***1) What are the main security/performance trade-offs for symmetric and asymmetric cryptography to be deployed in practice? Could you give real-life scenarios where symmetric and asymmetric crypto could be more suitable?***

***Symmetric Key:***

**Pro:**

1) It is fairly secured.

2) It can be implemented at a hardware level which yields a fast  
encryption time.   
 3) It is a well known encryption algorithm.

**Con:**

1) Key distribution can be a problem because the secret key must be transported securely and can be hard to achieve if you have a large network.   
 2) Each user pair requires to have a unique key for the network communication. Thus, this would generate a significant number of key and can easily outweigh the sending information.   
 3) Each user in a pair with the same key can cheat on each other by changing another person information.

***Asymmetric Key:*** **Pro:**  
 1) Allow a sender and receiver to trade a private key on an unsecured channel.

2) Each user has a private key of his own and only he can read the information encrypted by his public key. This prevents the problem of a pair of user cheating on each other.

**Con:**  
 1) Cannot be implement at hardware level

2) Slower to encrypt/decrypt than symmetric key method

***Real-life scenario:***  
Since the **symmetric key** encryption can be implement on at a hardware level, it will prove a faster encryption and decryption time. Thus, it would suit better with a real-time or low power devices such as pacemaker (real-time), or a modern credit card (low power).

Though, **asymmetric key** encryption has its place in large network where symmetric key distribution can become challenging because first, there is no secure channel to exchange the key. Second, it number of generated key is going to be very large and its storage size can easily outweigh the size of the message.

***OTPs are highly secure, but why we do not see them much in practice?***

One Time Pad encryption not so practical because   
 1) The OTP requires true randomness which does not exist in the real world.  
 2) It is really difficult to distribute the OTP key in a large network because there is no secure channel that the key can be used to exchange.  
 3) The OTP can only be used once. Thus, not really practical in a large network where there is a lot of data being sent.  
 4) OTP encryption does not provide any sorts of message authentication. The message can easily be modified without receiver realizing it.

***Does Kerckhoffs's principle contradict with the “secret algorithm” practice in military systems? Given sufficient financial capability, how could you incorporate Kerckhoffs's principle into such high-end systems?***  
Kerchoff states that a crypto system should be secure even if the attacker knows all the details about the system with the exception of the secret key. This is counter-intuitive and contradicts with the military "secret algorithm" approach. Given an efficient fund, Kerckhoff's principle can be implemented by doing cipher (both block and stream).

2)

***What is the difference between a weak key, a semi-weak key and a possible weak key?***

**Weak keys** refers to a key that does not raise the level of security if used in an encryption algorithm. In DES, the weak keys are keys that contain either all ones or all zeros.

**Note:** DES generate a sub-key by taking in K from DES Key Schedule, permute the K, halving the key (Kl, Kr), permuting each half, resulting a new key Ki.

With the DES key generation algorithm, the keys with all ones and all zeros will always generate the same sub-key Ki (bit permutation has no effects on the key). This will result in an undesirable effects of having the DES encryption block to be the same as DES decryption block.

PS. There is a pair of Weak Keys in DES

**Semi-Weak Keys**

In DES, a **Semi-weak key** refers to a key which its encryption equivalent to decryption of another key; each key is a compliment of each other. Conceptually, a key with its KL and KR repeats the same pattern every 2 bits would generate a compliment to a key in DES. This implies that an encryption with a semi-weak key is equivalent to decryption with its complimentary pair.

PS. There are six pairs of Semi-weak keys in DES (excluded weak key)

**Possible Weak Keys**

A **Possible weak key** refers to a key which bit patterns of KL and KR has a repetitive pattern every 4 bits; it can only produce a unique key up to four rounds. This could introduce a potential exploitation.

***What is double DES? What kind of attack on double DES makes it useless?***

Double DES is an encryption method using two DES blocks: a plain text goes into the first DES block and gets encrypted. The output is then goes into the second DES block. At the first look, 2DES seems to be an efficient way to extend the key length of DES from 56 bits to 112 bits length.

**Note:**  The primary reason why DES is breakable because its key space, 256, is too small and feasible for a brute-force attack to break the encryption. By increasing the key length, the key space also increases which would impotent the brute-force attack.

Tragically, 2DES encryption is also broken regarding "Meet-In-The-Middle" attack. Meet-in-the-middle works in two phases:  
 Phase 1:

i) Input all value KL   
 ii) Encrypt a plain text X1  
 iii) Store the intermediate values Ci   
 ( Complexity: 256 encryption operation + 256 storage )

Phase 2:

i) Input all value KR  
 ii) Decrypt a ciphertext Y1 iii) Find a collision of an acquired text with the stored intermediate values from Phase 1  
 ( Complexity: 256 encryption operation)

Total complexity = (Complexity of Phase 1) + (Complexity of Phase 2)  
= 256 encryption operations + 256 storage +256 encryption operations  
= 257 encryption operations + 256 storage

**Notice:** This is barely any better than DES complexity of 256.

With phase 1 and phase 2 together, you will acquire a set of keys which could potentially be the original key that Alice and Bob uses. To extract a real key from the false-positive keys, you can feed in the keys and have DES encrypts and decrypts Alice's plain text. If it entirely matches with Alice's cipher text. The key is the original key.

***What is triple DES? How many keys are use in triple DES process?***

Triple DES is an encryption method that encrypts a plain text by putting the plain text through DES encryption block three times. Each of encryption blocks takes a 56-bit key input; total of 3 keys input.

3DES solves the problem of Meet-in-the-middle of 2DES considering its size of key space is too large to compute. To clarify, let's take a look at each phase of Meet-in-the-middle attack.

Meet-in-the-middle attack:

Phase 1   
 Complexity: 256 encryption operations + 256 storage   
 Searching through all of the possible keys in the first DES block and store the intermediate values still costs the same

Phase 2   
 Complexity: 256 encryption operation x 256 encryption operation   
 Decrypting Yi from the third block takes 256 encryption operations and to get from the second block to the first block takes another 256 encryption operations

Total Complexity = Phase 1 + Phase 2

= (256 encryption + 256 storage) + (256 encryption x 256 encryption)

= 256 encryption + 256 storage + 2112 encryption

~ 2112 encryption complexity

2112 complexity is not feasible to compute.

3)

***What is the basic design technique, which is frequently used to construct modern symmetric ciphers?***

According to Shannon, there are two basic design techniques that can be used to construct a strong block cipher: Confusion and Diffusion.

Confusion: a relationship between plain and cipher text is obscured   
 Operation: Substitution   
Diffusion: the influence on one bit of the input is spread over many cipher bits  
 Operation: Permutation

Given an example of AES block cipher. AES has multiple rounds of encryption. Each round has four basic layers including ByteSub, ShiftRow, MixCol, and KeyAdd.

**ByteSub** is the first layer in a round of AES, It provides confusion to AES block cipher. ByteSub is very similar to an S-Box layer within DES cipher block. The way ByteSub works is it takes the input, computes inverse finite field, GF(28), and uses Affine Mapping to get the output. Note: There are only 128 possible data input; it is feasible to compute all possible output and store the output in a table. At a high-level, ByteSub can sometimes be looked as a lookup table.

**ShiftRow** is the second layer in a round of AES. It provides mild diffusion to an AES cipher block. ShiftRow layer is merely just a layer of byte permutation or at the hardware level equivalent of swapping wires around.

**MixCol** is the thrid layer within a round of AES. It provides a strong diffusion using weighted matrix multiplication.

**KeyAdd** is the last layer of an AES round. The KeyAdd is there to XOR the key with a semi-cipher text from MixCol. This KeyAdd is used to prevent attacker from retrieving the sender key from a cipher text. The KeyAdd provides Key Whitening.

**Note:** The level of operation of AES is slightly different from level of operation of DES. In AES, everything is handled at byte level while in DES everything is handled in bit.

***What are the benefits of the use of finite field arithmetic in AES?***

In general mathematics, we dealt mostly with infinite set (Real numbers, Counting numbers, etc). To do cryptography, we need a finite set because it is easier to manage and as well preserves the arithmetic of the infinite set. The most basic finite set that has such properties is called Finite Field or Galois Field.

In AES, Galois Field is used to introduce a diffusion property throughout the AES block. The field is used in Byte Sub, MixCol, and KeyAdd. In ByteSub, Galois field's inversion operation is used to compute a substitute value for the output. In MixCol, multiplication in Galois field is used to introduce a strong diffusion property to the data. In KeyAdd, Galois field is used to introduce diffusion to key and the data.

**Note:** All of the arithmetic is valid finite field operation. Thus, the output from each operation will still be contained within the field.

*4)* ***We have discussed various Symmetric Encryption Modes. Each of these modes satisfies certain properties, which can be an advantage or disadvantage for a given application. Construct a table providing a summary information about each mode and its corresponding properties. For example, each row of the table will be properties (e.g., parallel operation, ciphertext manipulation, pre-computation, etc., please see course notes for more properties) and columns are Modes (e.g., CBC, CTR…). Each cell will take a value such as “Yes”, “No”, “partially”, “high”, “low” etc. according to given encryption mode and property.***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Mode | Parallelism | Pre- Compute | No Ciphertext Manipulate | No  Information Leak | Minimal Error Propagation | Error Correction Code |
| ECB | High | Low | Low | Low | High | High |
| CBC | High | Low | High | High | Low | Low |
| OFB | High | High | High | High | High | High |
| CFB | High | High | High | High | High | Low |
| CTR | High | High | High | High | High | High |
| IND-CPA | High | High | High | High | High | High |

***5)******“Ciphertext manipulability” is generally considered as an undesirable property for Encryption Modes. However, for modes that operates in stream cipher fashion (discussed in class), by design, it is possible to flip bits in plaintext by flipping bits of ciphertext.***

***Why is this possible?***

The Ciphertext manipulatability is an undesirable property which is introduced by either deterministic encryption algorithm or the lack of message authentication.

Case 1: No message authentication  
Without message authentication, there is no security. Thus, Oscar can manipulate intercept a ciphertext from Alice, manipulate it, send it to Bob.

Case2: Deterministic Encryption algorithm  
In ECB mode, Oscar can use Man-in-the-middle attack to do cipher text manipulation. A good example of this is an Electronic Funds Transfer problem. Here is how it works:  
Assumption:  
 - Bank A and Bank B use a simple transfer protocol that has 5 fields:  
 Bank A ID, Bank A acct number, Bank B ID, Bank B acct number, amount  
- Each field 1 to 5 has exactly n bits wide  
- Key KAB is fixed for some times.

Attacker:  
 1) Oscar opens one account at bank A, another at bank B  
 2) Oscar transfer repetitively a small amount of money from acct A to B  
 3) Oscar wiretaps and check for messages with identical cipher text block  
 4) Oscar stores a section of ciphertext that contains the destination account (which is Oscar's second account at Bank B).  
 5) Oscar now can swap the ciphertext section of his and other packages in the network (This means Oscar can hi-jack a package and have the money the money deposit to his Bank B account without actually breaking the block encryption)

**Note:** Ciphertext manipulabilityis not limited to the change of ciphertext content, but also the swap of ciphertext arrangement, drop the ciphertext**.**

***Is there a way to turn this (potentially) undesirable property into an advantage, describe how if there is one?***

In OFB a block cipher can be used to generate random numbers for a stream cipher. By the nature of stream cipher, the input message can be any sized, get encrypted, and transmitted at real time. Thus, the This can be beneficial if used in stream-oriented communication over noisy channel.

***Which extra cryptographic function (a group of functions discussed in the class) can be applied to the ciphertext so that the aforementioned advantage can be obtained without compromising the security?***

In order to be able to communicate over a noisy channel without sacrificing the security, you can use Digital signature and error detection protocol to keep track of the package and its authentication.

***6) Encryption (E) and Compression (C) are generally used together to achieve confidentiality and efficiency simultaneously. Given a message M, with which order function E and C must be applied? What are the reasons behind of this particular order?***

M --> Compress --> Encrypt --> Y

Before we get into why the Compression-Encryption has to be done in such order, we need to know the basic outline on how each of the operation works.

**Compression** takes a data input, find a pattern in the data, and use the pattern to do the size reduction.

**Encryption** takes a data input, turn the data into a high-entropy output.

Thus, compression has to be in before encryption because that way compression can recognize the pattern and acquire the size reduction. Once the compression is done, there is less data to be encrypted which would yield a faster encryption time.

On the other hand, if you do the encryption before the compression, the encryption process will destroy any recognizable patterns. And if the data got passed to compression block, it probably would not be able to recognize a pattern which would result on no reduction is memory size.

***7) In symmetric key cryptography, it is desirable to achieve both confidentiality and authentication (also provides integrity) of the data. These properties can be achieved via Encryption (E) and Authentication Functions (A), respectively. What is the correct order of these operations? Let's assume the specific notation and order of these operations are as follows: Authenticate-then-Encrypt (AtE) Encrypt-then-Authenticate (EtA) Encryption and Authentication (E&A) or the opposite way as (A&E) Discuss the security implications of these choices, which one is recommended and why? Mention important crypto papers (at least one, cite it), in which the security of an important real life protocol (Hint: the protocol that securely connects your VPN for each e-commerce transaction!) analyzed based on the above orders. Explain why this order matters a lot in practice? You may provide some discussions from these papers (please be brief, just hit on important points).***

**Encrypt-then-Authenticate (EtA): C = Enc(x), a = Auth(C), transmit (C, a): IPSec**EtA is the ideal method of Encrypt-MAC because it provides integrity of both plaintext and ciphertext by having an encryption algorithm encrypts a message, takes an encrypted message, authenticates it, and transmit both ciphertext and authentication signature. In this order of operations, MAC will both protect against a malleability of a weak cipher scheme and MAC itself will not provide any information regarding plaintext.

**Authenticate-then-Encrypt (AtE): a = Auth(x), C = Enc(x, a), transmit C: SSL**AtE is not the ideal method of Encrypt/MAC because it does not provide any integrity of a ciphertext (not until you decrypt the message). Although, it does provide plaintext integrity.

**Encryption and Authentication (E&A): C = Enc(x), a = Auth(x), transmit (C, a): SSH**

With the scheme of E&A, there is no integrity of ciphertext but there is integrity of plaintext. Though with the MAC directly interfere with plaintext, it expose the plaintext at risk of chosen-ciphertext attack. Moreover if the cipher block is malleable, the content of the plaintext can also easily be exposed or altered. By far, this Encryption and Authentication is the worst among all.

**Summary:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scheme | Integrity of  Plaintext | Integrity of Ciphertext | Malleable Cipher Resist | MAC Leaks Plaintext |
| EtA | Yes | Yes | Yes | No |
| AtE | Yes | Maybe | No | No |
| E&A / A&E | Yes | No | No | Maybe |

(Source: Krawczyk, Hugo. "The Order of Encryption and Authentication for Protecting Communications."  (n.d.): n. pag. International Association for Cryptologic Research. Web.)

(Source: http://crypto.stackexchange.com/questions/202/should-we-mac-then-encrypt-or-encrypt-then-mac)

***8) Given a modern cryptographic hash function (e.g., SHA) with m-bit length output, what is the maximum security it can provide in terms of m (lets call it x-bit security)? A generalized proof for any given m-bit hash function is available to show why it can achieve at best x-bit security. Please describe this generalized proof (related birthday attack concept) that simply connects m-bit to x-bit.***

In general, a hash function should have the following properties:

Resistance to **preimages**:   
 given z, it should be infeasible to find m such that h(x) = z. Generically, if the has function has an output of n bits, and we try to find the preimages, it would take us at least 2n (using brute force attack) which is proven to be ineffective.

Resistance to **second-preimages:** given m, it should be infeasible to find x' such that h(x) = h(x'). Using a brute force attack, it would take us 2n to find the second-preimages.

Resistance to **collisions:** it should be infeasible to find x and x' such that x ≠ x' and h(x) = h(x'). Using birthday attack approach, it would take us 2n/2 to find a collision of a hash function.

From the properties above, it seems to be the easiest to attack a hash function via finding its collision. To find a collision in a hash function, we can model a hash function in words of birthday problem and use the algorithm that solves birthday problem to find a collision of a hash function. This method of finding a collision is also known as "Birthday attack." Basically how the birthday attack works is random *q* points will be chosen from the hash domain, hash those points, and if it finds two different point that yields the same output, it has found a collision of a hash function. The total number of trial *q* need to find a collision is

However, we cannot apply the birthday algorithm to the collision of hash function directly because each output of a hash function is not equally likely to be the output.

To find the probability of how often an output is hashed, it can be calculated by

where D is the number of total hash

In order for a hash function to satisfy the equally likely of each output,

A hash function with the property is called to be *Regular*. If a hash function is regular, the number of trail until a collision happens is

Thus, this would yield the security of a SHA to be 2n/2.

**Source:**

**https://cseweb.ucsd.edu/~mihir/cse207/w-hash.pdf (page 7)  
http://security.stackexchange.com/questions/64049/sha-and-bits-of-security**

***9) What is the hash length extension attack? Please describe by giving some specific real-life examples.***

The length extension attack is an attack that would allow the attacker to generate a valid hash {secret || data || attacker\_controlled\_data} without knowing the "secret". The goal of an attacker is to be able to forge a valid hash with his information added to it. To do so, the attacker continues where the hash function left off and generate a signature using the information from the hash output.

From attacker side:  
 Known information: data, append, H, and H(secret||data)

Steps:  
 1) Create a new hash function (H') that uses the same algorithm as H  
 2) Initial its state to be the final state of H(secret || data) ~ AKA Signature  
 3) Let's H' calculate {secret || data || appending\_data}  
 4) Send the hash output to the server

The server once receives the hash, it will check the hash against the signature, H(secret || data). But since the attacker's hash is designed to match the original hash signature (with some extension), the server will then append the attacker data to the original package.

**Note:** An attacker did not actually have to solve the hash table in order to add a data to the original.

**https://blog.whitehatsec.com/hash-length-extension-attacks/**

**https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks**

**10) Properties of cryptographic hash function**

***What are the essential properties that a cryptographic hash function must satisfy?***

**Note:** Hash function is different from other crypto algorithm that it has no keys.

There are three essential properties which has function need to posses in order to be secure:  
 1) Preimage Resistance (or one-wayness)  
 2) Second Preimage Resistance (or Weak collision resistance)  
 3) Collision Resistance (or Strong collision resistance)  
 4) Arbitrary message size (hash can be applied to message of any size)  
 5) Fixed output length  
 6) Efficiency (hash value should be relatively easy to compute)

**Preimage Resistance or One-wayness**

Given a hash output z it must be computationally infeasible to find an input message x such that z = h(x).

An example if a hash function is not preimage resistance:  
Alice is encrypting a message but not signature, transmits the pair: ( ek(x), sigk pr, A(z))  
where ek() is a symmetric key cipher. Now Alice decided to use RSA digital signature which is computed as:

s = sigk pr,A (z) = zd mod n  
Well, Oscar can use Alice's public key to compute  
 se = z mod n  
If the hash function is not preimage resistance, Oscar can compute plaintext x from h-1(z) = x. Thus, digital signature along with symmetric encryption can expose a plaintext if the hash function is not preimage resistance.

**Second Preimage Resistnace or Weak Collision Resistance**

Given m, it should be infeasible to find x' such that h(x) = h(x').

As the main application of a hash function is in digital signature, it is very important for two different messages to be hash to two different value.

An example if a hash function is not second preimage resistance:  
Assuming Alice hashes and signs a message, x1. If Oscar is capable of finding a second message, x2, such that h(x1) = h(x2). He can substitute x2 for x1 and sends it to Bob. Bob which then verify the message to be true.

From the theoretical viewpoint, a weak collision will always exist in the hash function due to its infinite input with a fixed number of bit output. From the Pigeon hole principle, hash function will always inherent the weak collision in the output.

What we can do to make hash function weak collision resistance is to make finding the collided messages unfeasible.

**Collision Resistance or Strong Collision Resistance**

To achieve strong collision resistance is harder to implement than weak collision resistance because attacker has freedom to choose both of the message. But the same solution will always work: make it is unfeasible for the attacker to find the colliding messages.

***What is a Random Oracle and how does it play a role in the security proofs in general (what is its relation with cryptographic hash functions?).***

A random oracle is a magic block that would response to every unique input with a truly random response from its uniformed output. In practice, a random oracle would be an ideal replacement of a hash function. Tragically, the random oracle does not exist. In general, a random oracle is used to proof that the system is very secure because the attacker would require an impossible task to break the system.

**http://crypto.stackexchange.com/questions/879/what-is-the-random-oracle-model-and-why-is-it-controversial**

**http://en.wikipedia.org/wiki/Random\_oracle**

***11) Basic Security Models for encryption schemes***