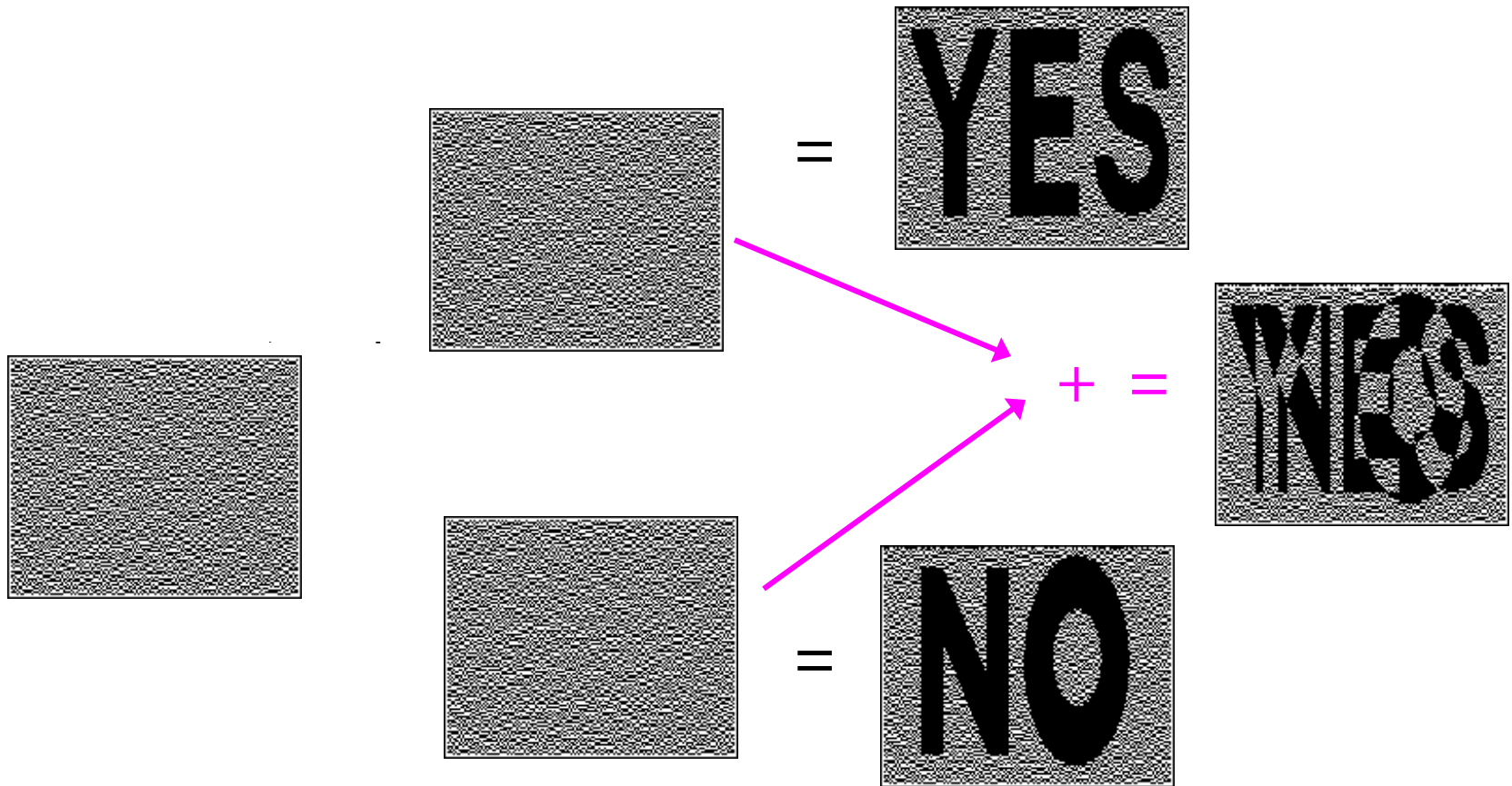
- If $p_i = 1$ at least d of the corresp. m subpixels are dark ("1").
- If $p_i = 0$ no more than $(d - \alpha m)$ of the m subpixels are dark

Security: Any subset of less than k shares does not provide any information about the secret x .

- All shares code "0" and "1" with the same number of dark subpixels in average.

Stefan's construct

One share can decrypt two images...



... but with less than perfect secrecy.

A (2,m) Secret Sharing Scheme

$\left[\begin{array}{l} 100 \dots 0 \\ 100 \dots 0 \\ \dots \\ 100 \dots 0 \end{array} \right]$ [Naor & Shamir] All shares receive 1 dark and (m-1) clear subpixels.

For a '0', all m shares have the same dark (random) subpixels.

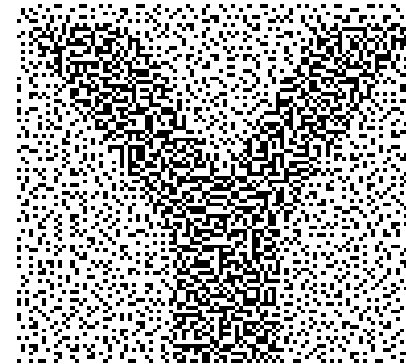
$\left[\begin{array}{l} 100 \dots 0 \\ 010 \dots 0 \\ \dots \\ 000 \dots 1 \end{array} \right]$ For a '1', all m shares have a different dark subpixels.

Thus all shares are indistinguishable, but any two have 1 dark subpixels for "0" and 2 for a "1".

How can we exclude a coalition, say (1,2)?

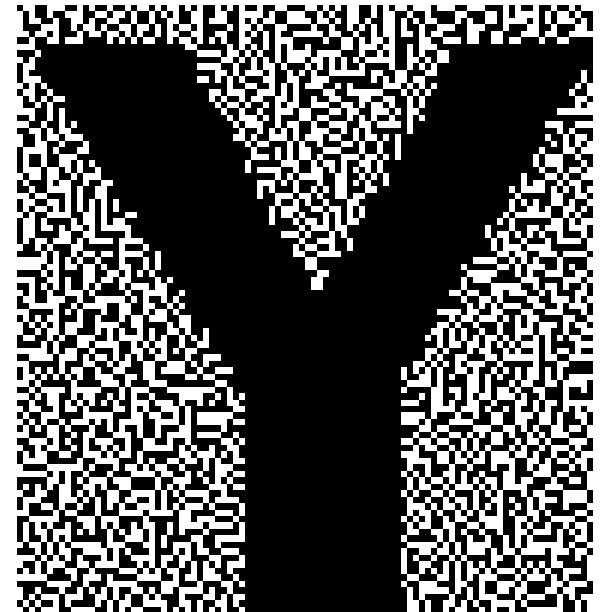
Two (2,6) sharing schemes

Previous scheme ($\alpha=1/4$)



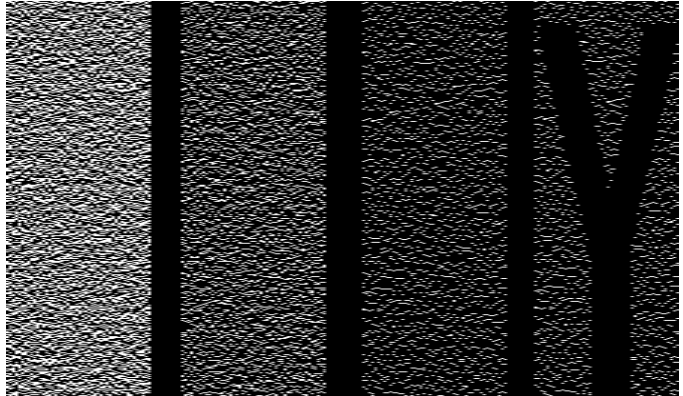
More efficient sharing matrices ($\alpha=1/2$)

$$S_0 = \left\{ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} (+ \text{ perm.}) \right\}$$
$$S_1 = \left\{ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} (+ \text{ perm.}) \right\}$$



A (4,4) Visual Sharing Scheme

$$S_0 = \left\{ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \text{ and permutations} \right\}$$

$$S_1 = \left\{ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \text{ and permutations} \right\}$$


Any subgroup of 3 or less shares have the same number of dark subpixels for 0 (S_0) and for 1 (S_1), but the 4 together have one clear subpixel for 0 and are all dark for 1.

Contrast is low: $\alpha=1/9$

General Results from Naor-Shamir

1. There is a (k, k) scheme with $m=2^{k-1}$, $\alpha=2^{-k+1}$ and $r=(2^{k-1}!)$.

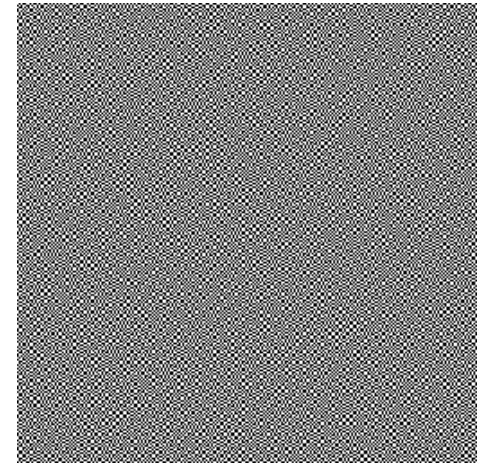
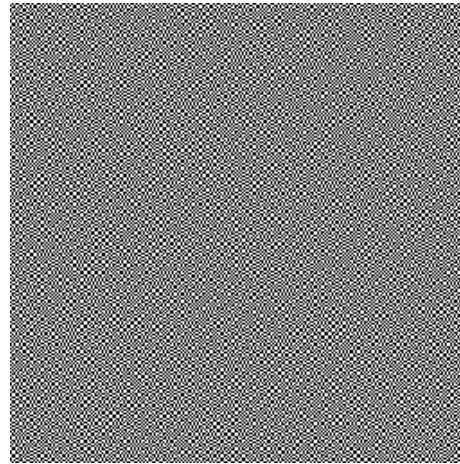
We can construct a $(5,5)$ sharing, with **16** subpixels per secret pixel and **1** pixel contrast, using the permutations of **16** sharing matrices.

2. In any (k, k) scheme, $m \geq 2^{k-1}$ and $\alpha \leq 2^{1-k}$.
3. For any n and k , there is a (k, n) VS scheme with $m = \log n \cdot 2^{O(k \log k)}$, $\alpha = 2^{\Omega(k)}$.

Example 1: Lena B&W



Original



Shares



**Superposition of Shares 1
and 2, perfectly aligned**

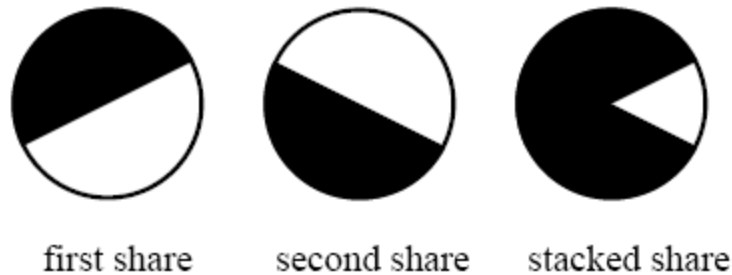
Extensions: Beyond (K,M)

General Share Structures [Ateniense et.al. 1996]:

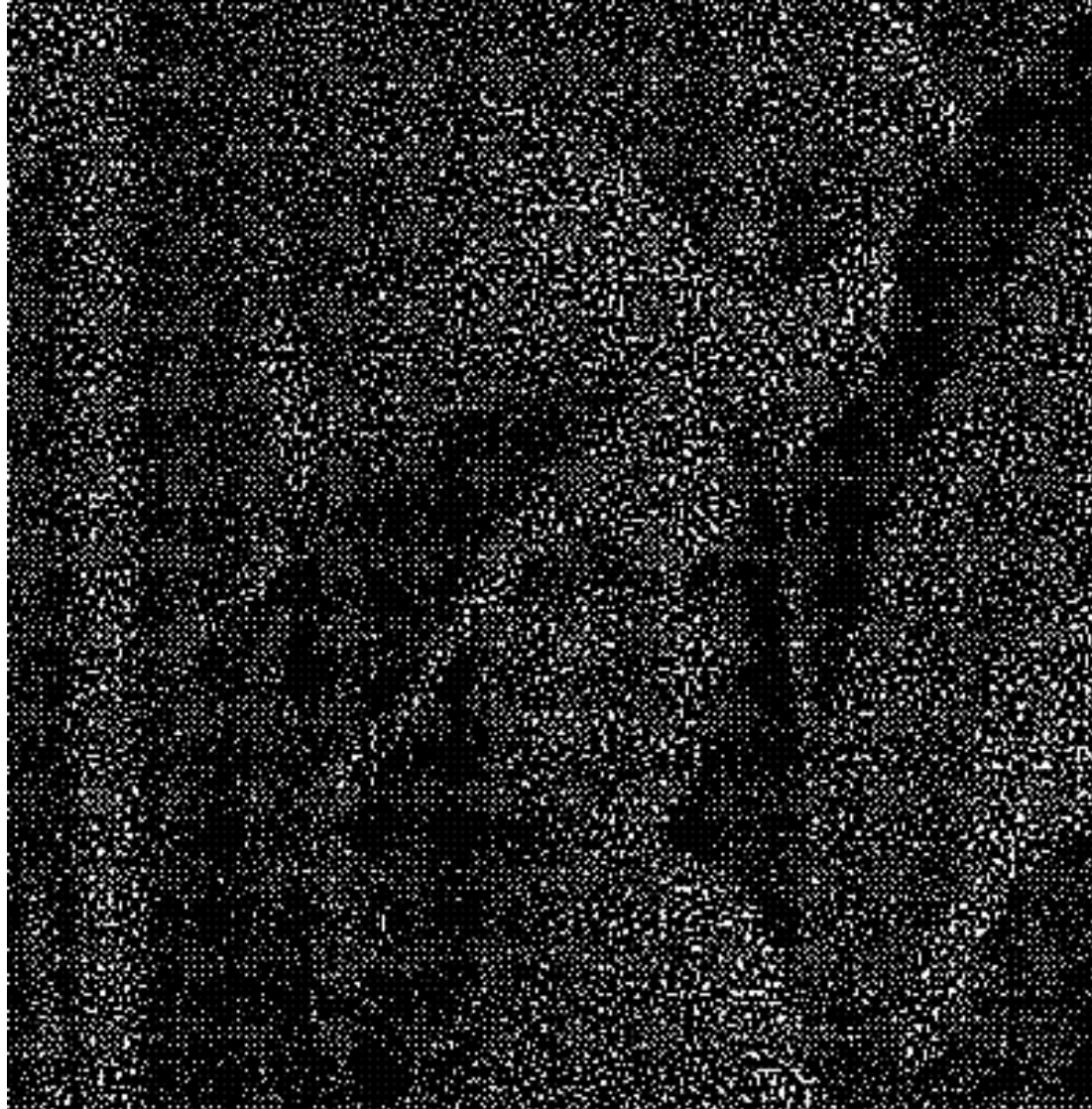
- Define arbitrary sets *Qual* and *Forb* as subsets of participants.
 - Any set in *Qual* can recover the secret by stacking their transparencies
 - Any set in *Forb* has no information on the shared image.
- They show constructions satisfying these requirements, with mild restrictions on the sets.

Extended VSS - Grey Scale

- Naor & Shamir suggested using partially filled circles to represent grey values.
- The actual implementation (vck, transparencies) is less than overwhelming.



Example 2: Lena Grey Scale



Another Grey Scale VSS system

- Use more subpixels to represent grey levels (Nakajima & Yamaguchi).
- Use g sets of sharing matrices (one for each grey levels, $g \geq 2$)

$$S_1 = \left\{ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, (+ \text{ perm.}) \right\}$$

$$S_2 = \left\{ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, (+ \text{ perm.}) \right\}$$

$$S_3 = \left\{ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, (+ \text{ perm.}) \right\}$$



Extended VSS- Multiple Images

[Nakajima and Yamaguchi, Stoleru] Adding more redundancy, shares can be a pre-specified image, instead of random noise.



No Perfect Secrecy for all images (need to adjust ranges of grey levels in cover pictures)

Concluding Thoughts

- **Not just a cute toy.** Proposed applications:
 - paper trail on electronic voting (Chaum).
 - encryption of financial documents (Hawkes)
 - tickets sale?
- Shares can be difficult to align (it helps to have fat pixels, but that reduces quality),
- Contrasts declines rapidly with the number of shares and grey levels.
- Can it be make to work with color?

References

- Moni Naor and Adi Shamir (1994) *Visual Cryptography*, Eurocrypt 94
- G. Ateniense, C. Blundo, A. de Santis and D.R.Stinson (1996) *Visual Cryptography for General Access Structures*.
- N. Nakajima and Y. Yamaguchi (n.d.), *Extended Visual Cryptography for Natural Images*
- D. Stoleru (2005), *Extended Visual Cryptography Schemes*, Dr. Dobb's, 377, October 2005
- D. Stinson (2002), *Visual Cryptography or Seeing is Believing*, pp presentation in pdf.

Interactive proof

Interactive Proof

- *Interactive proof* is a protocol between two parties in which one party, called the *prover*, tries to prove a certain fact to the other party, called the *verifier*
- Often takes the form of a challenge-response protocol

cont

- protocol in which one or more provers try to convince another party, called the verifier, that the prover(s) possess certain true knowledge, such as **the membership of a string x in a given language**, often with the goal of revealing no further details about this knowledge.
 - The prover(s) and verifier are formally defined as *probabilistic Turing machines* with special "interaction tapes" for exchanging messages.

Desired Properties

- **Desired properties of interactive proofs**
 - *Completeness*: The verifier always accepts the proof if the prover knows the fact and both the prover and the verifier follow the protocol.
 - *Soundness*: Verifier always rejects the proof if prover doesnot know the fact, and verifier follows protocol.
 - *Zero knowledge*: The verifier learns nothing about the fact being proved (except that it is correct) from the prover that he could not already learn without the prover. In a zero-knowledge proof, the verifier cannot even later prove the fact to anyone else.

Cont

➤ ZKP:

- if the statement is true, no cheating verifier learns anything other than this fact.
- This is formalized by showing that every cheating verifier has some *simulator* that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

Example: Graph Isomorphism

- Suppose G_1 and G_2 are two graphs known to both you and me. Furthermore, I know they are isomorphic.
 - But you don't know and can't figure it out.
- How can I prove to you that they are isomorphic?
 - We need to **interact** with each other to achieve this goal.

Interactive Proof

- More generally, suppose L is a language and x is in L . (Both of us know x .) How can I prove this fact to you?
- An interactive proof system (P, V) is two (interactive) PPT algorithms P and V such that
 - P and V have a common input x .
 - (P/V may have its own private input with length $\text{poly}(|x|)$)
 - Each algorithm runs interleavingly and sends message to the other.
 - Finally, V outputs “accept” or “reject”.

Completeness and Soundness

- The interactive proof system is **complete** for language L if for all x in L , the output of (P, V) is accept with high probability.
- The interactive proof system is **sound** for language L if for all x not in L , for all P^* , the output of (P^*, V) is reject with high probability.

Example: Proof for Graph Isomorphism

- Consider the example of graph isomorphic map. We can have:
 - P sends the isomorphic map directly to V ;
 - V accepts if and only if it is indeed an isomorphic map.
- This is complete because V accepts with probability 1 when G_1 is isomorphic to G_2 .
- This is sound because V rejects with probability 1 when G_1 is not isomorphic to G_2 .

What Languages Have Interactive Proofs?

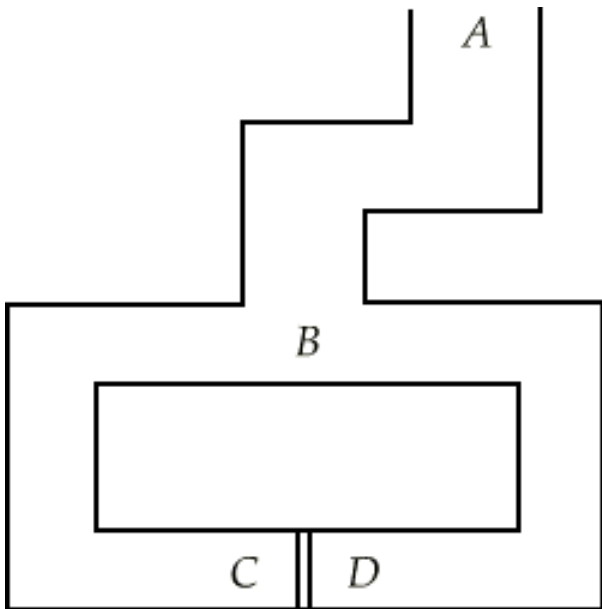
- For every language L in NP, we can construct an interactive proof system for L .
- In fact, Shamir showed that **IP=PSPACE**.
 - Where IP is the set of languages having interactive proofs.
 - PSPACE is the set of languages that can be recognized with polynomial-size space.

Typical Protocol for ZKP

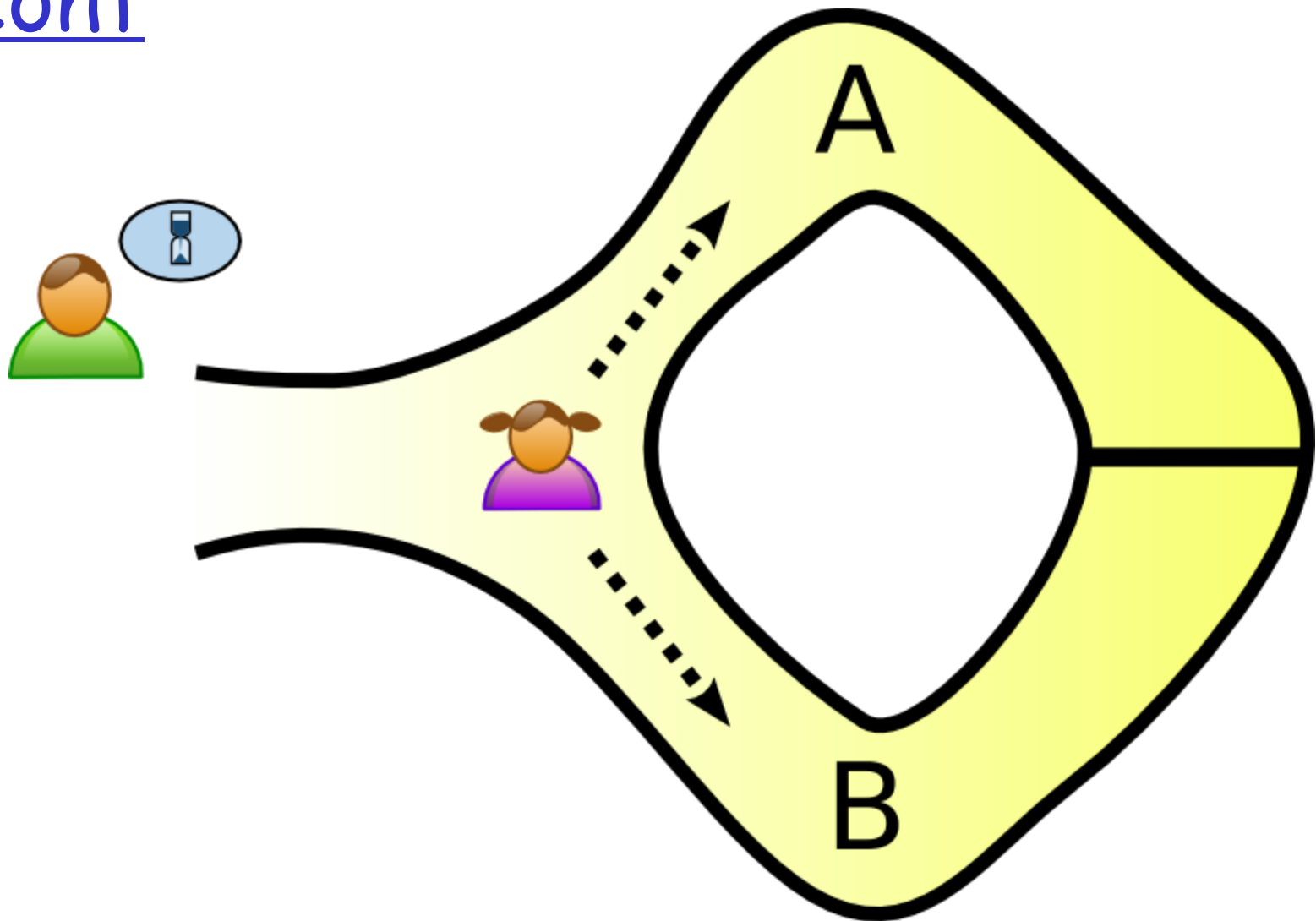
- A typical round in a zero-knowledge proof consists of
 - a "commitment" message from the prover,
 - followed by a challenge from the verifier, and
 - then a response to the challenge from the prover.
 - The protocol may be repeated for many rounds. Based on the prover's responses in all the rounds, the verifier decides whether to accept or reject the proof.

An example

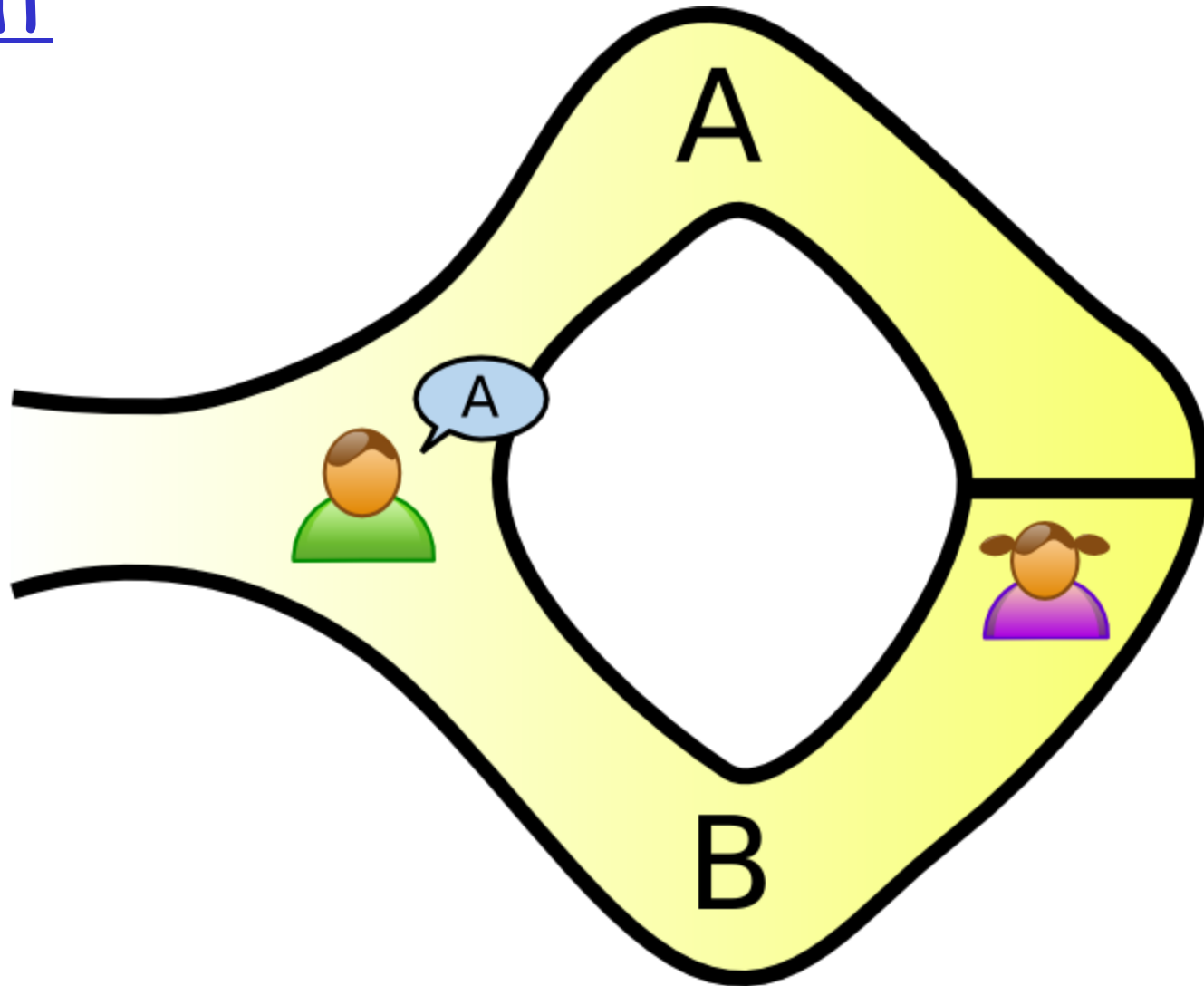
➤ Ali Baba's Cave



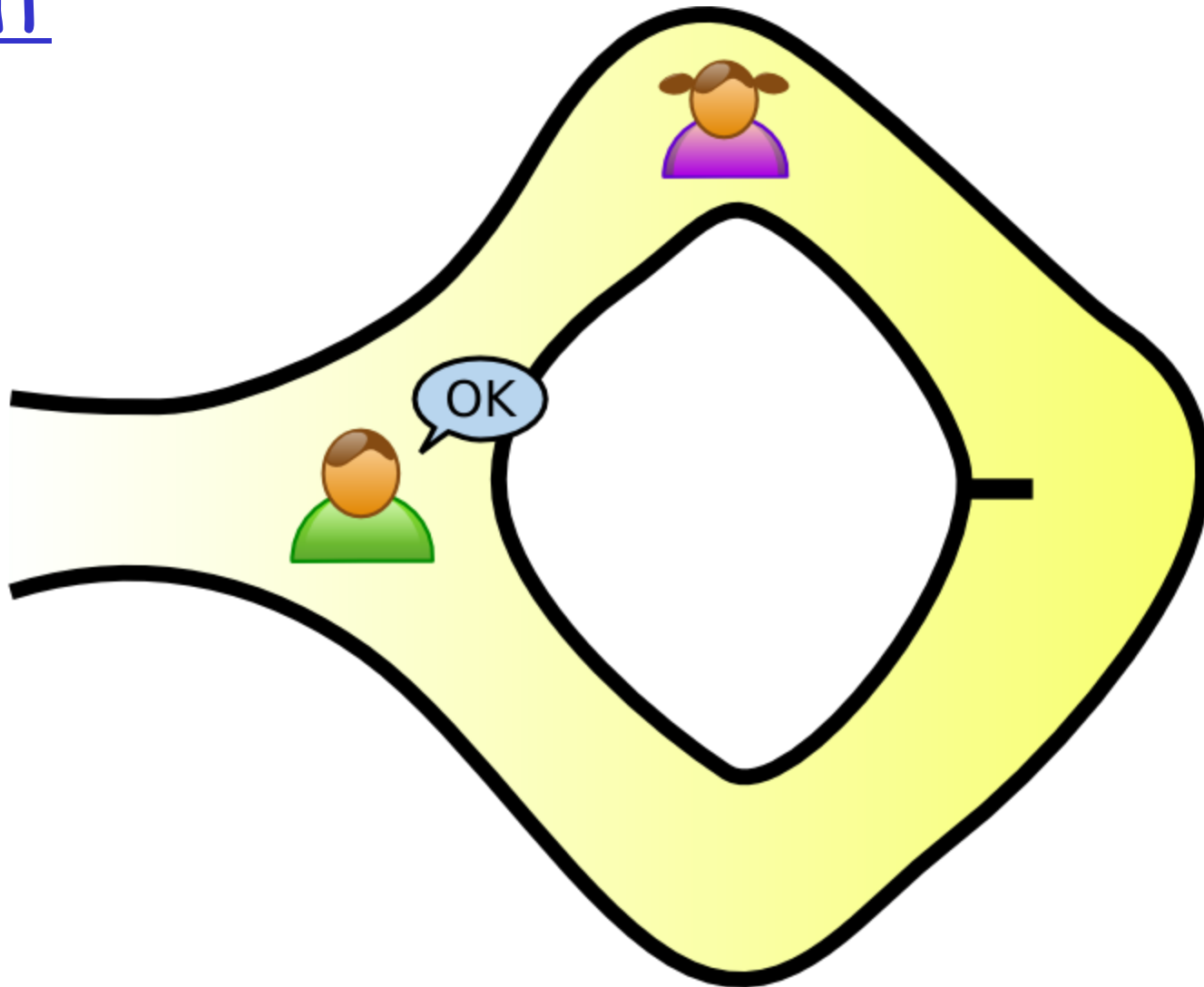
cont



cont



cont



Cont.

- Alice wants to prove to Bob that
 - she knows the secret words to open the portal at CD
 - but does not wish to reveal the secret to Bob.
 - In this scenario, Alice's commitment is to go to C or D.

Proof Protocol

- A typical round in the proof proceeds as follows:
 - Bob goes to A, waits there while Alice goes to C or D.
 - Bob then asks Alice to appear from either the right side or the left side of the tunnel.
 - If Alice does not know the secret words
 - there is only a 50 percent chance that she will come out from the right tunnel.
 - Bob will repeat this round as many times as he desires until he is certain that Alice knows the secret words.
 - No matter how many times that the proof repeats, Bob does not learn the secret words.

Knowledge in Proof

- The example proof for graph isomorphism is simple, complete and sound.
- However, it gives V too much knowledge.
 - The isomorphic map is completely revealed to V .
 - In cryptography, we often want to convince somebody without giving him knowledge. Such a proof is called a **zero-knowledge (ZK) proof**.

Perfect Zero Knowledge

- We say an interactive proof system (P, V) is perfect zero-knowledge (PZK) if for all V^* , there exists an efficient algorithm M such that
 - M outputs “fail” with negligible probability;
 - Conditioned on that M does not output “fail”, for each common input x , the output of V^* (after (P, V^*) is executed on x) is identically distributed to $M(x)$.
 - This is the well-known “simulation paradigm”. Here M is called a “simulator”.

Alternative Definition of PZK

- The **view** of a party consists of its internal coin flips and the messages it receives.
- (P, V) is PZK if and only if for all V^* , there exists an efficient algorithm M such that
 - M outputs “fail” with negligible probability;
 - Conditioned on that M does not output “fail”, for all x , the view of V^* (during the execution of (P, V^*) on x) is identically distributed to $M(x)$.

Output vs. View

- In fact, in the definition of computational zero knowledge (which we will cover in a moment), we can also replace the requirement of outputs with the corresponding requirement of views.
 - Whenever possible, we do this replacement because views are easier to work with.
- However, we can't always do the same thing—we will see counter examples.

Example of PZK Proof (1)

- Revisit the example of graph isomorphism:
 - P generates another random graph G_3 isomorphic to them.
 - P sends G_3 to V.
 - V sends a uniform bit to P as challenge.
 - If the challenge is 0, P sends back the isomorphic map between G_1 and G_3 ; otherwise, P sends back the isomorphic map between G_2 and G_3 .

Example of PZK Proof (2)

- If the challenge is 0, V accepts if and only if he receives an isomorphic map between $G1$ and $G3$; otherwise, V accepts if and only if he receives an isomorphic map between $G2$ and $G3$.
- Repeat the above procedure for a linear number of times.
- This interactive proof system is complete (why?), sound (why?), and PZK.

Why Is This Proof PZK? (1)

- We can construct a simulator M for view of V^* .
 - M chooses a uniform bit b .
 - If $b=0$, M simulates $G3$ with a random graph isomorphic to $G1$; otherwise, M simulates $G3$ with a random graph isomorphic to $G2$.
 - M feeds V^* with $G3$ and gets a simulated challenge.

Why Is This Proof PZK? (2)

- If the simulated challenge is equal to b , M simulates the response of P to the challenge with the isomorphic map chosen above.
- Otherwise, M restarts the above procedure.
- Finally, with high probability M gets an iteration of the procedure in which the simulated challenge is equal to b . M outputs the simulated variables.
- The output of M is identically distributed to the view of V^* .

Computational Zero Knowledge

- We can relax the requirement of PZK a little bit such that a polynomial-time adversary learns no knowledge.
- We say an interactive proof system (P, V) is computationally zero-knowledge (CZK) if for all V^* , there exists an efficient algorithm M such that
 - For each common input x , the output of V^* (after (P, V^*) is executed on common input x) is **computationally indistinguishable** from $M(x)$.

Computationally Indistinguishability

- A **probability ensemble** $\{x_i\}$ is a set of indexed random variables (normally of length polynomial in lengths of indices).
- Two probability ensembles $\{x_i\}$ and $\{y_i\}$ are computationally indistinguishable if for all probabilistic polynomial-time distinguisher A , for all polynomial $f()$, for all sufficiently long index i ,

$$|\Pr[A(i, x_i)=1] - \Pr[A(i, y_i)=1]| < 1/f(k)$$

where k is the length of i .

Honest Verifier Zero Knowledge

- As we have mentioned, in both PZK and CZK, we can consider either outputs or views; they are equivalent to each other.
 - Now we introduce a new definition that can only be based on views: Honest Verifier Zero Knowledge.
- We say an interactive proof system (P, V) is honest-verifier computationally zero-knowledge (HV-CZK) if there exists an efficient algorithm M such that
 - the view of V is computationally indistinguishable from $M(x)$.
- We can similarly define honest-verifier perfect zero knowledge (HV-PZK).

Properties of HV-CZK

- We can't replace views in the definition of HV-CZK with outputs.
 - Note that we can always make the output of V simulatable by making V output, e.g., 0. Clearly, this does not guarantee anything.
- It is trivial that CZK implies HV-CZK.
 - However, HV-CZK does not imply CZK.

Proof of Knowledge

- So far, all interactive proofs we considered are proofs of language membership.
 - In practice, we often use proof of knowledge (POK).
 - For example, we may want to have a proof of knowledge of the square root of a with respect to an RSA modulus N .
- But what do we mean by “an algorithm knows y ”?

Definition of POK

- The formal definition is complicated; so we skip it here.
- Informally, there exists a **knowledge extractor** M that outputs y after interacting with oracle machine describing the behavior of P .
 - The oracle machine gives the message P should send in response to any particular message P receives.
- But is this contradictory to ZK?

POK vs. ZK

- POK requires that interaction with P must lead to discovery of y .
- ZK requires that interaction with oracle machine describing P must not reveal anything.
- But POK is not contradictory to ZK.
 - Because oracle machine describing P is NOT P .
 - With the former, knowledge extractor can interact with **fragments** of P , not complete P .

Example of ZK POK (1)

- Let N be an RSA modulus. Suppose $a=y^2 \pmod{N}$. The following is a ZK POK of y .
- P chooses z at random and computes $b=z^2 \pmod{N}$.
- P sends b to V . (*)
- V chooses a random bit as challenge and sends it to P .
- If the challenge is 0, P sends z to V .

Example of ZK POK (2)

- If the challenge is 1, P sends yz back to V .
- If the challenge is 0, V accepts if and only if he receives a square root of b .
- If the challenge is 1, V accepts if and only if he receives a square root of ab .

Why Is This a POK?

- A knowledge extractor only needs to interact with the oracle machine describing P for twice:
 - First time with challenge 0;
 - Second time, starting from the step (*) in the first execution, and send challenge 1. (Note the extractor is working with a fragment of P here!)
 - Then the quotient of the two responses is exactly y .

Why Is This Proof ZK? (1)

- We can construct a simulator M for view of V^* .
 - M chooses a uniform bit t .
 - M chooses z' at random; if $t=0$, M simulates b with $(z')^2$; otherwise, M simulates b with $(z')^2/a$.
 - M feeds V^* with b and gets a simulated challenge.

Why Is This Proof ZK? (2)

- If the simulated challenge is equal to t , M simulates the response of P to the challenge with z' .
- Otherwise, M restarts the above procedure.
- Finally, with high probability M gets an iteration of the procedure in which the simulated challenge is equal to t . M outputs the simulated variables.
- The output of M is identically distributed to the view of V^* .

Graph Isomorphism

➤ Problem Instance

- Two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$

➤ Question

- Is there a bijection f from V_1 to V_2 , so $(u,v) \in E_1$ implies that $(f(u),f(v)) \in E_2$
- If such bijection exists, then graphs G_1 and G_2 are said to be isomorphic
- If such bijection does not exist, then graphs G_1 and G_2 are said to be non-isomorphic

Graph Non-isomorphism

- Input: graphs G_1 and G_2 over $\{1,2,\dots,n\}$
- Prover want to prove
 - G_1 and G_2 are not isomophic
- **Assumption**
 - Prover has unbounded computational power
 - Verifier has limited computational power

Proof Protocol

➤ Protocol (repeated for n rounds)

○ Verifier

- Randomly chooses $i=1$ or 2
- Selects a random permutation f and compute H to be the image of G_i under f , sends H to prover

○ Prover

- Determines the value j such that G_j is isomorphic to H
- Sends j to verifier

○ Verifier checks if $j=i$

○ If equal for n rounds, then accepts the proof

Correctness and Soundness

➤ Correctness

- If G_1 and G_2 are not isomorphic, then for any round, there is only one graph of G_1, G_2 that could produce H under a permutation f
- So if the verifier knows non-isomorphism, then each round a correct j will be computed

➤ Soundness

- If the verifier does not know (G_1 and G_2 are isomorphic), then each round two answers possible, and it has half chance to get the correct i chosen by the prover.

Graph Isomorphism

- Input: graphs G_1 and G_2 over $\{1,2,\dots,n\}$
- Prover want to prove
 - G_1 and G_2 are isomophic
- **Assumption**
 - Prover has unbounded computational power
 - Verifier has limited computational power

Proof Protocol

➤ Protocol (repeated for n rounds)

- Prover
 - Selects a random permutation f and compute H to be the image of G_1 under f , sends H to prover
- Verifier
 - Randomly chooses $i=1$ or 2 , sends it to prover
- Prover
 - Computes the permutation g such that H is the image of G_j under g , and sends g to verifier
- Verifier
 - checks if H is the image of G_j under g
- If yes for n rounds, then accepts the proof

Correctness and Soundness

➤ Correctness

- If G_1 and G_2 are isomorphic, and the verifier knows how to find the permutation between G_1 and G_2 , then each round a correct g will be computed

➤ Soundness

- If the verifier does not know (G_1 and G_2 are non-isomorphic or the permutation between G_1 and G_2), then each round prover can deceive the verifier is to guess the value i chosen by the verifier

Perfect Zero-Knowledge

- The graph isomorphism proof is ZKP
 - All information seen by the verifier is the same as generated by a random simulator
 - Define transcript of the proof as
 - $t=(G_1,G_2,(H_1,i,g_1),(H_2,i,g_2),\dots,(H_n,i,g_n))$
 - Anyone can generate the transcript without knowing which permutation carries G_1 to G_2
 - Hence the verifier gains nothing by knowing the transcript (I.e., the proof history)

Different ZKPs

- *perfect zero-knowledge*
 - if the distributions produced by the simulator and the proof protocol are distributed exactly the same. This is for instance the case in the first example above.
- *Statistical zero-knowledge*
 - means that the distributions are not necessarily exactly the same, but they are statistical close, meaning that their statistical difference is negligible function.
- *computational zero-knowledge*
 - if no efficient algorithm can distinguish the two distributions.

ZKP for Verifier

- Perfect Zero-knowledge for verifier
 - Suppose we have a poly-time interactive proof system and a poly-time simulator S . Let T be all yes-instance transcripts and let F be all transcripts generated by S . For any transcript t if
 - $\Pr(t \text{ occurs in } T) = \Pr(t \text{ occurs in } F)$
 - We say the interactive proof system are perfect zero-knowledge for the verifier

Isomorphism Proof: ZKP-verifier

- Graph isomorphism is a perfect zero-knowledge for verifier
 - A triple (H,i,g) . There are $2n!$ valid triples.
 - All triples (H,i,g) occurs equiprobable in some transcript
 - Here, assume that both the verifier and the prover are honest
 - Both of them randomly chooses parameters that supposed to be chosen randomly

Cheating Verifier

- What happened if verifier does not follow the protocol (does not choose i randomly)
 - Transcript produced by ZKP is not same as that produced by the random simulator anymore
 - The verifier may gain some information due to this imbalance
 - But, there is another expected poly-time simulator to generate the same transcript
 - Hence, the verifier still gains nothing

Perfect Zero-Knowledge

➤ Definition

- Suppose we have a poly-time interactive proof system, a poly-time algorithm V to generate random numbers by verifier, and a poly-time simulator S . Let T be all yes-instance transcripts (depending on V) and let F be all transcripts generated by S and V . For any transcript t if
 - $\Pr(t \text{ occurs in } T) = \Pr(t \text{ occurs in } F)$
- We say the interactive proof system are perfect zero-knowledge

Forging Simulator

- Initial transcript $t=(G_1, G_2)$, repeat n rounds
 - Let old-state=state(V), repeat follows
 - Chooses i_j from $\{1,2\}$ randomly
 - Chooses g_j to be a random permutation over $\{1,\dots,n\}$
 - Compute H_j to be the image of G_{i_j} under g_j
 - Call V with input H_j , obtaining a challenge i_j'
 - If $i_j=i_j'$, then concatenate (H_j, i_j, g_j) onto the end of t
 - Else reset V by state(V)=old-state
 - Until $i_j=i_j'$

Perfect Zero-knowledge

- The graph isomorphism is perfect ZKP
 - The expected running time of simulator is $2n$
 - For the k^{th} round of the interactive proof system
 - Let p_k be the probability that verifier chooses $i=1$
 - Then $(H,1,g)$ occurs in actual transcript with $p_k/n!$, $(H,2,g)$ occurs in actual transcript with $(1-p_k)/n!$
 - For simulator, when it terminates the simulation for the k^{th} round, same probability distribution for $(H,1,g)$ and $(H,2,g)$
 - Therefore, all transcripts by simulator or actual has the same probability distribution

Quadratic Residue

➤ Fiat-Shamir Identification

➤ Question

- Given integer $n=pq$, here p, q are primes.
- Prover wants to prove
 - Integer x is a quadratic residue mod n
 - In other words, knows u so $x=u^2 \pmod n$
- Quadratic residue is hard to solve if do not knowing the factoring of n

Proof Protocol

- Repeat the following for $\log_2 n$ times
 - Prover
 - Chooses random v less than n and computes $y = v^2 \pmod n$. Sends y to verifier
 - Verifier
 - Chooses a random i from $\{0,1\}$, sends it to prover
 - Prover
 - Computes $z = u^i v \pmod n$, sends z to verifier
 - Verifier
 - Checks if $z^2 = x^i y \pmod n$
 - Accepts the proof if equation holds all $\log_2 n$ rounds

Cont

➤ Correctness

- Show that verifier will accept the prover if indeed knows

➤ Soundness

- Show that verifier will detect the prover if it does not know with a good probability

➤ Zero-knowledge

- Show that verifier gets nothing from the protocol

Guillou Quisquater Protocol

- The GQ protocol is an extension of the Fiat Shamir protocol that limits the number t of rounds required.
- One Time Set-up:
 1. A trusted authority T selects two random primes p and q and forms a modulus $n = p \cdot q$.
 2. T defines a public exponent $v > 4$ with $\gcd(v, (p-1)(q-1)) = 1$ so that T can compute $s = v^{-1} \bmod (p-1)(q-1)$.
 3. T publishes parameters n and v .

Cont.

- Selection of per-user parameters:
 1. Each entity A has a unique identification $\text{Id}(A)$. Everyone can calculate a value $J(A) = f(\text{Id}(A)) \bmod n$ (the redundant identity).
 2. T gives to each entity A the secret data $\text{secret}(A) = J(A)^{-s}$, which it can calculate.

Cont.

- Protocol: A proves her identity to B using t rounds, each of which consists of:
 1. A selects a random secret r and sends her identity $\text{Id}(A)$ and $x = r^\nu \bmod n$ to B.
 2. B selects a random challenge e in $\{1, 2, \dots, \nu\}$.
 3. A computes and sends the following response to B: $y = r \cdot \text{secret}(A)^e \bmod n$.
 4. B receives y , constructs $J(A) = f(\text{Id}(A)) \bmod n$, computes $z = J(A)^e y^\nu$, and accepts this round if $z = x \bmod n$.
- In this protocol, ν determines the *security level*. In Fiat Shamir, $\nu = 2$ and there are many rounds. A fraudulent claimant can defeat the protocol by correctly guessing the challenge e (with a 1 in ν chance.) GQ seems secure, because we need to extract ν -roots modulo n .

Discrete Logarithm

➤ Question:

- Prover wants to prove to verifier that he knows x such that $y = g^x \pmod{p}$.
- Here g , y , and p are public information
- Prover does not want to publicize the value of x .

Proof Protocol

- Repeat the following for $\log_2 n$ times
 - Prover
 - Chooses random $j < p-1$ and computes $r = g^j \pmod p$.
Sends r to verifier
 - Verifier
 - Chooses a random i from $\{0,1\}$, sends it to prover
 - Prover
 - Computes $h = i \times j \pmod{p-1}$, sends h to verifier
 - Verifier
 - Checks if $g^h = y^{ir} \pmod n$
 - Accepts the proof if equation holds all $\log_2 n$ rounds

Cont

➤ Correctness

- Show that verifier will accept the prover if indeed knows

➤ Soundness

- Show that verifier will detect the prover if it does not know with a good probability

➤ Zero-knowledge

- Show that verifier gets nothing from the protocol

Bit Commitments

➤ Bit commitment

- Sometimes, it is desirable to give someone a piece of information, but not commit to it until a later date. It may be desirable for the piece of information to be held secret for a certain period of time.
- Example: stock up and down

Properties

➤ Bit commitment scheme

- The sender encrypts the b in some way
- The encrypted form of b is called blob
- Scheme $f: (X,b) \rightarrow Y$

➤ Properties

- Concealing: verifier cannot detect b from $f(x,b)$
- Binding: sender can open the blob by revealing x
- Hence, the sender must use random x to mask b

Methods

- One can choose any encryption method E
 - Function $f((x_0, k), b) = E_k((x_0, b))$
 - Need supply decryption k to reveal b
 - Assume the decryption method D is known
- Choose any integer $n = pq$, p and q are large primes
 - Function $f(x, b) = m^b x^2 \pmod n$
 - Goldwasser-Micali Scheme
 - Here $n = pq$, m is not quadratic residue, m, n public
 - $m x_1^2 \pmod n \neq x_2^2 \pmod n$
 - So sender can not change mind after commitment

Example: Bit Commitment based on Discrete Logarithm

- An example of bit commitment scheme:
 - Let p be a large prime.
 - Let g be a generator of \mathbb{Z}_p^* .
 - Commitment to 0: g^x , where x is a uniform random number in $[0, (p-1)/2)$.
 - Commitment to 1: g^x , where x is a uniform random number in $[(p-1)/2, p-1)$.
 - The scheme is secure under the assumption that discrete log is hard.

Coin Flip

➤ Even protocols

- Alice has a coin flip result i or j
- Bob wants to guess the result
- Alice has a message M that is commitment
- If bob guesses correct, Bob should have M received
- Alice starts with 2 pairs of public keys (E_i, D_i) and (E_j, D_j)
- Bob starts with a symmetric encryption S and a key k

Protocol

➤ Procedure

- Alice sends E_i, E_j to Bob
- Bob guess h and sends $y = E_h(k)$ to Alice
- Alice computes $p = D_j(y)$ and sends the encryption z of M by p using S to Bob
- Bob decrypts the encryption z using S and key k
- If the guess is correct, then Bob gets the commitment

Oblivious Transfer

- What is oblivious transfer
 - Alice wants to send Bob a secret in such a way that Bob will know whether he gets it, but Alice won't. Another version is where Alice has several secrets and transfers one of them to Bob in such a way that Bob knows what he got, but Alice doesn't. This kind of transfer is said to be oblivious (to Alice).

Oblivious Transfer

- A simple cryptographic primitive first studied by Rabin.
 - Kilian showed that you can essentially base ANY cryptographic protocol on this primitive.
- Suppose Alice sends a message to Bob.
 - We want that Bob receives the message with probability $\frac{1}{2}$.
 - We also want that Alice does not know whether Bob receives it or not.

Transfer Factoring

- By means of RSA, oblivious transfer of any secret amounts to oblivious transfer of the factorization of $n=pq$
 - Bob chooses x and sends $x^2 \bmod n$ to Alice
 - Alice (who knows p,q) computes the square roots $x, -x, y, -y$ of $x^2 \bmod n$ and sends one of them to Bob. Note that Alice does not know x .
 - If Bob gets one of y or $-y$, he can factor n . This means that with probability $1/2$, Bob gets the secret. Alice doesn't know whether Bob got one of y or $-y$ because she doesn't know x .

Factoring

- If one knows x and y such that
 - 1) $x^2 = y^2 \pmod n$
 - 2) $0 < x, y < n$, $x \neq y$ and $x + y \neq 0 \pmod n$
 - Number n is the production of two primes
- Then n can be factored
 - First $\gcd(x+y, n)$ is a factor of n
 - And $\gcd(x-y, n)$ is a factor of n

Quadratic Solution

- Given $n=p$, and a is a quadratic residue
 - Then there is two positive integers x less than n
 - Such that $x^2=a \pmod n$
- Given $n=pq$, and a is a quadratic residue
 - Then there is four positive integers x less than n
 - Such that $x^2=a \pmod n$

Oblivious Transfer of Message

- Alice has a message M , bob wants to get M through oblivious transfer
 - Alice does not know if Bob get M or not
 - Bob knows if he gets it or not
 - Bob gets M with probability $\frac{1}{2}$
 - Coin flipping can be used to achieve this

Security Analysis

- Bob indeed gets m with probability $\frac{1}{2}$.
 - Because the probability of picking a or $-a$ from the four square roots is $\frac{1}{2}$.
 - And because if a or $-a$ is sent to Bob, then Bob gets no help in factoring N .
- Alice has no way to learn whether Bob gets m or not.
 - Because she has no idea which of the four roots is a .

1-out-2 OT

- There are many variants of OT; 1-out-of-2 OT is a popular one.
 - Alice has two messages m_0 and m_1 .
 - Bob has a bit b (i.e., chooses to receive m_b).
 - Alice should not learn b .
 - Bob should not learn m_{1-b} .

Bellare-Micali Protocol (1)

- Let G be a cyclic group in which discrete log is hard; let g be a generator.
- Alice (or a public procedure) chooses y in G .
- Bob chooses x_b and computes $y_b = g^{x_b}$.
- Bob also computes $y_{1-b} = y / y_b$.

Bellare-Micali Protocol (2)

- Bob sends y_0, y_1 to Alice.
- Alice checks $y_0 y_1 = y$ and encrypts m_0, m_1 using ElGamal public parameters y_0, y_1 , respectively.
- Bob decrypts the encryption of m_b using x_b .

Security Analysis

- Bob can't learn m_{1-b} because he does not know the discrete log of y_{1-b} .
 - Guaranteed by the security of ElGamal cryptosystem.
- Alice can't learn b because everything she observes is independent of b .

1-out-of-2 OT implies OT

- Suppose we have a 1-out-of-2 OT protocol. Then we can construct an OT protocol.
 - Alice randomly permutes (m, trash).
 - Alice runs 1-out-of-2 OT with Bob with the above permuted pair.
 - Regardless of Bob's choice in 1-out-of-2 OT, he always receives m with probability $\frac{1}{2}$.

1-out-of-n OT

- An extension of 1-out-of-2 OT.
 - Alice has n messages.
 - Bob has an input in $[0, n-1]$ (i.e., chooses to receive one of the messages).
 - Alice should not learn Bob's choice.
 - Bob should not learn the other $n-1$ messages.

1-out-of-2 OT implies 1-out-of-n OT (1)

- Suppose we have a 1-out-of-2 OT protocol. Then we can construct a 1-out-of-n OT protocol.
- Let's temporarily assume $n=2^m$.
 - Alice chooses $2m$ random numbers $k_1, k'_1, \dots, k_m, k'_m$.
 - For each message m_i , for each j : if the j th bit of i is 0, then the message is xor'd by k_j ; otherwise the message is xor'd by k'_j .
 - For example: $i=1011$, then m_i is xor'd by k_1 xor k'_2 xor k_3 xor k_4 .

1-out-of-2 OT implies 1-out-of-n OT (2)

- For each j , Alice and Bob run a 1-out-of-2 OT protocol such that Bob learns one of the two random numbers k_j and k'_j .
 - The random numbers Bob learns can only help him learn one message.
 - Alice clearly can't learn Bob's choice.
- But what if n is not a power of 2?

1-out-of-2 OT implies 1-out-of-n OT (3)

- When n is not a power of 2, let n' be the smallest power of 2 such that $n' > n$.
 - Alice runs a 1-out-of- n' OT protocol with Bob using the n messages and $n' - n$ pieces of trash.
 - Alice tells Bob where the $n' - n$ pieces of trash are, so that Bob would not choose to receive any of them.
 - Clearly, Bob will receive a message of his choice; Alice won't learn Bob's choice; Bob won't learn other messages.

Contract Signing

➤ It requires two things

- Commitment: after certain point, both parties are bound by the contract, until then, neither is
- Unforgeability: it must be possible for either party to prove the signature of the other party

➤ With Pen and Paper

- Two party together, face to face
- Sign simultaneously (or one character by one)

Remote Contract Signing

➤ Simple one

- Alice generate a signature, divided into SL, SR
- Alice randomly select two keys KL, KR
- Encrypt the signatures SL, SR
- Transfer encrypted SL,SR to Bob
- Obliviously transfer KL, KR to bob
 - Bob gets one, but Alice does not know which one
- Bob decrypts the encrypted SL or SR
 - Verify the decrypted signature, if invalid, stop
- Alice sends the i th bits of keys KL and KR to Bob
 - Here $i=1$ to the length of the keys

Cont.

- The protocol will be conducted by Bob also
 - What is the chance of Alice to cheat successfully?
 - Alice can guess which key will be transferred obviously --- (1/2 chance)
 - Then send wrong signature for the other half or send the wrong key of the other half
 - Bob can not detect it if Alice can guess which key Bob got
 - How about Alice stop prematurely?
 - One bit advance over Bob
- Enhanced protocol
 - Use many pair of keys and signatures instead of one

Applications

- oblivious transfer to be a fundamental and important problem in cryptography. It is considered one of the critical problems in the field, because of the importance of the applications that can be built based on it.
 - In particular, it is a `complete' for secure multiparty computation: that is given an implementation of oblivious transfer it is possible to securely evaluate any polynomial time computable function without any additional primitive.

SECURE MPC (MULTI- PARTY-COMPUTATION)

Secure multi-party computation

- closely related to the idea of zero-knowledgeness.
- In general it refers to computational systems in which multiple parties wish to jointly compute some value based on individually held secret bits of information, but do not wish to reveal their secrets to one another in the process.

Cont.

- For example, two individuals who each possess some secret information— x and y , respectively—may wish to jointly compute some function $f(x,y)$ without revealing any information about x and y other than can be reasonably deduced by knowing the actual value of $f(x,y)$,
 - where "reasonably deduced" is often interpreted as equivalent to computation within polynomial time. The primary motivation for studying methods of secure computation is to design systems that allow for maximum utility of information without compromising user privacy.

Secure Computation

- Secure 2-party/multi-party computation: general-purpose cryptographic protocol.
 - Suppose there are n parties.
 - A common public input: function $f()$.
 - $f()=(f1(),f2())$
 - Each party has a private input x_i .
 - Can we construct a protocol for securely computing $f(x_1, \dots, x_n)$?
 - A should only learn $f1(x_1, \dots, x_n)$;
 - B should only learn $f2(x_1, \dots, x_n)$.

Adversary Models

- There are two major adversary models for secure computation: Semi-honest model and fully malicious model.
 - Semi-honest model: all parties follow the protocol; but dishonest parties may be curious to violate others' privacy.
 - Fully malicious model: dishonest parties can deviate from the protocol and behave arbitrarily.
 - Clearly, fully malicious model is harder to deal with.

Security in Semi-Honest Model

- A 2-party protocol between A and B (for computing a **deterministic function $f()$**) is secure in the semi-honest model if there exists an efficient algorithm MA (resp., MB) such that
 - the view of A (resp., B) is **computationally indistinguishable** from $MA(x_1, f_1(x_1, x_2))$ (resp., $MB(x_2, f_2(x_1, x_2))$).
- We can have a similar (but more complex) definition for multiple parties.

Security in Malicious Model (1)

- In the malicious model, security is much more complex to define.
- For example, there are unavoidable attacks:
 - What if a malicious party replaces his private input at the very beginning?
 - What if a malicious party aborts in the middle of execution?
 - What if a malicious party aborts at the very beginning?

Security in Malicious Model (2)

- To deal with these complications, we use an approach of ideal world vs. real world.
 - Consider an ideal world in which all parties (including the malicious ones) give their private inputs to a trusted authority.
 - After receiving all private inputs, the authority computes the output and sends it to all parties.
 - Clearly, those unavoidable attacks also exist in this ideal world.

Security in Malicious Model (3)

- We require that, for any adversary in the real world, there is an “equivalent” adversary in the ideal world, such that
 - The outputs in the real world are computationally indistinguishable from those in the ideal world.
- In this way, we capture the idea that
 - All “avoidable” attacks are prevented.
 - “Unavoidable” attacks are allowed.

Yao's Theorem

- The first completeness theorem for secure computation.
- It states that for ANY efficiently computable function, there is a secure two-party protocol in the semi-honest model.
 - Therefore, in theory there is no need to design protocols for specific functions.
 - Surprising!

The Setting

- Yao's theorem applies to the following setting of **Secure Function Evaluation**:
 - Alice has a function $f()$, which is efficiently computable.
 - Bob has an input x .
 - We need a way to evaluate $f(x)$ such that
 - Alice learns nothing;
 - Bob learns only $f(x)$.

Secure Function Evaluation vs. Secure Two-Party Computation

- We can view Secure Function Evaluation of $f()$ as a special case of Secure Two-Party Computation.
 - Because $f()$ is also an input, after all.
- We can also build Secure Two-Party Computation of $F()$ based on Secure Function Evaluation.
 - Just define $f(y)=F_2(x,y)$ and evaluate $f()$.
 - Then define $f'(x)=F_1(x,y)$ and evaluate $f'()$.
- So Secure Function Evaluation is essentially equivalent to Secure Two-Party Computation.

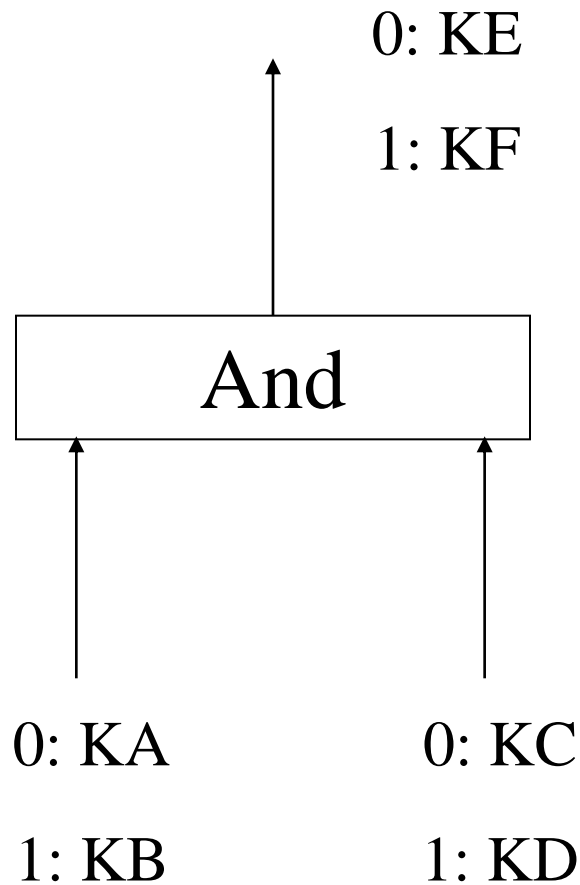
Circuit Computation

- The design of Yao's protocol is based on circuit computation.
 - Recall any (efficiently) computable function can be represented as a family of (polynomial-size) boolean circuits.
 - Recall such a circuit consists of **and**, **or**, and **not** gates.
 - It is enough if we can evaluate Alice's private circuit at Bob's private input.

Garbled Circuit

- We can represent Alice's circuit with a garbled circuit that does not reveal any knowledge about the circuit.
 - For each edge in the circuit, we use two random keys to represent 0 and 1 respectively.
 - We represent each gate with 4 ciphertexts, for input $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, respectively.
 - These ciphertexts should be permuted randomly.
 - The ciphertext for input (a,b) is the key representing the output $\text{Gate}(a,b)$ encrypted by the keys representing a and b .

Example of a Gate



- This gate is represented by: (a random permutation of)

$$E_{KA}(E_{KC}(KE));$$

$$E_{KB}(E_{KC}(KE));$$

$$E_{KA}(E_{KD}(KE));$$

$$E_{KB}(E_{KD}(KF)).$$

Evaluation of Garbed Circuit

- Given the keys representing the inputs of a gate, we can easily obtain the key representing the output of the gate.
 - Only need to decrypt the corresponding entry.
 - But we do not know which entry it is? We can decrypt all entries. Suppose each cleartext contains some redundancy (like a hash value). Then only decryption of the right entry can yield such redundancy.

Translating Input?

- So, we know that, given the keys representing Bob's private input, we can evaluate the garbled circuit.
 - Suppose Alice also sends the garbled circuit to Bob. Then Bob can evaluate the garbled circuit if he knows how to translate his input to the keys.
- But Alice can't give the translation table to Bob.
 - Otherwise, Bob can evaluate the circuit at ANY input.

Jump Start with Oblivious Transfer

- A solution to this problem is 1-out-of-2 OT for each input bit.
 - Alice sends the keys representing 0 and 1;
 - Bob chooses to receive the key representing his input at this bit.
 - Clearly, Bob can't evaluate the circuit at any other input.

Finishing the Evaluation

- At the end of evaluation, Bob gets the keys representing the output bits of circuit.
 - Alice sends Bob a table of the keys for each output bit.
 - Bob translates the keys back to the output bits.
- For privacy, we need to be careful:
 - The topology of the circuit should be the same for all circuits of a particular input size.
 - Then privacy is guaranteed.

More Completeness Theorems in Semi-Honest Model

- Goldreich-Micali-Widgerson Theorem: General-purpose secure protocols exist for multiple parties.
 - But just like Yao Theorem, the conclusion relies on intractability assumptions.
- If we assume there is a perfectly private channel between each pair of parties, we can construct protocols without intractability assumptions.

From Semi-Honest to Malicious

- Based on general-purpose protocols in the semi-honest model, we can construct general-purpose protocols in the malicious model.
 - The main tools are bit commitment, (verifiable) secret sharing, and zero-knowledge proofs.
 - In fact, “compilers” are available to automatically translating protocols.

Example: Sending Square

- Consider the following problem: Alice wants to send $a^2 \pmod N$, where N is an RSA modulus) to Bob, where a is her private input.
- In the semi-honest model, Alice only needs to send a^2 to Bob directly.
- However, in the malicious model, Alice may send a' to Bob, whose square root she does not know.
 - To deal with this cheating possibility, we can have Alice prove that she knows the square root of a' .

Multi-Party Computation in Malicious Model (1)

- In the malicious model, the number of malicious parties affect the security we can achieve.
 - If there could be a majority of malicious parties, then early abortion of the protocol is unavoidable.
 - But if the protocol is not aborted, we still can achieve a form of security.

Multi-Party Computation in Malicious Model (2)

- If we restrict the number of malicious parties to a strict minority, we can guarantee that the honest parties finish the protocol securely.
 - We can have each party distributes shares of his private input with threshold = majority.
 - Then computation is performed on shares.
 - Since malicious parties are minority, they can't abort the entire computation. (But of course, they can replace or refuse to give their own private inputs.)