# COM5336 Cryptography
# Lecture 12

Scott CH Huang

# Certificate

# Certificate

- A **public-key certificate** is a digitally signed statement from one entity, saying that the public key (and some other information) of another entity has some specific value.

# More terms

- *Digitally Signed*
  - If some data is *digitally signed* it has been stored with the "identity" of an entity, and a signature that proves that entity knows about the data. The data is rendered unforgeable by signing with the entitys' private key.
- *Identity*
  - A known way of addressing an entity. In some systems the identity is the public key, in others it can be anything from a Unix UID to an Email address to an X.509 Distinguished Name.
- *Entity*
  - An entity is a person, organization, program, computer, business, bank, or something else you are trusting to some degree.

# More about CA

- Why need it
  - In a large-scale networked environment it is impossible to guarantee that prior relationships between communicating entities have been established or that a trusted repository exists with all used public keys. Certificates were invented as a solution to this public key distribution problem. Now a *Certification Authority* (CA) can act as a *Trusted Third Party*. CAs are entities (e.g., businesses) that are trusted to sign (issue) certificates for other entities. It is assumed that CAs will only create valid and reliable certificates as they are bound by legal agreements. There are many public Certification Authorities, such as VeriSign, Thawte, Entrust, and so on. You can also run your own Certification Authority using products such as the Netscape/Microsoft Certificate Servers or the Entrust CA product for your organization.

# Who uses Certificate?

- Probably the most widely visible application of X.509 certificates today is in web browsers (such as Netscape Navigator and Microsoft Internet Explorer) that support the SSL protocol.
  - SSL (Secure Socket Layer) is a security protocol that provides privacy and authentication for your network traffic. These browsers can only use this protocol with web servers that support SSL.
- Other technologies that rely on X.509 certificates include:
  - Various code-signing schemes, such as signed Java Archives, and Microsoft Authenticode.
  - Various secure E-Mail standards, such as PEM and S/MIME.
  - E-Commerce protocols, such as SET.

# How to create certificate?

- There are two basic techniques used to get certificates:
  - you can create one yourself (using the right tools, such as keytool)
    - Not everyone will accept self-signed certificates, ☺
  - you can ask a Certification Authority to issue you one (either directly or using a tool such as **keytool** to generate the request).
- The main inputs to the certificate creation are:
  - Matched *public and private keys*, generated using some special tools (such as keytool), or a browser.
  - *information about the entity being certified* (e.g., you). This normally includes information such as your name and organizational address. If you ask a CA to issue a certificate for you, you will normally need to provide proof to show correctness of the information.

# business

- Many companies sale the service of creating the certificate (such as SSL certificate)






# X.509 Authentication Service

- Public key certificate associated with user
  - The certificates are created by Trusted Authority
  - Then placed in the directory by TA or user
  - Itself is not responsible for creating certificate
  - It includes
    - Version, serial number, signature algorithm identifier, Issuer name, issuer identifier, validity period, the user, user identifier, user’ s public key, extensions, signature by TA
  - The signature by TA guarantees the authority
  - Certificates can be used to certify other TAs
  - Y<<X>>: certificate of user X issued by TA Y

# What is inside X.509 certificate?

- **Version**
  - Thus far, three versions are defined.
- **Serial Number**
  - distinguish it from other certificates it issues. This information is used in numerous ways, for example when a certificate is revoked its serial number is placed in a Certificate Revocation List (CRL).
- **Signature Algorithm Identifier**
  - This identifies the algorithm used by the CA to sign the certificate.
- **Issuer Name**
  - The X.500 name of the entity that signed the certificate. This is normally a CA. Using this certificate implies trusting the entity that signed this certificate. *root or top-level CA* certificates, the issuer signs its own certificate.

# cont

- **Validity Period**
  - This period is described by a start date and time and an end date and time, and can be as short as a few seconds or almost as long as a century. It depends on a number of factors, such as the strength of the private key used to sign the certificate or the amount one is willing to pay for a certificate. This is the expected period that entities can rely on the public value, if the associated private key has not been compromised.
- **Subject Name**
  - The name of the entity whose public key the certificate identifies. This name uses the X.500 standard, so it is intended to be unique across the Internet.
- **Subject Public Key Information**
  - together with an algorithm identifier

# Certificate Revocation

- Need the private key together with the certificate to revoke it
- The revocation is recorded at the directory
- Each time a certificate is arrived, check the directory to see if it is revoked

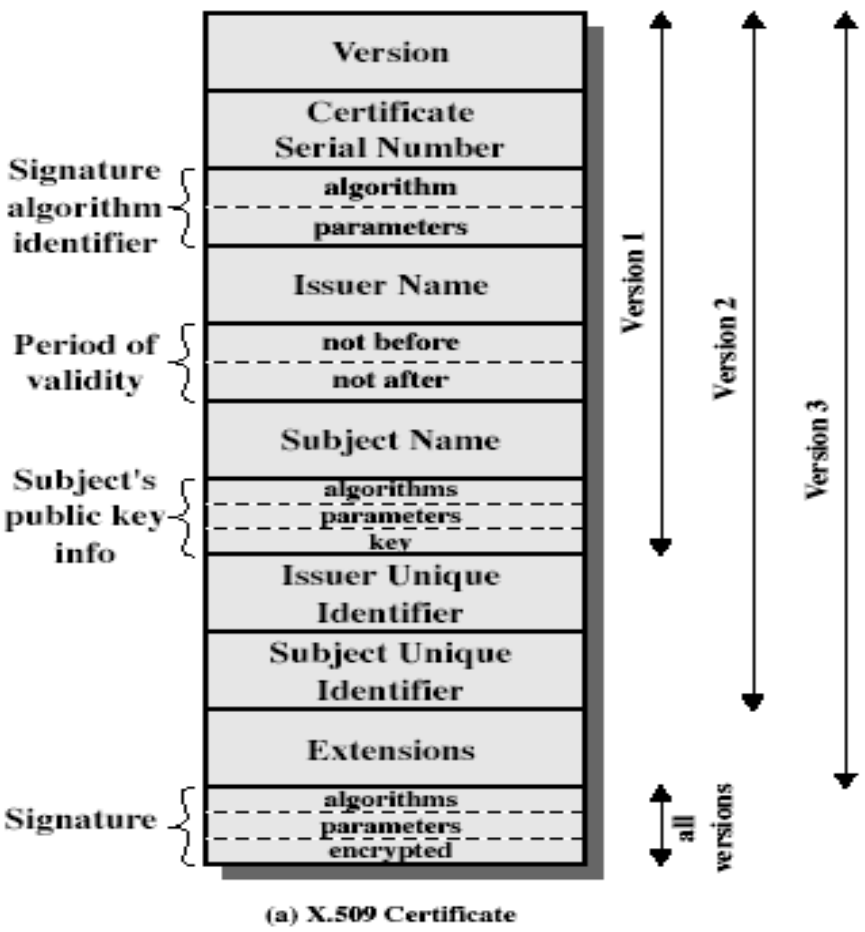
# X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining some info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user
  - signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended

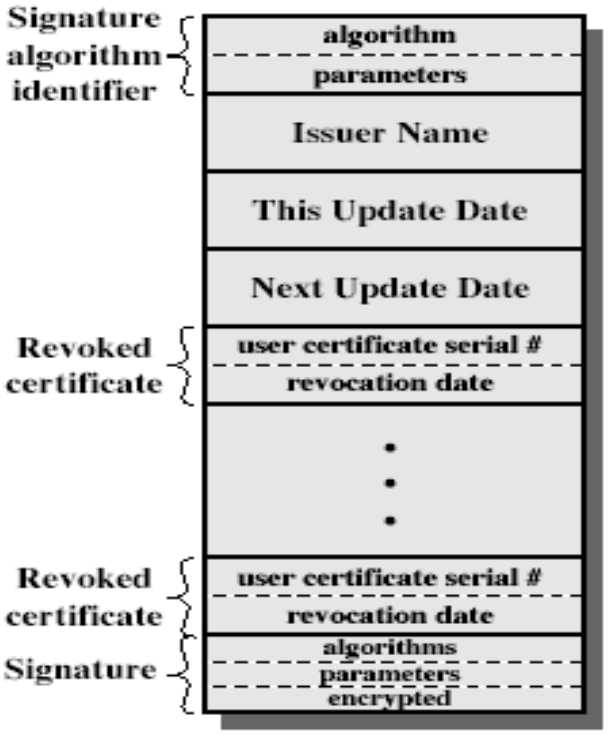
# X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- notation `CA<<A>>` denotes certificate for A signed by CA

# X.509 Certificates




(a) X.509 Certificate




(b) Certificate Revocation List

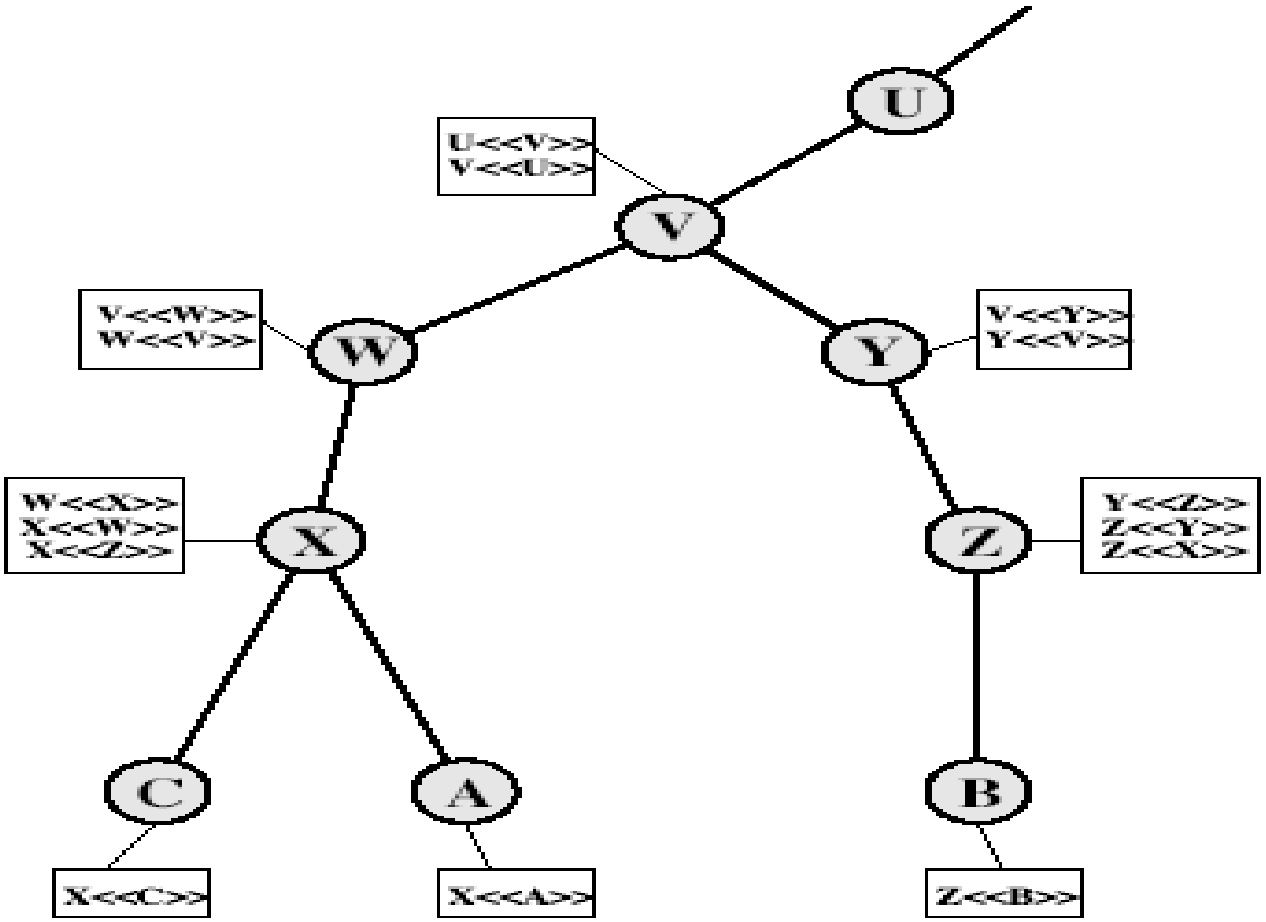
# Obtaining a Certificate

- any user with access to CA can get any certificate from it
- only the CA can modify a certificate
- because cannot be forged, certificates can be placed in a public directory

# CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- use certificates linking members of hierarchy to validate other CA's
  - each CA has certificates for clients (forward) and parent (backward)
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

# CA Hierarchy Use



# Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CA's maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certs with CA's CRL

# Authentication Procedures

- X.509 includes three alternative authentication procedures:
- One-Way Authentication
- Two-Way Authentication
- Three-Way Authentication
- all use public-key signatures

# One-Way Authentication

- 1 message ( A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A

# Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B

# Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing signed copy of nonce from B
- means that timestamps need not be checked or relied upon

# X.509 Version 3

- has been recognised that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields defined a general extension method
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

# Certificate Extensions

- key and policy information
  - convey info about subject & issuer keys, plus indicators of certificate policy
- certificate subject and issuer attributes
  - support alternative names, in alternative formats for certificate subject and/or issuer
- certificate path constraints
  - allow constraints on use of certificates by other CA’ s

# Top 10 SSL Certificate Providers

| Name | Rating | Starting price / 1yr in USD | SSL Type | Warranty | Visit Provider |
|---|---|---|---|---|---|
| Network solutions | 5 | $49.99 | OV EV DV | $1000000 | More |
| Entrust | 4.5 | $155 | EV OV | $1000000 | More |
| Symantec | 4.5 | $399 | EV OV | $1500000 | More |
| Digicert | 4 | $195 | OV EV DV | $1000000 | More |
| GeoTrust | 4 | $149 | OV EV DV | $500000 | More |
| Thawte | 4 | $149.99 | OV EV DV | $1000000 | More |
| Rapid SSL | 3.5 | $49 | DV | $1000000 | More |
| Comodo | 3.5 | $64.59 | OV EV DV | $1000000 | More |
| Geocerts SSL | 3.5 | $99 | OV EV DV | $500000 | More |
| GoDaddy | 3.2 | $63.10 | OV EV DV | $1000000 | More |
| SSL2BUY | 5 | $8.67 | OV EV DV | $10,000.00 | More |

# Identification

# Identification

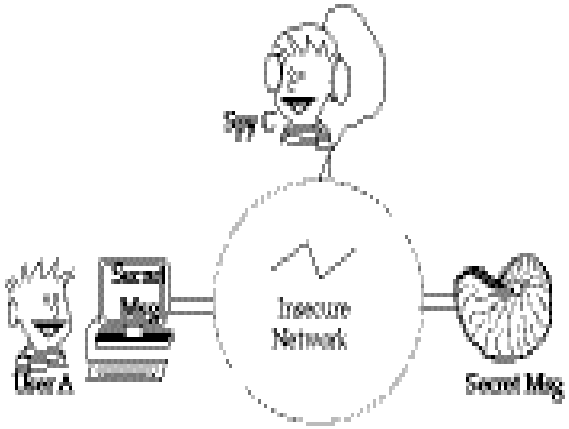
- Identification: user authentication
  - convince system of your identity
  - before it can act on your behalf
  - sometimes also require that the computer verify its identity with the user
- Based on three methods
  - what you know
  - what you have
  - what you are
- Verification
  - Validation of information supplied against a table of possible values based on users claimed identity

# What you Know

- Passwords or Pass-phrases
  - prompt user for a login name and password
  - verify identity by checking that password is correct
  - on some (older) systems, password was stored clear
  - more often use a one-way function, whose output cannot easily be used to find the input value
  - either takes a fixed sized input (eg 8 chars)
  - or based on a hash function to accept a variable sized input to create the value
  - important that passwords are selected with care to reduce risk of exhaustive search

# Weakness

- Traditional password scheme is vulnerable to eavesdropping over an insecure network



# Solutions?

- One-time password
  - these are passwords used once only
  - future values cannot be predicted from older values
- Password generation
  - either generate a printed list, and keep matching list on system to be accessed
  - or use an algorithm based on a one-way function *f* (eg MD5) to generate previous values in series (eg SKey)
    - start with a secret password s, and number N , $p_0 = f^N(s)$
    - *i*th password in series is $p_i = f^{N-i}(s)$
  - must reset password after N uses

# What you Have

- Magnetic Card, Magnetic Key
  - possess item with required code value encoded
- Smart Card or Calculator
  - may interact with system
  - may require information from user
  - could be used to actively calculate:
  - a time dependent password
  - a one-shot password
  - a challenge-response verification
  - public-key based verification

# What you Are

- Verify identity based on your physical characteristics, known as biometrics
- Characteristics used include:
  - Signature (usually dynamic)
  - Fingerprint, hand geometry
  - face or body profile
  - Speech, retina pattern
- Tradeoff between
  - false rejection (type I error)
  - false acceptance (type II error)

# Secret Sharing

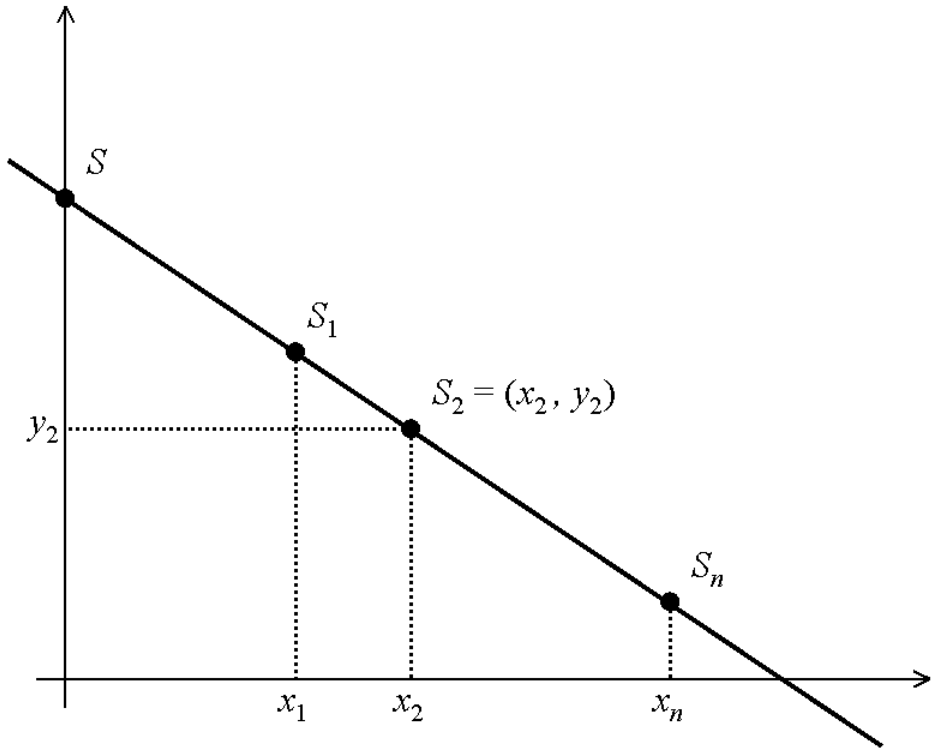
# Threshold Scheme

- 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

# 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 $a_i$
  - Dealer computes $y_i=f(x_i)$
    - Here $f(x)=k+\Sigma a_j x^j \bmod p$
  - Dealer gives share $y_i$ to person $p_i$

# Geometry View



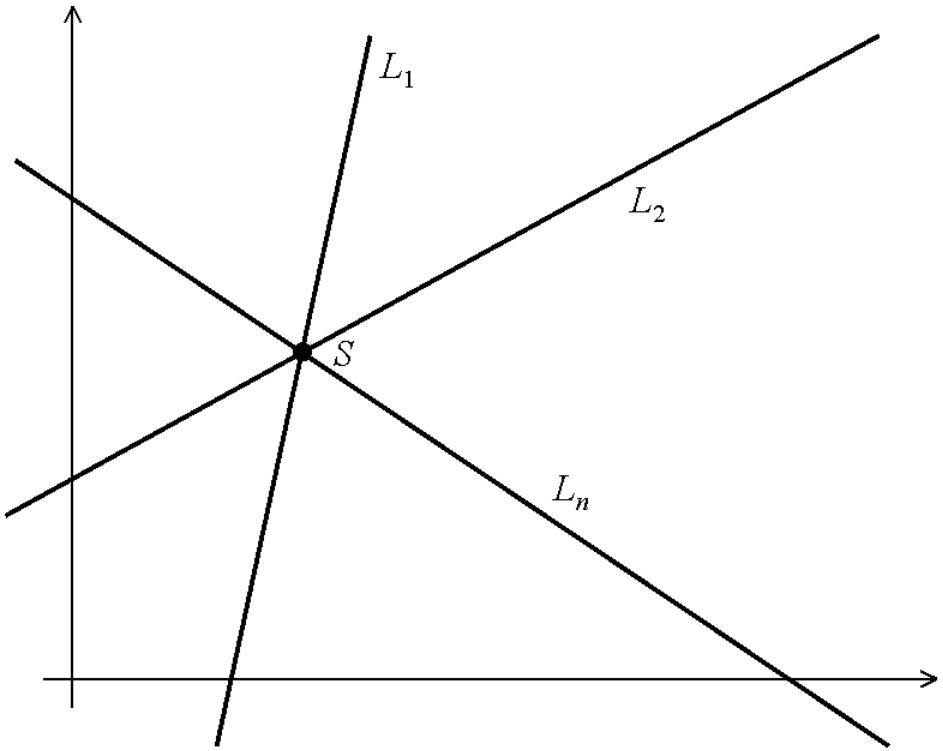
# Simple (t,t) Sharing

- Procedure
  - D secretly chooses *t-1* random elements $y_i$ from $Z_n$
  - D computes
    - Value $y_t = K - \Sigma y_j \bmod n$
  - 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

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

# Chinese Remainder Theorem

- Given a number $m<n$, and $n=n_1n_2...n_k$,
  - 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

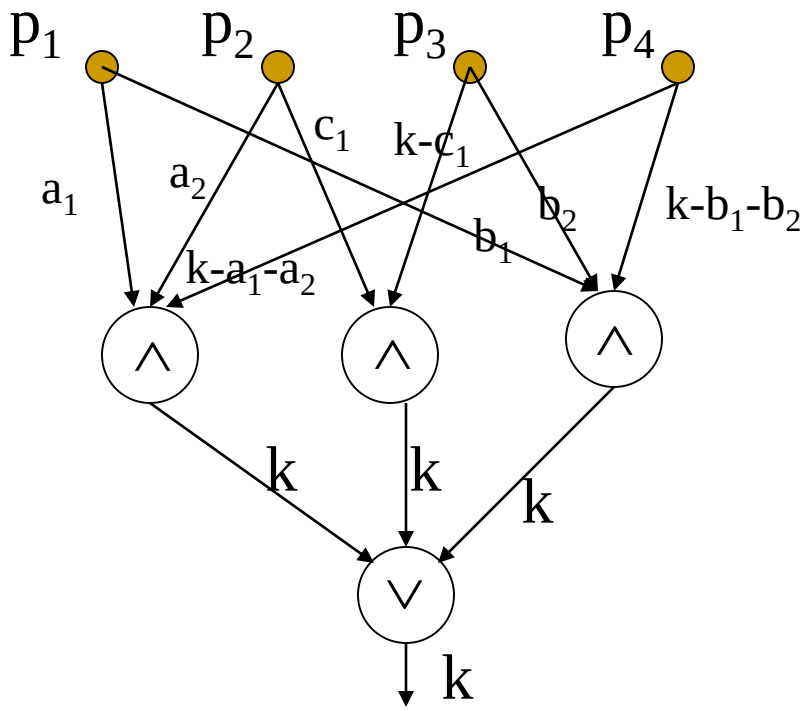
- Each user pre-computes
  - $N_i = n/n_i$
  - Inverse of $N_i$: $y_i = N_i \bmod n_i$
  - Compute the product $s_i = a_i N_i y_i \bmod n$
- Recover the secret $m$
  - Each user submits $s_i$
  - Computes $s_1 + s_2 + .... + s_k \bmod n$

# 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 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

# 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

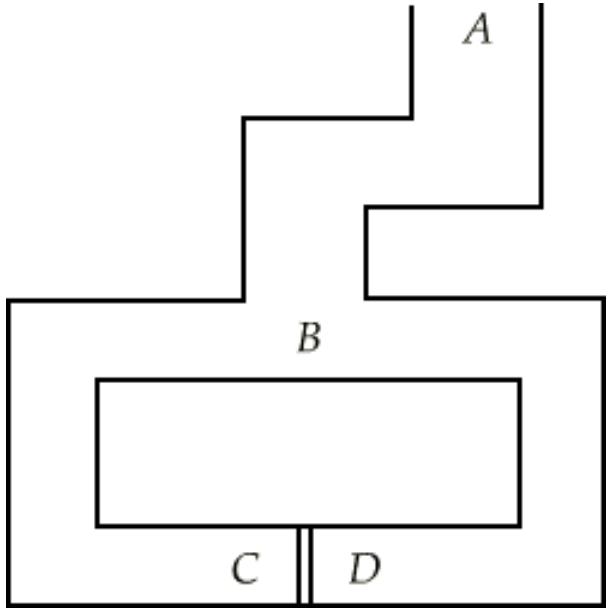
- Desired properties of interactive proofs
  - *Completeness*: The verifier always accepts the proof if the prover knows the fact and both the prover and the verifier follow the protocol.
  - *Soundness*: Verifier always rejects the proof if prover doesnot know the fact, and verifier follows protocol.
  - *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.

# Typical Protocol

- 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.

- 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.

# 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,…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,…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 $G_j$ under f, sends H to verifier
  - 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
    - $t=(G_1,G_2,(H_1,i,g_1),(H_2,i,g_2),....(H_n,i,g_n))$
  - 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)

# 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 occurs in T)=Pr(t 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 occurs in T)=Pr(t 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,...n}
    - Compute $H_j$ to be the image of $G_i$ under g
    - 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

- **Feige-Fiat-Shamir Identification**
- Question
  - Given integer n=pq, here p, q are primes.
  - Prover wants to prove
    - Integer $s_j$ is a quadratic residue of $v_j$ mod n
    - In other words, knows $v_j$ s.t. $v_j=s_j^2$ mod n for all j
  - 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 r less than n, a random sign s, and computes $x=sr^2 \bmod n$. Sends y to verifier
  - Verifier
    - Chooses n random bits $a_j$ from {0,1}, sends it to prover
  - Prover
    - Computes $y= rs_1^{a_1} s_2^{a_2} \cdots s_k^{a_k} \bmod n$, sends y to verifier
  - Verifier
    - Checks if $y^2= \pm rs_1^{a_1} s_2^{a_2} \cdots s_k^{a_k} \bmod n$
  - 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 Id(A). Everyone can calculate a value J(A) = $f$(Id(A)) mod $n$ (the redundant identity).
2. T gives to each entity A the secret data 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 Id(A) and $x = r^v \bmod n$ to B.
2. B selects a random challenge *e* in {1, 2, ... , *v*}.
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*(Id(A)) mod *n*, computes $z = J(A)^e y^v$, and accepts this round if $z = x \bmod n$.

- In this protocol, *v* determines the *security level.* In Fiat Shamir, *v* = 2 and there are many rounds. A fraudulent claimant can defeat the protocol by correctly guessing the challenge *e* (with a 1 in *v* chance.) GQ seems secure, because we need to extract *v*-roots modulo *n.*

# Discrete Logarithm

- Question:
  - Prover wants to prove to verifier that he knows x such that $y=g^x \mod p$ .
  - 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 $r=g^j \bmod p$. Sends r to verifier
  - Verifier
    - Chooses a random i from {0,1}, sends it to prover
  - Prover
    - Computes $h=i x + j \bmod p-1$, sends h to verifier
  - Verifier
    - Checks if $g^h = y^i r \bmod n$
  - 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)→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 $f(x,b)=m^b x^2 \bmod n$
    - Goldwasser-Micali Scheme
    - Here n=pq, m is not quadratic residule, m,n public
    - $mx_1^2 \bmod n \neq x_2^2 \bmod n$
    - So sender can not change mind after commitment

# 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 (Ei,Di) and (Ej,Dj)
  - Bob starts with a symmetric encryption S and a key k

# Protocol

- Procedure
  - Alice sends Ei, Ej to Bob
  - Bob guess h and sends y=Eh(k) to Alice
  - Alice computes p=Dj(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).

# 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 $x^2$ mod n to Alice
  - Alice (who knows *p*,*q*) computes the square roots x,-x,y,-y of $x^2$ mod 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) $x^2=y^2 \bmod n$
  - 2) $0<x,y<n$, $x \neq y$ and $x+y \neq 0 \bmod n$
  - 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 $x^2=a \bmod n$
- Given n=pq, and a is a quadratic residue
  - Then there is four positive integers x less than n
  - Such that $x^2=a \bmod n$

# 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 ½
  - Coin flipping can be used to achieve this

# 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 ith 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 obliviously ---(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

# Pseudorandom Number Generation

# Random number, Pseudorandom

- The outputs of pseudorandom number generators are not truly random
  - they only approximate some of the properties of random numbers.
  - "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." --- John von Neumann
  - Truely random numbers can be generated using hardware random number generators

# Inherent non-randomness

- Because any PRNG run on a deterministic computer (contrast quantum computer) is deterministic, its output will inevitably have certain properties that a true random sequence would not exhibit.
  - guaranteed periodicity—it is certain that if the generator uses only a fixed amount of memory then, given a sufficient number of iterations, the generator will revisit the same internal state twice, after which it will repeat forever. A generator that isn't periodic can be designed, but its memory requirements would grow as it ran. In addition, a PRNG can be started from an arbitrary starting point, or seed state, and will always produce an identical sequence from that point on.

# cont

- In practice, many PRNGs exhibit artifacts which can cause them to fail statistically significant tests. These include, but are certainly not limited to:
  - Shorter than expected periods for some seed states (not full period)
  - Poor dimensional distribution
  - Successive values are not independent
  - Some bits may be 'more random' than others
  - Lack of uniformity

# Pseudo-random Bit Generator

- Several applications
  - Key generation
  - Some encryption algorithms, or one-time pad
- Let $l>k$ be integers
  - Function f: $Z_2^k \rightarrow Z_2^l$ computable in poly-time
  - Then f called $(k,l)$-pseudo-random bit generator
  - The input $s_0 \in Z_2^k$ is called the seed
  - Output $f(s_0)$ is called the pseudo-random string

# Desired Properties

- Three important properties:
  - Unbiased (uniform distribution):
    - All values of whatever sample size is collected are equiprobable
  - Unpredictable (independence):
    - It is impossible to predict what the next output will be, given all the previous outputs, but not the internal "hidden" state.
  - Irreproducible:
    - Two of the same generators, given the same starting conditions, will produce different outputs.

# Desired Properties

- Usually when a person says
  - A "good" pseudo-random number generator
    - they mean it is unbiased.
  - A "true" PRNG
    - they usually mean it's irreproducible
  - A "cryptographically strong"  PRNG
    - they mean it's unpredictable
  - Very rarely they mean it's all threes

# More Properties

- Long period
  - The generator should be of long period
- Fast computation
  - The generator should be reasonably fast
- Security
  - The generator should be secure
  - What is security level of PRNG?

# Security

- A PRNG suitable for cryptographic applications is called a *cryptographically secure PRNG* (CSPRNG).
  - Its output should not only pass all statistical tests for randomness but satisfy some additional cryptographic requirements.
  - Used in many aspects of cryptography require random numbers, for example:
    - Key generation
    - Nonces
    - Salts in certain signature schemes, (ECDSA, RSASSA-PSS).
    - One-time pads

# CSPRNG

- CSPRNG requirements fall into two groups:
  - their statistical properties are good (passing tests of randomness),
  - they hold up well in case of attack, even when (part of) their secrets are revealed.
- A CSPRNG should satisfy the 'next-bit test'.
  - Given the first *l* bits of a random sequence there is no polynomial-time algorithm that can predict the next bit with probability of success significantly higher than 1/2.
  - It has been proven that a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness.
- should withstand state compromise extensions.
  - That is, in the unfortunate case that part or all of the state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the incident. Also if there is an input of entropy, it should be infeasible to use knowledge of the state to predict future conditions of the state.

# Example

- the CSPRNG being considered produces output by computing some function of the next digit of pi (ie, 3.1415...),
  - it may well be random as pi appears to be a random sequence.
  - However, this does not satisfy the next-bit test, and
  - thus is not cryptographically secure.
  - There exists an algorithm that will predict the next bit.

# Design

- divide designs of CSPRNGs into classes:
  - those based on block ciphers;
  - those based upon hard mathematical problems, and
  - special-purpose designs.

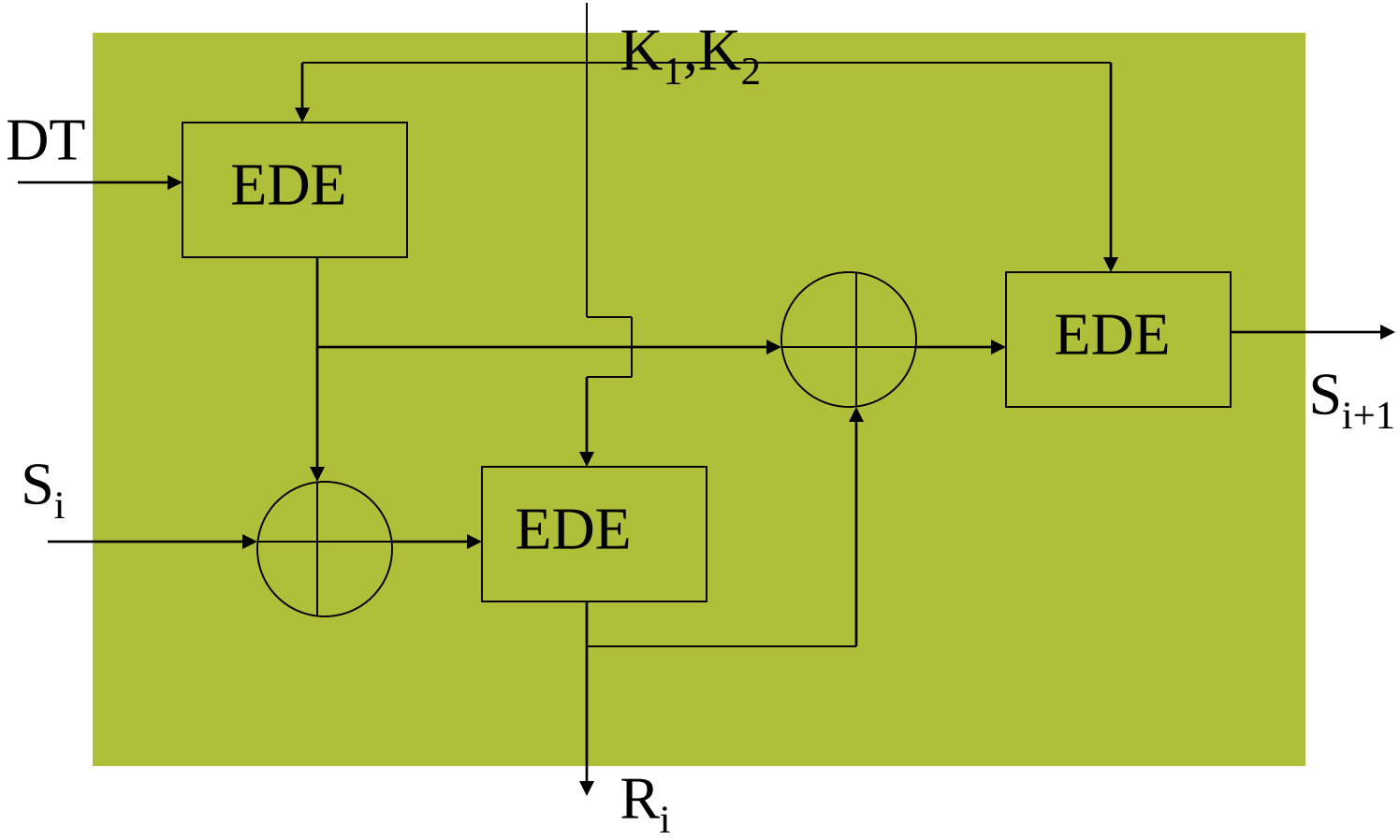
# Designs based on cryptographic primitives

- Designs based on cryptographic primitives
  - A secure block cipher can also be converted into a CSPRNG by running it in counter mode.
    - This is done by choosing an arbitrary key and encrypting a zero, then encrypting a 1, then encrypting a 2, etc. The counter can also be started at an arbitrary number other than zero. Obviously, the period will be $2^n$ for an *n*-bit block cipher; equally obviously, the initial values (i.e. key and 'plaintext') must not become known to an attacker lest, however good this CSPRNG construction might be otherwise, all security be lost.
- A cryptographically secure hash of a counter might also act as a good CSPRNG in some cases.
  - it is necessary that the initial value of this counter is random and secret. If the counter is a bignum, then CSPRNG could have an infinite period.

# DES Based Generator

- ANSI X9.17 PRNG (used by PGP,..)
  - Inputs: two pseudo-random inputs
    - one is a 64-bit representation of date and time
    - The other is 64-bit seed values
  - Keys: three 3DES encryptions using same keys
  - Output:
    - a 64-bit pseudorandom number and
    - A 64-bit seed value for next-round use

# ANSI X9.17



# Linear Congruential Generator

- Protocol
  - Let M be an integer and a, b less than M
  - Let k be number of bits of M
  - Integer *l* is between k+1 and M-1
  - Let $s_0$ be a seed less than M
  - Define $s_i = as_{i-1} + b \bmod M$
  - Then the *i*th random bit is $s_i \bmod 2$
  - It is not proved to be secure

# Parameter Setting

- Not all a, b are good and m should be large
- For example, m is a large prime number
- For fast computation, usually $m=2^{31}-1$
  - And b is set to 0 often
- For this m, there are less than 100 integers a
  - It generates all numbers less than m
  - The generated sequences appear to be random
- One such $a=7^{516807}$
  - Used in IBM 360 family of computers

# RSA Generator

- Protocol
  - Let p, q be two k/2 bits primes and define n=pq
  - Integer b:  gcd(b, $\varphi$(n))=1
  - Public: n, b; Private p,q
  - A seed $s_0$ with k bits
  - Sequence $s_{i+1}=s_i^b \bmod n$
  - Then the *i*th random bit is $s_i \bmod 2$
  - It is proved to be secure!

# BBS Generator

- Blum-Blum-Shub Generator
  - Let p, q be two k/2 bits primes and define n=pq
  - Here p=q=3 mod 4
    - this guarantees that each quadratic residue has one square root which is also a quadratic residue
  - gcd(φ(*p*-1), φ(*q*-1)) should be small
    - this makes the cycle length large.
  - Let QR(n) be all quadratic residues modulo n
  - Public: n; Private p,q
  - A seed $s_0$ with k bits from QR(n)
  - Sequence $s_{i+1}=s_i^2$ mod n
  - Then the *i*th random bit is $s_i$ mod 2

# Cont on BBS

- Provably "secure"
  - When the primes are chosen appropriately,
  - and O(log log *n*) bits of each $S_i$ are output,
  - then in the limit as *n* grows large, distinguishing the output bits from random will be at least as difficult as factoring *n*.
- However,
  - it's theoretically possible that a fast algorithm for factoring will someday be found, so BBS is not yet guaranteed to be secure.

# Discrete Logarithm Generator

- Protocol
  - Let p be a k-bit prime,
  - Let $\alpha$ be primitive element modulo p
  - A seed $s_0$ is any non-zero integer less than p
  - Define $s_{i+1} = \alpha^{s_i} \bmod p$
  - Then the *i*th random bit is
    - 1 if $s_i$ is larger than p/2
    - 0 if $s_i$ is less than p/2

# Standards

- A number of designs of CSPRNGs have been standardized. They can be found in:
  - FIPS 186-2
  - ANSI X9.17-1985 Appendix C
  - ANSI X9.31-1998 Appendix A.2.4
  - ANSI X9.62-1998 Annex A.4