Information Systems Security (SOEN321)

Dr. Amr Youssef

Concordia Institute for Information Systems Engineering (CIISE)
Concordia University
Montreal, Canada

youssef@ciise.concordia.ca

Topics covered

- Authentication Protocols
- Secret Sharing
- Commitments scheme
- Public Key Infrastructure
- Key Escrow

Authentication Protocols

Classification

- Password based
- Challenge response
- Zero knowledge (not covered)

Authentication

- **Entity authentication (identification)**: the process whereby one party is assured of the identity of a second party involved in a protocol an that the second has actually participated.
- **Data source authentication**: is represents an indication about the source of the data.

Authentication Using Fixed Passwords

- Prover authenticates to a verifier using a password.
- Require secure communication channels
- Total trust in verifier
- Passwords must be kept in encrypted password files or just digests of passwords are kept.
- Attacks
  - Replay of fixed passwords
  - Online exhaustive password search
  - Offline password-guessing and dictionary attacks
- Possible Variants: One time password

Lamport's One-Time Password

- Stronger authentication that password-based
- One-time setup:
  - A selects a value w, a hash function H(), and an integer t, computes $w_0 = H^t(w)$ and sends w_0 to B
  - B stores w_0
- Protocol: to identify to B
  - for the ith time, $1 \leq i \leq t$
  - A sends to B: A, i, $w_i = H^{t-i}(w)$
  - B checks $i = i_A$, $H(w_i) = w_{i-1}$
- if both holds, $i_A = i_{A+1}$

Challenge Response Protocols

- Goal: one entity authenticates to other entity proving the knowledge of a secret, ‘challenge’
- Time-variant parameters used to prevent replay attacks, provide uniqueness and timeliness : nounce (used only once)
- Three types:
  - Random numbers
  - Sequences
  - Timestamp

Challenge Response Protocols

- **Random numbers**
  - pseudo-random numbers that are unpredictable to an adversary
  - vulnerable to birthday attacks, use larger sample
  - must maintain state
  - do not prevent interleaving attacks (parallel sessions
- **Sequences**
  - serial number or counters
  - long-term state information must be maintained by both parties+ synchronization
- **Timestamp**
  - provides timeliness and detects forced delays
  - requires synchronized clocks.

Challenge Response Based on Symmetric Key

- Unilateral authentication, timestamp-based
  - A to B: $E_K(t_A, B)$
- Unilateral authentication, random-number-based
  - B to A: r_B
  - A to B: $E_K(r_B, B)$
- Mutual authentication, using random numbers
  - B to A: r_B
  - A to B: $E_K(r_A, r_B, B)$
  - B to A: $E_K(r_B, r_A)$

- PKI
- Secret Sharing
- Authentication Protocols

- Key Escrow
- Commitment schemes
- Zero Knowledge

Secret Sharing

Secret Sharing

- Example:
  - Share a treasure map between two people who don't quite trust each other. Split the map and make sure both pieces are needed to find the treasure. Give each person half of the map.
  - Overall, example of *secret sharing*
  - Generalization
    - Given a secret, s, would like n parties to share the secret so that the following properties hold:
      - 1. All n parties can get together and recover s
      - 2. Less than n parties cannot recover s
    - Map example, s is the map and the people are the parties that share the secret.
    - Split the secret into n pieces, s1, s2, ... sn and give a piece to each party
    - Special case of secret sharing called *secret splitting*

Secret Aharing (N out of N Scheme)

- Protocol Secret Splitting
  - Given a secret, M of length n
  - Given N persons who will share the secret (named p1, p2, ..., pn)
  - Generate random bit strings R{1}, R{2}, ... R{N-1} or length
  - Calculate P = M XOR R{1} XOR R{2} ... XOR R{N-1}
  - Destroy or hide M
  - Give P to p1
  - Give R{1} to p2
  - [...]
  - Give R{N-1} to pN
- Secret Splitters need not even know which one receives P and which one receive R's.
- Either way, M can only be constructed by XOR'ing back together the information given to every person

Example

- Secret Splitting
  - Like a one-time pad
  - Same degree of absolute security
    - Subject to bad random numbers and human weaknesses
- Example for two people: M = 1011
  - Generate secret shares by generating Random number = length to 1011
  - Random number, R = 1100
  - XOR of 1011 and 1100, P = 0111
  - So, destroy M and distribute R to one partner and P to the other
  - To recreate M then XOR back together R and P

Problems

- Note that secret splitting was vulnerable to the loss of one site
  - Or, one site is subverted
- If you wish to balance a desire to preserve a message with the need to keep the message secret, a “secret sharing” technique may be more appropriate
  - Talk about one involving Polynomials!
  - Note that this example is vastly scaled down for ease of typing :)

Solution

- (m,n) Threshold Scheme
  - A secret is divided into "n" pieces (called the shadows), such that combining any "m" of the shadows will reconstruct the original secret.
- We'll use Shamir's LaGrange Interpolating Polynomial Scheme as our example (scaled down!)

Shamir Secret Sharing Scheme

- Choose a (public) large polynomial “p” bigger than
  - the possible number of shadows
  - the size of the secret
  - other requirements for strength
  - all arithmetic will be “mod p”
- Generate an arbitrary polynomial of degree “m-1”
- Evaluate the polynomial at “n” different points to obtain the shadows “ki”
- Distribute the shadows and destroy M and all the polynomial coefficients

Example

- (n,3) Threshold scheme
- Form of our arbitrary polynomial
- m=3 so polynomial is degree 2
  - $F(x) = ax^2 + bx + M \pmod P$
- We must decide on a size for n - this is the number of shadows. The number of shadows is independent of the size of the polynomial

Example

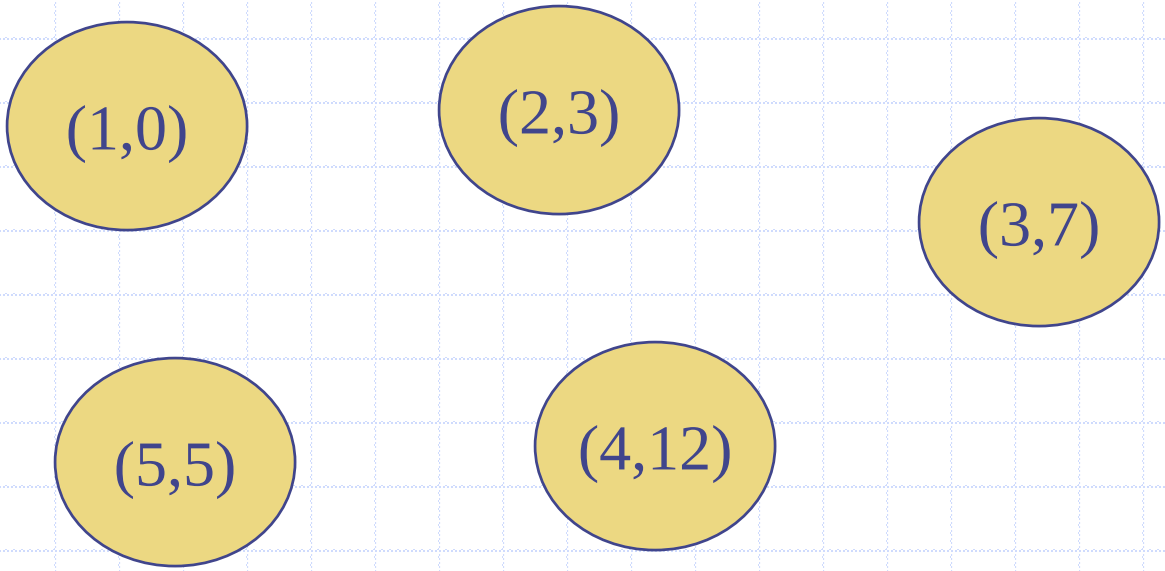
- Suppose we want a (5,3) scheme - that means we will have 5 shadows - for hiding message "11" (eleven)
- We choose a prime > 5, 11: say 13
- Our polynomial must be degree 2. We select the coefficients a, b randomly:

$$F(x) = 7x^2 + 8x + 11 \pmod{11}$$

- We must now generate five shadows. We decide to evaluate at points 1,2,3,4,5 (normally we'd mix them up!)
  - $F(x) = 7x^2 + 8x + 11 \pmod{13}$
  - $k1 = F(x1=1) = 7+8+11 = 0$
  - $k2 = F(x2=2) = \ldots = 3$
  - $k3 = F(x3=3) = \ldots = 7$
  - $k4 = F(x4=4) = \ldots = 12$
  - $k5 = F(x5=5) = \ldots = 5$

Example (Cont.)

- Shares Distribution
- We then put the shadows (the ki) somewhere. We need to either keep the selected value for x with the shadow or with the “coordinator”. Discard M, a, b.



Example (Cont.)

- We know that this is a (5,3) scheme, so the polynomial is known to be of degree 2:
- For instance, choose shadows k2, k3, k5

$$F(5) = A5^2 + B5 + M = 5$$

(5,5)

(2,3)

$$F(2) = A2^2 + B2 + M = 3$$

$$F(3) = A3^2 + B3 + M = 7$$

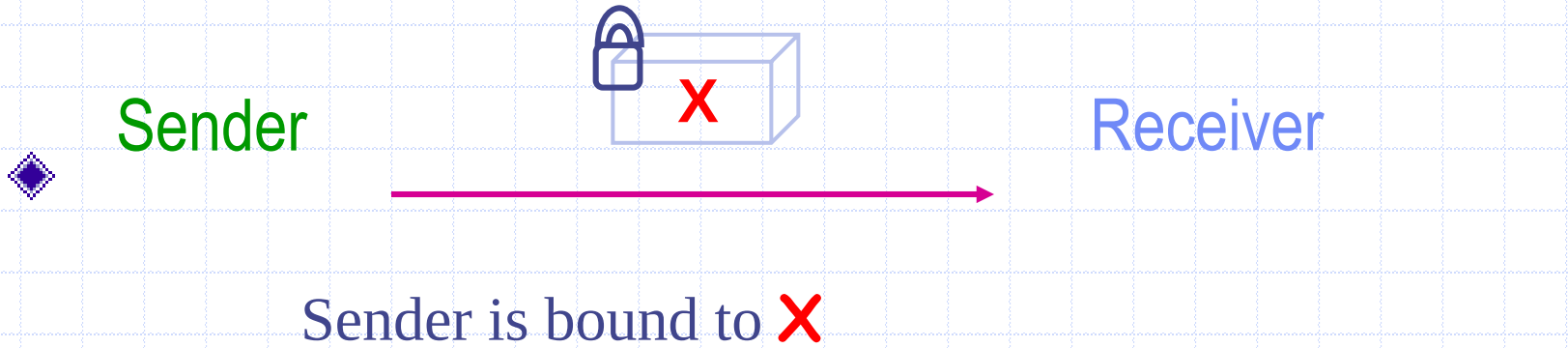
(3,7)

This gives us three equations and three unknowns, which is solvable, and yields A=7, B=8, M=11.

Commitments

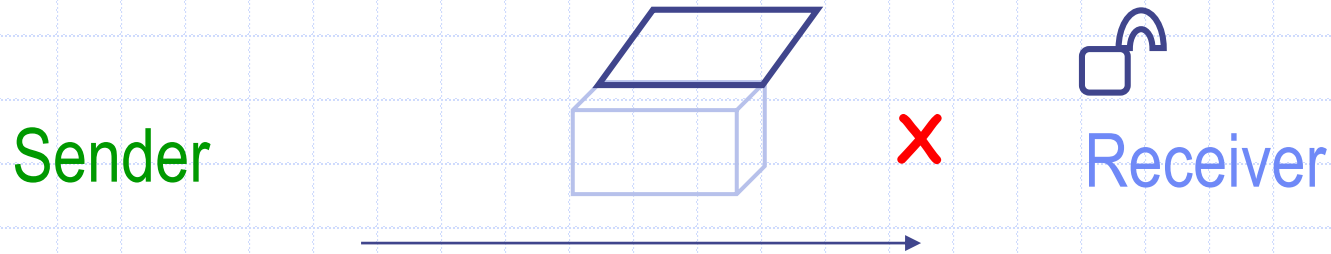
Commitment Protocol

Commit Phase



Sender is bound to X

Reveal Phase



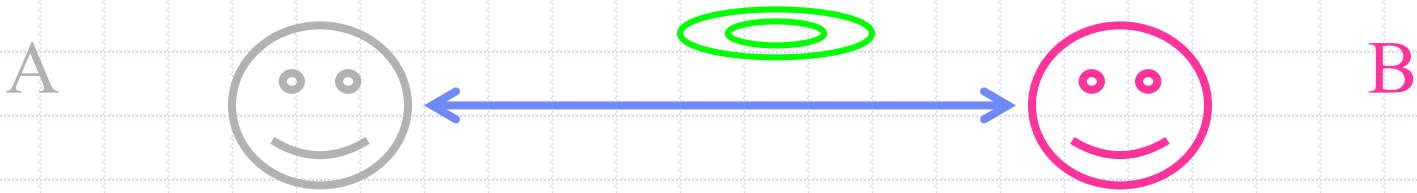
Receiver can verify X was the value in the box

Following the commitment phase

- Receiver should not have gained any information about **X** (what was inside the box)
  - Information theoretic?
  - Computationally?
- Sender should be bound to **X**
  - No two different and *valid* openings exist
  - It is computationally infeasible to find two different *valid* openings

Coin Flipping

- Two parties want to **agree** on a **random** value $\in_R \{0,1\}$
- Should be random even if one party cheats



Coin Flipping Protocol

- A selects $r_A \in_R \{0,1\}$;
  Commits to r_A
- B sends bit $r_B \in_R \{0,1\}$
- Coin is $r_A \oplus r_B$

If A doesn't open - result is in valid

If A's opening is invalid - result is invalid

Public Key Infrastructure

Who uses PKI?

- Web's HTTP and other protocols (SSL)
- VPN (PPTP, IPSec, L2TP…)
- Email (S/MIME, PGP)
- Good ID Smartcards (Certificates and Challenge/Response)
- Executables (Drivers, Authenticode)
- Copyright protection (DRM)
- Etc.

Summary of the problem

Public Key:P_A
Secret key: S_A

Public Key:P_B
Secret key: S_B

- **How are public keys stored**
- **How to obtain the public key?**
- **How does Bob know or 'trusts' that P_A is Alice's public key?**

Public Key Distribution Problem

- We just solved the problem of symmetric key distribution by using public/private keys
- But…
- Eve creates a keypair (private/public) and quickly tells the world that the public key he published belongs to Bob
- People send confidential stuff to Bob
- Bob does not have the private key to read them…
- Eve reads Bob's messages

- We need PKI to solve that problem

How to Verify a Public Key?

- Two approaches:
  - Before you use Bob's public key, call him or meet him and check that you have the right one
    - Fingerprint or hash of the key can be checked on the phone
  - Get someone you already trust to certify that the key really belongs to Bob
    - By checking for a trusted digital signature on the key
    - But there has to be one…
    - And you have to have friends to trust in first place…

Trust Models

- Web-of-Trust (PGP)
  - Peer-to-peer model
  - Individuals digitally sign each other keys
  - You would implicitly trust keys signed by some of your friends
- Trusted Authority + Path of Trust (CAs)
  - Everyone trusts the root Certificate Authority (e.g., Verisign)
  - CA digitally signs keys of anyone having checked their credentials by traditional methods
  - CA may even nominate others to be CAs – and you would trust them automatically, too

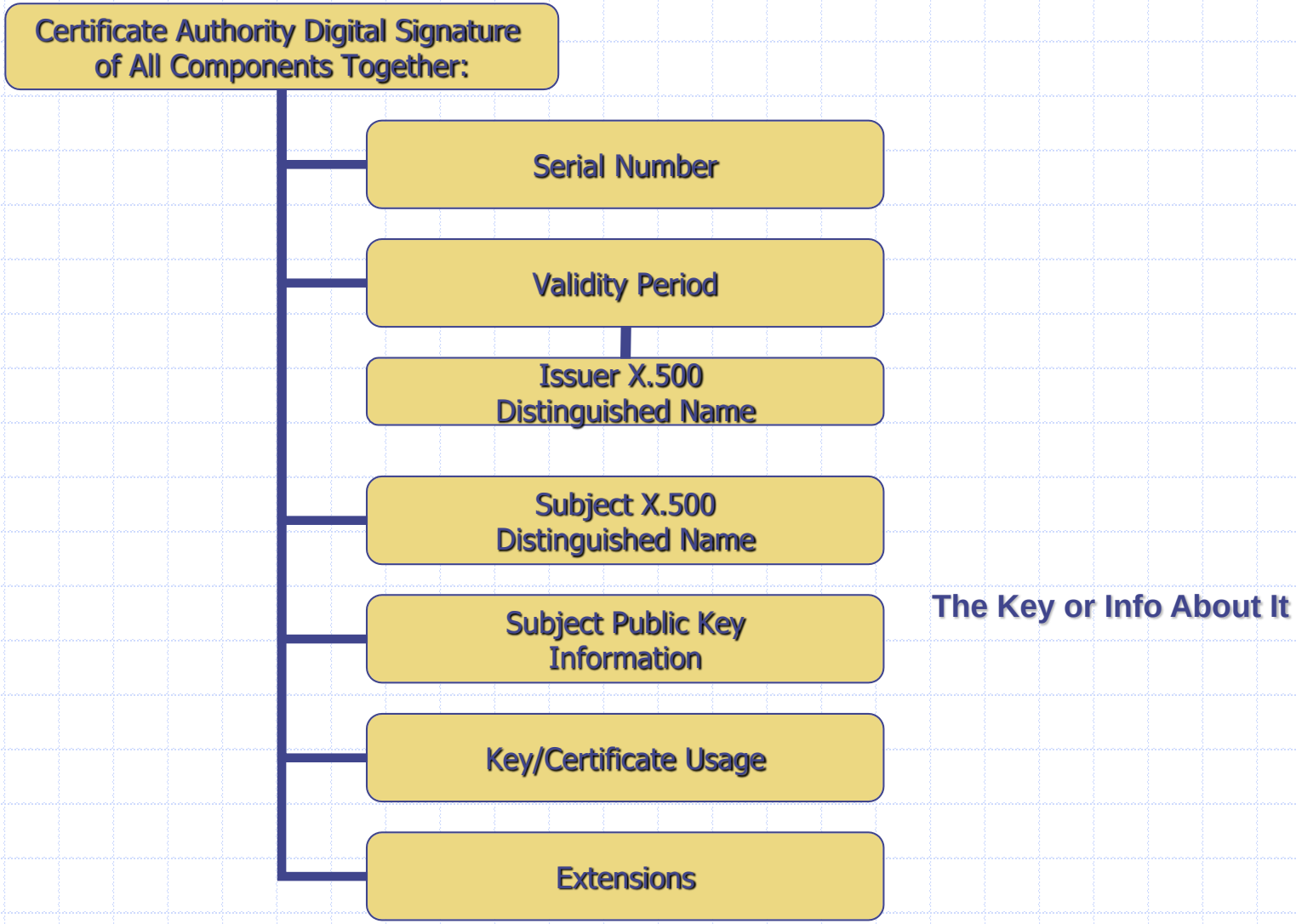
Issues

- Web-of-trust is more trustworthy
  - But it is time-consuming, requires lots of work and general public doesn't understand it
- CAs tend to be a little bit like a big brother as we all have to trust them implicitly
  - But it is a simpler model, easier to deploy and manage
- Combination strategy?
  - Let's trust a CA that verifies keys by traditional strong methods and peer-to-peer recommendations

Certificates

- The simplest certificate just contains:
  - Information about the entity that is being certified to own a public key
  - That public key
- And all of this is
  - Digitally signed by someone trusted (like your friend or a CA)

X.509 Certificate



Authentication with Certificates

- Alice gets Bobs certificate
- She verifies its digital signature
  - She can trust that the public key really belongs to Bob
  - But is it Bob standing if front of her, or is that Eve?
- Alice challenges Bob to encrypt for her a phrase etc. she just made up (*"I am Bob and not Eve"*)
- Bob has, of course, the private key that matches the certificate, so he responds
- Alice decrypts this with the public key she has in the certificate (which she trusts) and if it matches the phrase she challenged Bob with then it must really be Bob himself!
- That's the basic concept of how SSL works

Advantages

- Certificates are “safe”
  - No need to protect them too much, as they are digitally signed
  - Store anywhere, a file or a “dumb” memory-only smartcard
- Private keys that match the public key are extremely vulnerable (key assets)
  - You must protect them well
  - Store in “Protected Storage” on your OS or a “smart” smartcard that will have crypto functionality on board

Certification Hierarchy

- Most organisations do not use just one root key for signing certificates
  - Dangerous, if that one key is compromised
  - Does not scale to large organisations
  - Difficulty in managing responsibility
- Certificate Hierarchies
  - Start with CA root cert
  - Create more keys (e.g. for Microsoft etc.), sign with root key, mark as subordinate CAs
  - Create more levels in your organisation (for departments etc.)
- Validating a cert possibly involves validating a path of trust

Certificate Validation

- Essentially, this is just checking the digital signature
- But
- You may have to "walk the path" of all subordinate authorities until you reach the root
  - Unless you explicitly trust a subordinate CA

Certificate Revocation

- Keys get compromised, as a fact of life
- You or your CA issue a certificate revocation certificate
  - Must be signed by CA, of course
- And you do everything you can to let the world know that you issued it
- This is not easy
  - Certificate Revocation Lists (CRL) are used
  - They require that the process of cert validation actively checks the CRL and keep it up-to-date
  - There are some scalability issues
  - Many people disable this function
- That is why short expiration policies are important

Key Escrow

Policy Debate

- Export Control
  - PRO: It is desirable to keep strong encryption out of the hands of international terrorists and unfriendly governments.
  - CON: Since strong encryption is available and permitted in other countries, export restrictions reduce the ability of US companies to compete internationally.
- What about national security
  - Key Escrow

The Clipper Chip

- Each ***Clipper Chip*** has the following:
- Circuitry to implement SKIPJACK, an 80-bit private-key encryption algorithm developed by the NSA (classified as Secret)
- A 22-bit unique identification number (UID)
- An 80-bit family key (KF) common to all Clipper Chips.
- An 80-bit unique chip key (KU)
- For each chip, the manufacturer produces two additional keys KU1 and KU2 that can be combined to recover KU.
- Each additional key is combined with the chip's UID and given to an escrowing agency -- currently the National Institute of Standards and the Treasury Department

The protocol

- When two users wish to exchange encrypted messages, their chips establish an 80-bit shared session key (KS).
- The chip also generates a Law Enforcement Access Field (LEAF) that includes the UID, the session key encrypted using the chips unique key (KU) and a check sum, all encrypted using the family key (KF)
- With appropriate authorization (e.g., a court order), the escrowing agencies must relinquish KU1 and KU2 to the police.
- The police can then recover KU. From that and the LEAF, they can recover KS and "listen in" on the exchange (and any future exchanges that originate from the same chip).

- Lots of debate
  - Big Brother is watching you
  - Privacy concerns
  - Now it is a sort of *History (maybe!)*