Section 6: Advanced Cryptography

Ref: [Mark Stamp] Chapter 4
Objectives

- Introduce relevant examples for symmetric ciphers (block cipher: AES and stream cipher: RC4)
- Discuss the security of different encryption modes
- Learn how public key encryption methods such as knapsack and RSA work.
- Introduce PGP encryption tool and learn the usable security issues preventing end-users from securely using the tool for protecting emails.
Objectives

- To learn how Encryption Library API can be used for developing applications.
- To develop an understanding of Elliptic Curve Cryptography (ECC), Key exchange, and Public Key Infrastructure.
- To discuss hash functions.
- To discuss secure protocol (SSL) that employs cryptography.
Symmetric Key Cryptography
Review: the two basic types of Symmetric Key Ciphers

• Block Cipher
  - based on codebook model; a block of characters is encrypted at a time

• Stream Cipher
  - Encryption is performed on one bit (or one character) at a time

Stream cipher could be an approximation of one time pad. In one time pad the key stream is supp
Advanced Encryption Standard (AES) has open design in contrast with Data Encryption Standard (DES). AES was originally called Rijndael’s (rhine-dahl). 256bit AES is considered “virtually unbreakable“. Since higher key length usually means higher security, more rounds are performed for higher key lengths. Each round has four layers (ByteSub, ShiftRow, MixColumn, AddRoundKey).
AES defines a “state”, which is a 4x4 array of bytes. A 128 bit (16 byte) data block can be easily converted to 4x4 state and back.
SubBytes

- Each byte is replaced with a byte from a substitution box
- This provides non-linearity to the cipher

SubBytes introduces non-linear property to the cipher. Each byte in the state is considered as row number and column number of the lookup table. The data value in the lookup table for the row and column would be the output.
ShiftRows

- Figure shows the ShiftRows transformation where the first row remains unchanged, while the second, third, and fourth rows rotate by one, two, and three bytes respectively.

On the left side, for demonstration purposes, each column of the state is given a unique color. The right side shows the mixed output after “ShiftRows”.

Linear Mixing Layer
MixColumns

- Figure shows the MixColumns transformation where each column is treated as a four-term polynomial over Galois Field $GF(2^8)$ and multiplied by $a(x) = 3x^3 + x^2 + x + 2$. (Modulo $x^4 + 1$)

MixColumns mixes each byte in the column using an invertible transformation. The same is repeated for the other columns (not shown in the picture). This is also a non-linear layer.
In this stage, each bit of the state is XORed with each bit of the round key. Round Key is the key used for each round, and is generated from the master key using a key schedule. This is the level that is key dependent. Previous levels are done to remove any potential patterns from the input text, thus making the cryptanalysis difficult.
128 bit AES is sufficiently secure for most applications. AES also provides the option of 256 bit encryption. AES has been immune to linear and differential cryptanalysis.
i.e, $2^{80}$ is breakable, $2^{100}$ is not, at least for now. However, more attacks may lower the security of AES.
AES can provide sufficient security for a while. AES, however, doesn’t prevent Trudy (hacker) from changing the information or deleting it during its transmission or storage. Also, since AES is a symmetric key cipher, for secure communication each pair of users should have a unique key.
Symmetric block ciphers may perform encryption in different modes.

**Encryption Modes**

- Electronic Codebook Mode (ECB)
- Cipher Block Chaining (CBC)
In ECB mode, each input block is encrypted separately using the secret symmetric key, this “electronic codebook” computes the corresponding output block for the input block.
The weakness of this scheme is that the same input block always results in the same ciphertext block. This will make the cryptanalysis easier. Also, it is easier to launch replay attacks.
With all block encryption algorithms such as DES, AES, different modes of operations are possible. In Electronic Codebook mode, each block of input plaintext is independently converted to ciphertext. Thus, identical blocks of plaintext will result in identical blocks of ciphertext. From the picture it is clear that ECB encryption mode does not completely hide information, and it is still possible to infer things from it. Other modes of encryption such as Cipher Block Chaining (CBC) provide more secure encryption as evident from the right most picture.
In Cipher Block Chaining (CBC), sender and receiver must share the secret key and a random Initialization Vector (IV). For the first block, plaintext is XORed with the IV before encryption. For the second block onwards, each plaintext block is XORed with the ciphertext output of it’s previous block. This makes each new ciphertext block dependent on the previous block and “chains” them together.

Doing block encryption in CBC mode enhances confidentiality: identical plaintext blocks will produce different ciphertext blocks, which provides less information for cryptoanalysis.

The last ciphertext block in CBC mode can be used as a Message Authentication Code (MAC) for that entire plaintext message. MACs can be used in integrity checks. Examples are shown in later slides.
Because of the way CBC chaining is done, an error in one ciphertext block will affect the decryption of exactly 2 blocks: itself and the next block; then, if no more errors, rest of ciphertext blocks will decrypt correctly. Also, CBC mode makes replay of ciphertext blocks more difficult to do without detection.
Suppose there is a change in the $3^{rd}$ plaintext block. That would change the $3^{rd}$ ciphertext block, which would change the $4^{th}$ ciphertext block, which would change the $5^{th}$, ..., until second last block changes the last ciphertext block (the MAC). Similarly, changing any bit of the plaintext will change the MAC.
Using a MAC for File Integrity

- If a file is encrypted in CBC mode, the MAC for that file can be stored as a kind of “fingerprint” of that file.
- Whenever that file is accessed, its current MAC can be re-computed and compared to the stored MAC. Same? File is unchanged since fingerprint was stored. Different? Corruption detected.
- Some anti-virus programs use MACs for unauthorized change detection.

A cryptographic hash (discussed in later slide) of a file could be used in a similar way.
In the above system, the plaintext is NOT confidential since it was sent in the clear.

However, Bob can use the secret key he shares with Alice to encrypt the received plaintext, calculating that plaintext’s MAC himself. If his computation matches the received MAC he will know the plaintext he received was exactly the one Alice sent since any change in the plaintext in transit (deliberate or accidental) would make him compute a different MAC than the one received. Plaintext integrity is verified.

Non-repudiation is NOT achieved since Bob could also create a plaintext/MAC pair using the key he shares with Alice and then claim Alice sent it to him.
Unless you know the key, you won’t be able to produce the ciphertext. However, since both sender and receiver share the same secret key it is hard to prove which of them actually performed the encryption, so the ciphertext is repudiable. Non-repudiation property could be provided by using a trusted third party as long as the third party has some way to authenticate both sender and receiver. In this case, the sender sends the message to the third party and the third party delivers the message to the recipient.

Ciphertext can be corrupted (deliberately or accidently) while in transit. Corrupted ciphertext will be decrypted to incorrect plaintext.

By computing the MAC of the decrypted plaintext, receiver can check to see if it matches last block of received ciphertext. If it does, the decrypted plaintext must be exactly the same as the one sender encrypted and sent.
Software implementation of RC4 is efficient. RC4 has well documented security issues. However, if first 256 keystream bytes are discarded, security vulnerability can be mitigated. Note also that RC4 is a byte based stream cipher.
Perfect Secrecy

- $x \in M$ (x is a member of plaintext set M)
- $y \in C$ (y is a member of ciphertext set C)
- $p(x) = p(x|y)$

Perfect Secrecy is achieved if the fact that ciphertext is given does not make it any easier to find the plaintext.
Observation

• For perfect secrecy, after observing a given $y$, the variable $x$ can still be any member of the set $M$.

• Thus, there should be at least $|M|$ possible decryptions for $y$. There can be only one decryption per instance of the key (or the cipher won’t be reversible). i.e., $|K| \geq |M|$.
If the key is reused, stream ciphers are not provably secure.

Since $|K| < |M|$ (when the key is reused as in the context of the stream cipher), condition of perfect secrecy is not satisfied.
Three Perspectives

What algorithm should I use? I know the key size should be large.

This site is really slow!

How can I analyze the cipher?
Kerberos offers single sign-on authentication and a solution to the key distribution problem for symmetric key crypto. Kerberos has a Key Distribution Center (KDC) consisting of an Authentication Service (AS) and a Ticket Granting Service (TGS).

Every participating entity must initially register once with the KDC: at that time they must be authenticated and given a unique symmetric key (often derived from the hash of entity’s password) that will be used by that entity and the KDC to secure future authentications of that entity.

Once registered, whenever a client entity logs in, it requests a Ticket Granting Ticket (TGT) from the KDC. The KDC’s AS sends a reply encrypted with client’s registered key. Client decrypts reply to get a temporary key and the TGT. The TGT also contains that temporary key and is encrypted by KDC’s master key. Only the KDC can read the TGT. Until that temporary key expires, the client presents the TGT as proof to KDC that client is already authenticated and it allows KDC to communicate securely with client using the temporary key.
When client wants to use a server, the client’s workstation sends TGT back to the KDC and requests a ticket for obtaining the service. KDC creates a symmetric “session key” $K_{client, service}$ for the communication session between client and server. A ticket for the server is also generated and passed to the client along with $K_{client, service}$. The ticket for service is encrypted with the server’s unique pre-registered symmetric key. The client presents the ticket to the server. The server decrypts ticket to get client’s credentials and $K_{client, service}$ which it uses for communication with the client during that session. The ticket’s encryption is proof that it originally came from the KDC so the server can trust that KDC has authenticated client and provided the session key.
Active Learning Task

Discuss the security of the Kerberos model from the three points of view: user, administrator, attacker.

Single sign-on makes the system more convenient for user. User doesn’t need to remember multiple passwords for different servers.

User logs in once normally, her workstation takes care of all ticket and encryption details transparently.

KDC handles secure authentication for all servers registered with it. Registered keys are more secure: used only for login and for securing server tickets.

Temporary keys and session keys reduce available ciphertext encrypted with long term keys making cryptoanalysis harder.
Public Key Cryptography
Review some basic number theory needed to understand how RSA Public Key crypto works.
Prime Factorization

• Factors: A number may be written as the product of numbers (called factors)
  e.g., $30 = 3 \times 10$

defn: a number is prime if it is $> 1$ and is divisible only by itself and 1

• Prime factors: a number may be written as product of primes (called prime factors)
  e.g., $30 = 2 \times 3 \times 5$

Every number has a unique set of prime factors.
Relatively Prime Numbers

- Two numbers are relatively prime, if there are no common factors other than 1.
- In other words, two numbers are relatively prime if their greatest common divisor is 1.

e.g., \(\gcd(30, 77) = 1\) so 30 and 77 are relatively prime but 30 and 66 are not since \(\gcd(30, 66) = 6\)
Residues

• Residues: when doing arithmetic modulo $n$, the residues are 0, 1, 2, …, $n-1$ represented by $\mathbb{Z}_n$

• Reduced set of residues: set of those residues that are relatively prime to $n$. Represented by $\mathbb{Z}_n^*$

• Euler Totient function gives the number of elements in the reduced set of residues – represented by $\varnothing(n)$

For example:
$\mathbb{Z}_6 = \{1, 2, 3, 4, 5\}$
$\mathbb{Z}_6^* = \{1, 5\}$ Note: GCD(1, 6) = 1, GCD(5, 6) = 1

GCD(2, 6) = 2  GCD(4, 6) = 2  GCD(3, 6) = 3

Since there are two numbers in the set $\mathbb{Z}_6^*$, $\varnothing(6) = 2$.

For prime number, $p$, $\varnothing(p) = p-1$.
For two prime numbers $p$ and $q$, $\varnothing(pq) = (p-1) \times (q-1)$
Fermat’s Theorem

• if $p$ is a prime and $\gcd(m, p) = 1$, then
  \[ m^{(p-1)} \mod n = 1 \]
Euler’s theorem is the basis for an ingenious cipher called RSA.

Euler’s theorem says: If $\gcd(m, n) = 1$, then

$$m^{\varphi(n)} \mod n = 1$$

where $\varphi$ is Euler Totient Function.

Thus, if we can find two numbers $E$ and $D$ such that

$$E \cdot D = k\varphi(n) + 1$$

$$= 1 \mod \varphi(n)$$

then, $m^{ED} \mod n = m$

The above will be used later in the RSA public key cipher.
NP Complete Problems

• There is no known polynomial time solution for the problem (i.e., solution cannot be found quickly)
• However, any given solution to the problem can be verified in polynomial time (quickly).
In a public key crypto system, one key is used for encryption and a different key is used for decryption. Anyone who knows the encryption key can encrypt the message, but can’t decrypt the message unless they also know the decryption key. Thus, by keeping the decryption key private, confidentiality of message may be ensured.

- Security is based on infeasibility of computing Bob’s private key, given the knowledge of Bob’s public key.
- Confidentiality is assured since only Bob can decrypt it.

In a public key crypto system, one key is used for encryption and a different key is used for decryption. Anyone who knows the encryption key can encrypt the message, but can’t decrypt the message unless they also know the decryption key. Thus, by keeping the decryption key private, confidentiality of message may be ensured. Given someone’s encryption key (public key), it should be infeasible to compute the decryption key (private key). To prevent Trudy from quietly substituting Bob’s public key with his own, the public keys must be transported through a tamper proof channel.
One of the public key encryption schemes is based on Knapsack problem. Given a total weight, can you find a subset of weights such that their sum is equal to the total weight? This is the knapsack problem.

Why is the knapsack problem hard to solve? It is because the only known way to solve the problem is to try out different weight combinations. How many weight combinations are possible, if there are n weights? $2^n$. Thus, one has to try out $2^n$ possibilities.

While the general knapsack problem is hard to solve, a more specific case, super increasing weight knapsack can be easily solved.
Knapsack Problems

- General Knapsack is NP Complete
- A super increasing knapsack (where when weights are arranged in ascending order each weight is greater than the sum of the previous weights) on the other hand, is easy to solve

For example suppose the weights are: 1 4 6 13 25 52, they are super increasing because every weight is greater than the sum of previous weights. Now given 30, can we find a combination? Yes, that is easy. 1, 4, and 25. Can you get the sum 3? No.

If the weights are not super increasing, it is not as easy. Try with a general knapsack and see.
A super-increasing knapsack can be converted to general knapsack by multiplying with a multiplication factor modulo (say) 256. Thus, the general knapsack would be the public key. Given the sum it is hard to find out the individual weight – however, it is trivial to find the sum, given the weights. The conversion factor can be kept private. Thus, the receiver can convert the general knapsack into super increasing knapsack by simply multiplying with modular inverse of the multiplication factor. Once the super increasing knapsack is calculated, the message can be decrypted.
Another example of public key cryptography is RSA, named for its inventors: Rivest, Shamir, and Adelman. The public key is a number $E$, and another number $n$. The private key is a number $D$ used with the public number $n$.

For confidentiality, each block of message $M$ (considered as a number – combination of ASCII values of individual characters in the message), is encrypted to a Ciphertext block using equation 1. Use equation 2 for decryption.

For this to work $E$, $D$ and $n$ must meet certain conditions as explained in the next few slides.
The above example shows how RSA works. Obviously, the above won’t work for Ms greater than 10.
A number $n$ is computed from two large primes $p$ and $q$.

By picking $E$ and $D$ so that $E \cdot D \equiv 1 \pmod{\varphi(n)}$

Euler’s theorem guarantees that $M^E \mod n = M$

Such $E$ and $D$ can be the public and private key pair for that $n$ in RSA algorithm:

Encrypt $M$: $M^E \mod n = C$

Decrypt $C$: $C^D \mod n = M^{ED} \mod n = M$
If n is the product of two LARGE primes it is not very easy to find n’s prime factors. To find factors, one can use trial division, but it will take quite a long time.

Question: Given E and n the formula for computing corresponding D is known:

\[ D = E^{\varphi(n) - 1} \mod \varphi(n) \]

why can’t Trudy just use that formula to compute the private key D?

Answer: To use the above formula for D, Trudy must first calculate \( \varphi(n) \) but that requires that Trudy be able to prime factor n – which is infeasible when \( n = p \times q \) and p and q are large enough primes.

Note: The one who creates the key pair starts out with p and q so they know \( n = p \times q \) and are easily able to compute D for the given E.
Any RSA system must be implemented carefully to be secure. Some past implementations have been found to have weaknesses.

**Attacks on RSA implementations**

- **Forward search attack**: if small message space, attacker encrypts all possible messages with the public key, then can look up any intercepted ciphertext’s message. Mitigation? Pad messages with random bits before encryption.

- **Timing attack**: careful measurement of time decryption takes can help attacker determine the private key. Mitigation? Add random time delays.
In general, RSA requires raising a large base number to a large exponent. Many optimization tricks can be used to speed this up.

One way RSA can be optimized uses repeated squaring. For example, if you want to find $M^{16}$, you can find out $M^2$, then square it to get $M^4$ and square it again to get $M^8$, and then square it again to obtain $M^{16}$. This is easier than multiplying $M$ sixteen times to get $M^{16}$.

How do you find $M^{15}$ by repeated squaring? We cannot directly find $M^{15}$. But if we can find $M^8$, $M^4$, and $M^2$ by repeated squaring, $M^{15} = M^8 \times M^4 \times M^2 \times M$.

To simplify encryption, one can set $e=3$ for everyone. Thus, the encryption is just three multiplications. Of course, $p$, $q$, and $d$ need to be different for each user.
Bob’s needs only one public key, everyone can communicate confidentially with him using that key.

Public key systems still have a key distribution problem: how can someone who wants to talk to Bob get his public key and be sure that it is indeed Bob’s key?

- Only Bob can decrypt the Ciphertext.
- Anyone can send Bob a confidential message if they can get his public key.
For public and private key pair E and D, $M^E \mod n = M$ is equivalent to $M^D \mod n = M$

which means we can use either key to encrypt M and the other will decrypt the result.

If Alice uses her private key D to encrypt a message, anyone can use Alice’s certified public key E to decrypt it (so M is not private). A message encrypted with Alice’s private key is considered “digitally signed” by Alice since only Alice could have encrypted it.

If Bob receives C and uses Alice’s valid public key to decrypt it to M, he can be sure Alice sent M. Also, he can keep C as non-repudiable proof that Alice sent M to him since a 3rd party could also use Alice’s public key to verify $C^E \mod n = M$
Question

• How can we use RSA encryption to simultaneously ensure Confidentiality, Integrity and Non-repudiation of a communication between two parties?

It will take two encryptions at one end and two decryptions at the other end.
It is critical to verify a public key by examining its certificate. An intruder can switch the public key. If you use the wrong public key (i.e., intruder’s public key), the intruder can decrypt.

A self-signed certificate may not be acceptable for the other party. Why?
Certificates actually contain a lot of necessary detail. Since certificates are usually exchanged by browser’s on behalf of the users, users generally aren’t aware of the exchange and need not know the details. However, if the browser notices problems with a certificate (e.g. it doesn’t seem to belong to the server presenting it) browser may ask the user whether to accept it or not, allowing users to override security.

An X.509 certificate contains certificate version and serial number. It contains certificate issuer related information such as digital signature algorithms and parameters used, issuer’s name, certificate validity period. Then it contains subject information such as subject’s name and subject’s public key information: algorithm, parameters, the actual public key. Issuer ID number and Subject ID number may also be present. Optional extension fields may be added. Finally, the certification authority’s digital signature is added.
PGP (Pretty Good Privacy)

- Created by Phil Zimmermann
- Public key encryption
- Free versions available (PGP 5.0, GPG4Win)

PGP is downloadable from pgpi.org. You may want to download and try.

PGP:
http://www.pgpi.org

GPG
http://www.gnupg.org/
http://www.gpg4win.org/

Note that while it may be tempting to write your own encryption algorithms, it is often challenging to come up a secure algorithm. Further, its implementation should be secure and should be extensively tested before deployment. Thus, it makes sense to use already existing well reputed software for most cases.
Assignment

• Conduct Usability Study of PGPMail interface.
• Design a better interface
Sample GUI screens augmenting thunderbird with encryption/decryption capability. Cognitive walkthrough/low fidelity prototyping is performed in class in order to improve Usable Security.
When users are about to send unsecure email, an unmistakable warning is generated which is hard for users to ignore. It is not sufficient just to warn users, but also should offer to correct the problem. Thus, a “Make Secure” button is provided.
First, a simple but precise mental model of the system should be formed. Either the system design follows user’s mental model or helps users form a mental model. The former, although would be the best in terms of usability, the design can be quite complex, since the mental model should be valid across different individuals, cultures, and so on.

In this case, users are told that both sender and receiver should establish security certificates (to exchange public keys securely).
Set a strong password even if you need to write it down. Make sure you keep the written password confidential. You will need this password to read the secure email. Do not use this password on other systems.

This password will be used to help generate your key pair for your certificate.

Usable password schemes could be experimented for this screen. One possibility is to write it down and keep it in a safe place approach, which may not be the best possible method. The “Weak Password” Warning box will change to a submit button when a strong password is typed and both passwords match. If the passwords don’t match, warning will be generated when the mismatch occurs.
Both the sender and receiver need security certificates to be established before secure emails can be sent and received.

The certificates may be requested by email. Users may also be given a bail out option, but should be given several warning advising of the consequences of doing so.
Generate an email with an attachment – email may be customized if the users wishes so.

Hi Bob,

Please send me your certificate for me to communicate with you. My certificate is attached with this email.

Thanks,
Binto George
This is a security critical step. Both sender and receiver should confirm the fingerprints (of the public keys) by reading and verifying contents of both boxes. Users can verify the fingerprints over the phone or using SMS (not by email or chat!). All words should match. **This step is very critical for Security.**

This screen can be improved. How?
RSA encryption in Java
RSA Encryption: Key Steps

1. Generate a key pair (i.e., private key and public key)
2. Encrypt
3. Decrypt
Import necessary classes for the program

Initialization

Uses the following classes:
- import java.security.KeyPairGenerator;
- import java.security.KeyPair;
- import java.security.PrivateKey;
- import java.security.PublicKey;
- import javax.crypto.Cipher;
KeypairGenerator class is used for generating RSA keys. The generator is then initialized for generating 512 bytes keys. A keypair is actually generated by calling generateKeyPair() method. The generated key pair is stored in the object kp. cryptSpec is set as RSA. Crypt spec can also include character encoding and character filling schemes.

```
// Generate key pair
String cryptSpec = "RSA";
KeyPairGenerator kg =
KeyPairGenerator.getInstance(cryptSpec);
kg.initialize(512);
KeyPair kp = kg.generateKeyPair();
```
For encryption, in this example, public key is used. The message, “This is a test message” is converted to bytes because “Cipher” class operates on bytes. An instance of Cipher class is created and initialized for encryption using public key. Actual encryption is performed using doFinal() method. The resulting ciphertext bytes are stored in c.
Private key is generated and stored in a variable called `pvtKey`. Another cipher object is created and initialized for decryption using private key. Finally the decryption is performed by calling `doFinal()`. The resulting text is stored in `m`. 

```java
PrivateKey pvtkey = kp.getPrivate();
Cipher cipher2 = Cipher.getInstance(cryptSpec);
cipher2.init(Cipher.DECRYPT_MODE, pvtkey);
m=cipher2.doFinal(c);
```
Byte to String conversion

//convert bytes to string -- for printing
String s = new String(m);
System.out.println(s);

The above two lines convert the message bytes into a string. String is displayed.
Diffie-Hellman Key Exchange

- The "Diffie-Hellman Method For Key Agreement" allows two hosts to create and share a secret key for symmetric ciphers.

- First the hosts must get the "Diffie-Hellman parameters". A prime number, 'p' (larger than 2) and "base", 'g', an integer that is smaller than 'p'. They can either be hard coded or fetched from a server.

- The hosts each secretly generate a private number called 'x', which is less than "p - 1".

Diffie-Hellman key exchange protocol, lets two hosts communicate securely without any pre-agreement.
Two hosts independently compute their own public key \( y \) and then send it to the other. Both hosts can now compute the same secret key from the received public key as explained in the next slide.

**Diffie-Hellman (cont..)**

- Each host next generates their own public key, \('y'\) using their private key \( x \) with function:
  \[ y = g^x \mod p \]
- The two hosts exchange their public keys ('\( y'\)')
- Exchanged numbers are converted into a shared secret key, '\( z'\) by
  \[ z = y^x \mod p \]
  where \( y \) is the other host’s public key
  and \( x \) is their own private key
Diffie-Hellman (cont..)

- Mathematically, the two hosts should have generated the same value for 'z':
  \[ z = (g^{x_1} \mod p)^{x_2} \mod p = (g^{x_2} \mod p)^{x_1} \mod p \]
  (where \( x_1 \) and \( x_2 \) are the value of \( x \) generated at both sites)
  
  **All of these numbers are positive integers**

- 'z' can now be used as the key for whatever symmetric encryption method is used to transfer information between the two hosts.

The above expression shows how both hosts can compute the same secret key.
A few identified points above lie on the elliptical curve.

For example (1, 1) substituting x = 1

(1 + 2 *1 + 3) mod 5

Y^2 = 6 mod 5. So, Y = 1.

Since the point (1, 1) satisfies the EC equation, it lies on the curve.
DH can be done over ECC very easily. The slide explains how both Alice and Bob can come up with the same secret key.

**ECC Diffie Hellman**

- Public information: elliptic curve and a point, say \( a \)
- Both sender and receiver should select secret multipliers, say \( e \) and \( d \)
- Sender computes: \( a^e \); Receiver: \( a^d \)
- Products are exchanged
- Sender computes: \( a^d e \)
- Receiver computes: \( a^e d \), which are the same!
So public key crypto can be a solution to the symmetric key distribution problem. Instead, we have to find a way to reliably obtain someone’s public key.

Why can’t Alice just send a message to Bob asking him for his public key? What danger is there in sending his public key over an insecure channel? Consider how Trudy might intercept that message and substitute his own public key to launch a man-in-the-middle attack.

**Practical Use of Public Key Crypto**

- Symmetric key crypto is a lot faster than public key crypto.
- In practice, public key crypto is often used to share symmetric session keys which are then used to encrypt the rest of the session between two entities.
- Some sort of Public Key Infrastructure is needed so that any two entities can securely get each other’s public keys.
Unluckily, DH can be subject to man in the middle attack. How?

To avoid man in middle attacks we get public keys from certificates digitally signed by trusted authorities. PKIs are infrastructures that can supply certified public keys. Public keys are certified by well known certificate authorities (CA). A subject can verify the validity of certificate by verifying the CA’s signature on it. If a certificate has to be revoked, for some reason, it is placed in CRL. This way, someone can verify whether a certificate is still valid or revoked by CA.
How can trust be established in PKI? Two models above describe how trust can be established and verified.
CA signs the public key of the organization. Organization certifies department, department certifies employees.

For someone outside the organization to verify the signature of an employee, they have to trace the certificate all the way back to CA, whose public key is well-known.

An organization may obtain certificate from a well known CA and make the certificate available on its website. This way, a user’s browser can request the certificate and verify its validity and warn the user if the certificate appears to be invalid or expired.
Three Perspectives

I need to log on to the website. Let me ignore this silly warning.

Can I get a certificate issued in Microsoft's Name?

Users may be scared off by unnecessary warnings?
Hash Functions

Ref: [Mark Stamp] Chapter 5

One of the uses of hash functions is to check the integrity of data.
A good hash function should have certain properties.

**Properties of Hash Functions**

- **Compression** – a hash function takes variable size input and produces small fixed size output
- **Efficiency** – It must be easy to compute $h(x)$ for any input $x$
- **One-way** – it should be difficult to invert the hash
- **Collision resistance** – it is not easy to find two values that map into the same hash output
There are two types of collision resistance: weak and strong.

**Hash Collision Resistance**

- Weak collision resistance – given an input value it should be infeasible to find another input value that would map into the same output hash
- Strong collision resistance -- it should be infeasible to find any two values that map into the same output hash
A simple hash can be created by adding all bytes together modulo 256. However, this hash is prone to certain attacks.

For example, if need to send the number 56, 21, 200

- the hash value would be: \((56+21+200) \mod 256 = 21\)

#1 – the hash will not detect the change in sequence. 200, 21, 56 would produce the same hash

#2 – it is easy to find collisions, i.e., 50, 27, 200 would also hash to the value 21
CRC is another commonly used hash. A data value is divided by a specified value and remainder is calculated. For efficiency, subtractions are replaced by XOR.
Cryptographic Hashes

- A cryptographic hash is designed to make it infeasible to change a message without changing its hash value.
- Any deliberate or accidental change to a message should change its hash value so it can be used for message integrity checks.

Examples:

- MD – Message Digest, MD5
- SHA-1 – Secure Hash Algorithm (US Government Standard)
Tiger Hash

- Designed for 64 bit processors
- Three registers a, b, c has initial values:
  a = 0x0123456789ABCDEF
  b = 0xFEDCBA9876543210
  c = 0xF096A5B4C3B2E187

a, b, and c are 64 bit registers; they are initialized as above.
Message Block

- 512 byte blocks; padded as needed
- Each 512 byte block is divided into eight 64-bit words (x0, x1, x2, x3, x4, x5, x6, x7)

Message block is 512 bytes long; shorter messages are padded to makeup.
Hash algorithm

1. save_abc();
2. pass(a, b, c, 5);
3. key_schedule();
4. pass(c, a, b, 7);
5. key_schedule();
6. pass(b, c, a, 9);
7. feedforward();

The hash algorithm consists of three passes, with key_schedule between passes. Before the first pass, the registers a, b, and c are saved; the saved registers are used in the last step: feed_forward.
The slide shows the code for save_abc, which is pretty straight forward

save_abc

• The registers, a, b, c are saved into the back up registers

    aa = a;
    bb = b;
    cc = c;
Each pass has eight rounds.

pass

pass(a, b, c, mul)
{
    round(a, b, c, x0, mul);
    round(b, c, a, x1, mul);
    round(c, a, b, x2, mul);
    round(a, b, c, x3, mul);
    round(b, c, a, x4, mul);
    round(c, a, b, x5, mul);
    round(a, b, c, x6, mul);
    round(b, c, a, x7, mul);
}

In each round, registers a, b, c are altered as above. t1, t2, t3, t4 are substitution tables. They expand a byte into 64 bit substitution values. c register can be considered as an array of 8 bytes, represented as c_1, c_2, ..., c_7.
Feedforward step recalculates the register values as above.

- $a = a \text{ XOR } aa$;
- $b = b - bb$;
- $c = c + cc$;
Practical Use of Public Key Crypto

• Using public key crypto to digitally sign a long message is slow.
• In practice, public key crypto may be used to sign a cryptographic hash of the message rather than the message itself. Since the hash is small, the encryption takes less time and the signed hash is easier to store, transmit and verify.
A network connection is created over the protocol stack. Physical layer includes hardware such as Network Interface Cards (NIC), wires and cables. Link layer includes drivers that help communication over a link between two nodes. Network layer manages the network, making decisions on the best route for data packets over the network. Transport layer is responsible for creating and managing an end to end connection and using the layers below for communication. Application layer includes services that users can use for communication such as HTTP.
SSL layer is a layer between Application Layer and Transport Layer, which is responsible for data security. SSL layer makes this possible by a series of handshakes between peer SSL layer. SSL is probably a good example for usable security, since most of the security related functions are transparent to the user. When it comes to website authentication, however, the browser generates a warning which users may decide to override.

Reference: [Mark Stamp] Chapter 10
This slide takes a closer look at SSL (only the top two layers of the protocol stack is shown). Only most important steps are shown. Let us assume Bob’s computer runs a server and Alice uses a client to make a connection to the Server. Client authentication is optional. Client generates a premaster secret which is sent to the server. Server generates a master secret and sends it to the client. Now, both client and server generate the same session key, which is used for encrypting data.
A Usability Problem With SSL?

- SSL is designed to prevent man-in-the-middle attack
- If the server certificate is not valid, a warning page is displayed.
- Since users may not understand the risks, they are likely to override the warning in order to be able to continue, inadvertently breaking the security of SSL.

If the server certificate is invalid, a warning is generated. It is up to the user to heed the warning and take the appropriate action. Thus, the security of SSL to a great extent depends on user decisions.

The users have two choices:

- ignore the warning and get to their primary task
- listen to the warning and not to be able to do what they want to do

(The outcome is obvious)

Some times ignoring the warning is OK and sometimes it could be dangerous. How would the user know?