|  |  |  |
| --- | --- | --- |
| **Confidentiality** | **Integrity** | **Authentication** |
| * **Classical Ciphers** * **Symmetric Encryption:** AES, Block Ciphers * **Asymmetric Encryption**: RSA, Diffie-H | * MAC * HMAC * Hash Functions | * MAC * HMAC * Hash Functions * Public Key Infrastructure |
| **Others** | | |
| * **Steganography**: Security by obscurity * **Salts:** Protect against pre-computation attacks (Dictionary / Rainbow Table) * **Nonce:** Protect against Replay Attacks – Old info can only be used once * **Session ID / Tokens**: Protect against Replay Attacks | | |

**PHYSICAL ATTACKS**

**Breaking physical security systems**

**Insider Attack**

**Rubber-Hose Cryptanalysis** is physically coercing or torturing someone to give up the key / secrets.

**Passive Recon**

* Listen to employee chatter by sitting in nearby café’s or their environment to learn their “language”

**Active Recon**

* Do as much research as possible on the building you are looking to break into.
  + Know all the building plans, security systems such as alarms, the types of locks used, cameras etc.
  + Know the incidence response plans for staff, police or other parties which can stop the attack.
* Know all employees by name, their families, where they live etc. to use it against them.
  + Know the rosters for staff, break times to determine windows of opportunity.

**Misdirection**

* Draw attention of people to a different area than your target area, before launching the attack.
* Disguise yourself as someone else who is supposed to be around the building / area you are targeting.
* **Security Theatre** is the practice of investing in countermeasures intended to provide a feeling of improved security while doing little or nothing to actually achieve security.
  + E.g. escalation of security at airports after the September 11 attacks.
  + Security theatre typically involves restricting / modifying aspects of people’s behaviour or surroundings in very visible / specific ways, which could involve restrictions of personal liberty and privacy.
  + It also encourages people to make uninformed, counterproductive political decisions. The feeling of safety can actually increase the risk. The disruption, cost and fear caused by Security Theatre acts as positive feedback to attackers who wish to exploit it (even if they fail to kill anyone, they cause huge economic damage)

**Human Weaknesses**

Trust, confirming bias (Horton’s Principle – when people confirm something which shouldn’t be the thing that they’re confirming), human error / making mistakes in general, can’t focus

**DIGITAL / NETWORK ATTACKS + PROTECTIONS**

**Breaking digital security systems**

**(HASH) Dictionary Attacks** is a technique for defeating a cipher based on trying all strings in a pre-arranged listing, typically derived from a list of words such as in a dictionary.

* Dictionary Attacks succeed because people choose short passwords that are ordinary words / variants.

**(HASH) Rainbow Table Attacks** is a pre-computed table cracking password hashes.

* Space/Time Trade-off: uses less computing power, but takes up more storage to perform attack.
* Databases usually store a cryptographic hash of a user’s password 🡪 no-one can determine the plaintext pw.
* When users login to the system and put in their password, the pw is hashed and compared to the stored hash.
* Rainbow Table Attacks store a pre-computed table of hashes,

**(HASH) Birthday Attacks** is a class of brute-force attack, relying on the Birthday Problem. Generally focuses on finding collisions in HF’s.

* Continuously change two different inputs (in general test different inputs), hash them both, then repeat until a new hash matches one of the previously computed hash where **h(x) = h(x’)**
* Has broken MD5 hash, Digital signatures

**Replay Attack** A network attack in which a valid data transmission is maliciously or fraudulently repeated or replayed to one or more parties who may process the data as legitimate. (An extended version of a MiTM attack)

* Attackers steals a copy of info from communication between two parties, then replays / reuses it later in a different context for malicious use such as duplicate transactions, circumventing authentication etc.

**Man-in-the-Middle** is a general term for attacks where the hacker is capturing traffic / info between two communicating parties.

* Attackers may try to eavesdrop / intercept data, tamper with messages, block messages etc.

**Length Extension Attack** is an attack where you just append additional information at the end of a hash + rehash the value.

* E.g. HASHVALUEgiveme9999dollars 🡪 You rehash this to get a new hash + send this new hash

**Side-Channel Attacks** is an attack based of information gained from the physical implementation of a cryptosystem, rather than from brute force or theoretical weaknesses in an algorithm.

* *Timing Attack, Power Consumption, Electromagnetic Leaks, Sound* can all sources of extra info which can be exploited.

**Reflection Attack** is a method of attacking a Challenge-Response Authentication system that uses the same protocol to authenticate both sides. The idea is to trick the target into providing the answer to its own challenge.

1. Attacker initiates a connection with the target
2. The target attempts to authenticate the attacker by sending it a challenge
3. The attacker opens another connection to the target + sends the target the same challenge masked as its own
4. The target responds to the attacker’s challenge
5. The attacker sends the same response back to the target on the original connection

**Denial of Service Attacks (DoS)**

The most effective DOS attacks are those that have a massive amplification factor.

* Attacks with amplification result in an attacker turning a small amount of bandwidth coming from a small number of machines into a massive traffic load / overwhelming number of packets hitting a victim from all over the world.

**Buffer Overflow**: Crashing an application from overflowing its memory / causing a segmentation fault

* Ping of Death: Sending a ping request that is larger than max size that IP allows, causing an overflow

**Smurf Attack:** Flood a target’s network by sending PING requests to a broadcast address, which broadcasts all messages to hosts connected to the subnet, then spoofing the return address as the victim’s address so they get flooded with replies from the hosts.

**SYN Attack**: An attacker sends a succession of SYN packets / requests to a target system in an attempt to fill up the server’s connection queues and thus denying the service to legitimate TCP users.

1. Attacker sends several SYN packets to the server.

2. The server sends a SYN-ACK packet to the attacker

3. The attacker ignores the SYN-ACK packet, leaving half-opened connections and consuming server resources.

4. Legitimate users try to connect to the server and send a SYN packet, but the server refuses to open a connection with them resulting in a Denial of Service.

**TCP - IP**

**TCP (Transmission Control Protocol)** is a protocol used when one computer wants to talk to another computer by sending it messages through lots of intermediate machines along the way.

* **Packets** are units of data that are of a certain size and can’t be bigger than x-kb size. If data being send is larger than a single-packet size, it will be split into multiple packets of data.
* **IP (Internet Protocol)** is the system where each packet is sent through a “postal system” with an address on it. Intermediate computers look at the address and passes on the packet until it reaches the destination.
* **TCP** sits on top of IP and creates a connection which means the two parties do not have to set up communication each time they want to send something to the other party.

**The Handshake (SYN, SYN-ACK, ACK) =** TCP sets up this connection by sending three packets

* + - * **SOURCE: S-Y-N packet** **(synchronise)** 🡪 “Hey, I want to initiate a connection with you”
      * **DEST: SYN + ACK packet** **(synchronise + acknowledge)** 🡪 “Okay I will connect with you”Sends both SYN and ACK)
      * **SOURCE: ACK packet** 🡪 When the ACK is sent back to the source, the connection is complete
* There is now a SYNCHRONISED STREAM OF PACKETS between the Source / Destination. Your packets will now be reassembled into the right order, rather than a random order w/ missing data filled in and resent.

**Sequence Number** is the main mechanism used to sync a stream of data + keep communication going.

* When the SRC talks to the DEST, it sends a sequence number and vice versa.
* Every time SRC sends a msg, they increase the sequence number 🡪 DEST will reassemble all packets in the right order
* The initial sequence number is usually random, however subsequence seq no’s increase by #bytes that have been sent in the last packet.
  + E.g. If starting sequence # = 0 and then SRC sends 100bytes to DEST 🡪 next sequence # = 100 and so on.
  + DEST can see what the sequence # is and if it doesn’t match, something has gone wrong / there may be some missing data and they will ask the SRC to resend the data.

**Sequence Number Prediction** is an attempt by attackers to predict sequences numbers being sent from the SENDER

1. Attacker tests a couple of connections with SRC to find a pattern OR listens in to conversations occurring between the 2  
   E.g. Predicts that the next S.N sent would be 100 bigger than the last one sent from the SRC 🡪 DEST
2. After predicting the next S.N, it becomes a race between the attacker / SRC to send the correct packet out.
3. The attacker can prevent the SRC from sending it first by launching a DoS attack on them.

**Session Hijacking** is an attack on a session between two parties to gain unauthorised access to info / services.

* Someone can hijack your connection / session if they jump in and persuade the server to send the next packet to the attacker themselves (by **Spoofing** = falsifying their IP address as the SRC IP + increase the sequence number  
  🡪 SRC will end up timing out since their S.N now mismatches but attacker will have picked up your session.  
  🡪 E.g. You are on Amazon, typing in credit card details and are about to buy something. Suddenly, Amazon stops responding. Someone has hijacked your session, has your credit cards, login and are using it to buy things.

**Problems with TCP/IP (The protocol is bad with Authentication)**

* Relying on an IP address / sequence number as an identity is NOT a good way of authentication + sequence numbers are also predictable

**Protection against some attacks**

**Salts** are a non-secret, random value generated to protect against pre-computation attacks such as Dictionary / Rainbow attacks

* Concatenated with a password, then hashed together. Hashed output + original salt is stored in the database.
* Even if an attacker knows all the salts and hash function used for a database, they need to check against every salt for each password to find the matching hash value to figure out the password.

**Nonces** are also non-secret, random value generated to be used only ONCE and protects against Replay attacks.

* Ensures that old communications can’t be re-used. Used in authentication protocols
* E.g. eCommerce: attacker could steal information from a victim and use it to make continuous orders. However, if the company receives any orders from the same person with the same nonce, it will discard those as invalid orders.

**Session ID / Session Tokens** is a piece of data used to identify a session (series of message exchanges). Prevents Replay attacks.

* They expire after a short period of pre-set time of inactivity or become invalid after a goal is met.
* Usually in the form of a hash generated by a hash function sent from a server to a client to identify a current session.
* **One-Time Passwords** are similar to session tokens, where the password expires after one use or after a very short time.
* **Time-stamping** is a method to identify when a certain event occurred. Prevents Replay attacks.

**Message Authentication Code (MAC)** is a string that is sent alongside a message, used to check Integrity + Authentication

* Work out a shared key *“HELLO”* 🡪 Add key to the msg *“HELLO” + “ WORLD”* 🡪 Function generates an output from it *“57P”*  
  🡪 Send msg w/ the hash *“WORLD”* + *“57P”* 🡪 Recipient adds shared key to msg, runs through func & compares w/ *“57P”*
* You can add another layer of security by encrypting the message, where the shared secret would be the decryption key.

**Keyed-Hash Message Auth Code (HMAC)** is a specific version of a MAC, using a Cryptographic Hash Function and Key.

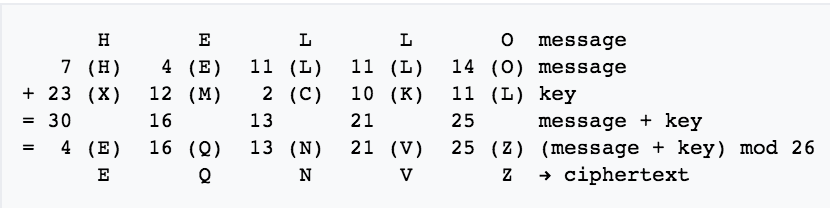
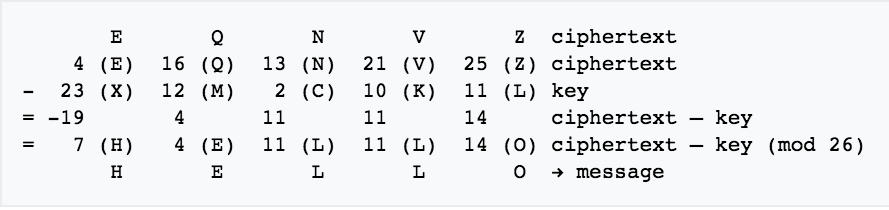
* Can use Cryptography Hash Functions such as MD5 / SHA family etc.
* Adding salts to the start AND the end to prevent Length Extension Attacks.

**CIPHERS + CRYPTANALYSIS**

**One-Time Pad (OTP)**

**One-Time Pad** is an encryption technique where each character in the plaintext is combined with a character / number from a chosen key on the pad.

* It has a **Perfect Secrecy** property = the encrypted string provides absolutely no info about the original message. (difficult to cryptanalyse / the probability distribution of possible plaintext is independent of the ciphertext)
* Example: We want to encrypt the plaintext “H E L L O”
* Two pads of paper containing identical keys are issued to both parties.  
  ^There is usually a rule to this: e.g. “use the 10th page on 1st Jan”
* The sequence of letters or numbers on the page will be the key for the message.
* Starting from the first letter / number, combine the key and message using **modular addition** (mod 26)
* Decoding is just using **modular subtraction**.

**Cryptanalysis methods: Crib-Dragging**

**Crib-Dragging** is a method of performing cryptanalysis against multi-time pads (incorrectly used One-Time Pads) where the same key is used for different messages.

* OTP: **message XOR key = cipher**
* XOR two ciphers together: **cipher1 XOR cipher2** **= (message1 XOR key) XOR (message2 XOR key)**

**= message1 XOR message2**

* Both keys are removed when ciphertexts are XOR’d with each other.

How to use Crib-Dragging:

1. **Guess a word**:  
   "the"
2. **Encode the word you chose**:  
   “the” 🡪 "746865"
3. **XOR the two cipher-texts**:  
   cipher1 XOR cipher2 = "3c0d094c1f523808000d09"
4. **XOR** **the chosen word AT EACH POSITION with the XOR of the two cipher-texts**:  
   | 3c0d094c1f523808000d09 |  
   | XOR  746865 |  
   |------------------------------------------------| 🡪 Converting “48656c” to ASCII = “Hel”  
   |          48656c |

Since this is readable text, we can assume “the” is in the 1st position of one message / “Hel” in the other

1. **If readable, guess + expand on the text “Hel”:**  
   “Help” / “Hello” ???  
   **XOR the expanded word to the same XOR of the two cipher-texxts**  
   |         3c0d094c1f523808000d09 |  
   | XOR  48656c6c6f |  
   |-----------------------------------------------| 🡪 Converting “7468652070” to ASCII = “the p”  
   |          7468652070 |
2. **If NOT readable, move up one position and XOR the chosen word again**
3. **Repeat steps until you work out the plaintext of both messages**.

**ENCRYPTION: SYMMETRIC CRYPTOGRAPHY**

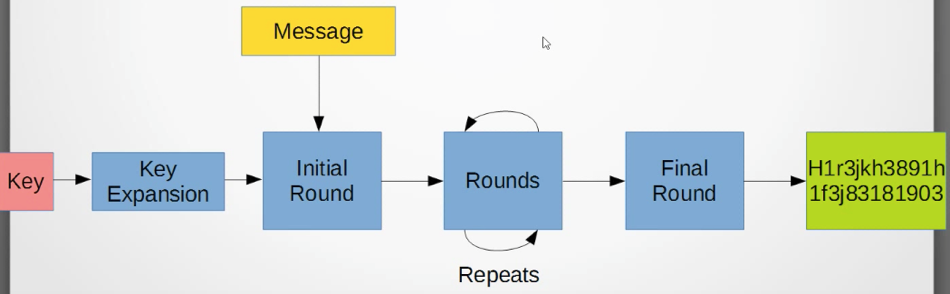
**Advanced Encryption Standard (AES) (Symmetric: same key to encrypt and decrypt)**

The **AES** is a BLOCK CIPHER that is now adopted worldwide. It succeeds the **Data Encryption Standard (DES)** which had issues with a relatively small 56-bit key and slow.

* Encrypts **128 | 192 | 256 bit blocks** at a time. **(3 options)**
* Each block is encrypted separately using the same algorithm.
* If a message is not divisible by the block length, then padding is added.

AES rounds

* AES is made up of a couple of initialisation steps (key expansion + initial round)
* Then a series of rounds of encryption are performed using the expanded key.
* The number of times we repeat the “rounds” of scrambling step is determined by the size of the key we select.
* The final round is simpler than the normal rounds.

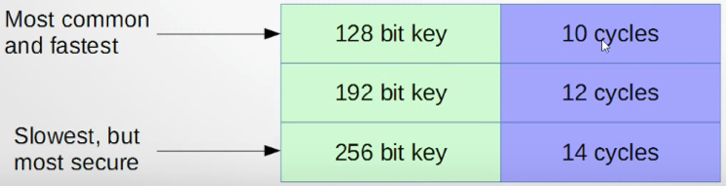


**Blue boxes**

**=  
Encryption process**

AES key lengths

* Key length options are: **127 bits | 192 bits | 256 bits**
* Size of the key dictates how many rounds of scrambling we have to perform.
* Larger keys = more rounds, more secure but slower encryption

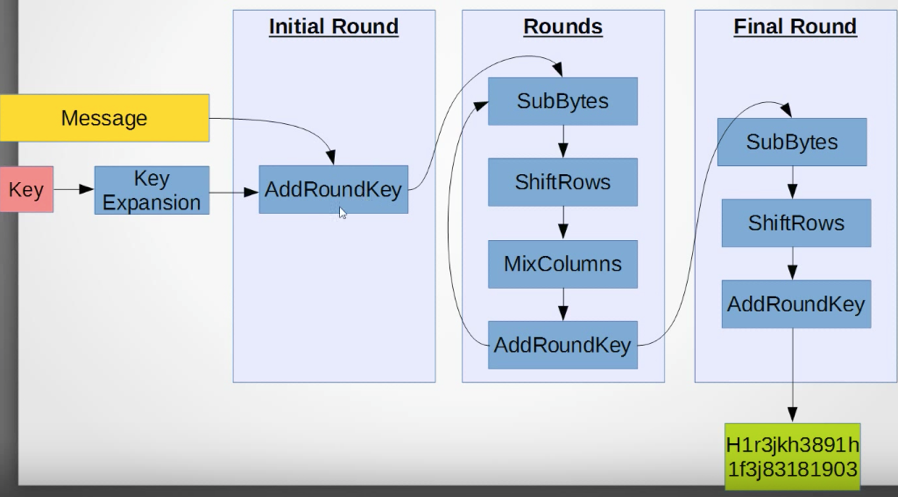


AES Key Expansion

* Each round we use a modified version of the original key = AES Key Expansion
* Original key is a seed to generate as many keys as we need for each round
  + I.e. 10 rounds of scrambling = 10 keys from the original seed key.
  + Each of these round keys are used to encrypt our data

Stages within Rounds

1. Key Expansion
2. Initial Round:
   1. AddRoundKey (XOR)
3. Rounds (10 / 12 / 14 rounds)
   1. SubBytes
   2. ShiftRows
   3. MixColumns
   4. AddRoundKey (back to (a) and repeats)
4. Final Round:
   1. SubBytes
   2. ShiftRows
   3. AddRoundKey



**Block Cipher Modes (Symmetric Key Algorithm)**

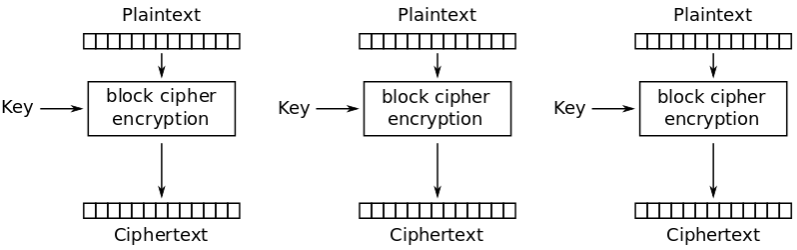
When encrypting large amounts of data using a cipher such as AES, the data in practice is usually much larger than the block size (128 | 192 | 256 bit) so it is split into blocks and encrypted, then combined together to form the cipher text.

**Block modes are the different ways to split up the data, encrypt and combine to form the cipher text.**

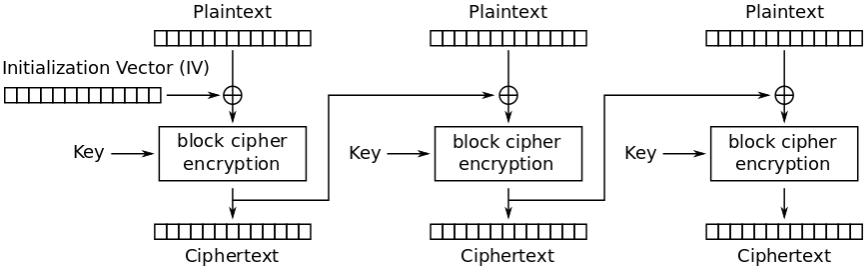
**EBC (Electronic CodeBook)**

Parallel Encryption  
1. Break message into blocks  
2. Add padding if necessary  
3. Encrypt each block separately  
4. Concatenate

Problems:  
Poor entropy  
Identical plaintext = identical ciphertext



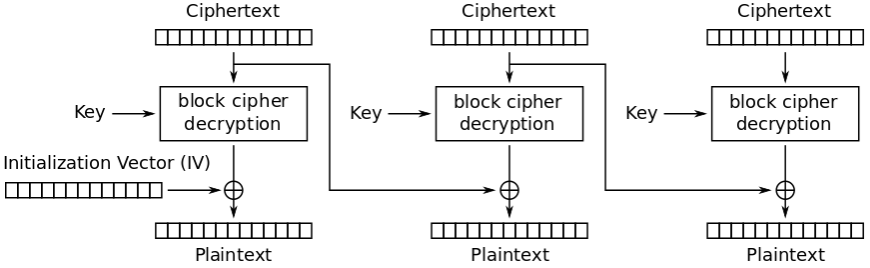
**CBC (Cipher Block Chain)**

****

Solves the identical output problem

1. Initialisation Vector **XOR** Plaintext1  
2. Encrypt block1 = Ciphertext1  
3. Ciphertext1 **XOR** Plaintext2

4. Encrypt block2 = Ciphertext2  
5. Repeat #3 onwards for each block

****

Parallel Decryption

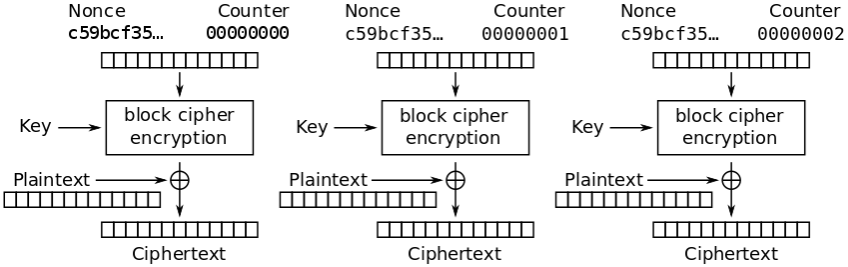
1. Apply the key to each ciphertext block.

2. For the first block:  
InitVector **XOR** FirstCipher = Plaintext  
3. For all else:  
CurrCipher **XOR** PrevCipher = Plaintext

**CTR (Counter Mode)**

Parallel Encryption / Decryption  
1. Choose an initial starting value for the counter  
2. Encrypt the counter using the key = Nonce  
3. Nonce **XOR** Plaintext Block = Cipher-text Block

4. Increment the counter (usually by +1)  
5. Repeat #2 and onwards until all blocks encrypted  
Decryption = reverse process



**Block Cipher Vulnerability: PADDING ORACLE ATTACKS**

It is an attack which is performed using the PADDING of a cryptographic message.

* Attacker intercepts a ciphertxt, submits it to the server to learn if the last bytes of the decrypted ciphertxt are valid padding
  + The “oracle” / server returns either a: PADDING ERROR / MAC ERROR
* From this information, the attacker can use it to completely decrypt the ciphertext.
* Attacker needs an “Oracle” / some program to take in cipher-texts and will return whether the plain-text.

**Block Cipher Question**

The ECB ones have repetitions in the cipher text. So if you put in a huge string like aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa, the ciphertext would have four blocks different blocks, that would just repeat.

The CBC ones would pad out when you only entered a really short string. so 'a' would still give a default block size for the cipher text.

The CTR ones had a cipher that was in direct proportion to the length of the plain text.

I used a huge string of 0's (000000000000000000...) The way I saw it was a change in one bit it caused all the following bits to change in the CBC ones. In the ECB ones you got repetitions so it was easy to spot. And in the CTR ones there were no repetitions but if you changed one bit in the plain text you only got a corresponding one bit change in the cipher text.

**ENCRYPTION: ASYMMETRIC CRYPTOGRAPHY**

**Asymmetric / Public Key Cryptography** is any crypto-system that generates a pair of keys:

1. **Public Key**: given out to the public and used to ENCRYPT a message.
2. **Private Key**: known only to the owner and used to DECRYPT & READ the message.

Everyone keeps their Private Key / Reading Key secret, but they all share a Public Key / Writing Key.

**Diffie-Hellman Key Exchange**

The **Diffie-Hellman Key Exchange** is a method of securely exchanging cryptographic keys over a public channel.

It is based off the **Discrete Log Problem** = difficulty of finding the factors of the product of two prime numbers.

**Gx % M = S shared key: S private key: X**

* Alice and Bob agree to use a **Primitive Mod M = 23** and a **Base G = 5**
* Alice chooses a **Private Key X = 6** then calculates a public key **A = 56 % 23** 🡪 Sends **A = 8** to Bob
* Bob chooses a **Private Key X = 15** then calculates a public key **B = 515 % 23** 🡪 Sends **B = 19** to Alice
* Alice computes the **Shared Key S** (using Bob’s public key): **196 % 23 = 2**
* Bob computes the **Shared Key S** (using Alice’s public key)**: 815 % 23 = 2**

This shared key = 2 would be used to send messages to each other.

Even if a spy can see the shared secret, it would be difficult to find the private key X.

In practice, the protocol would be combined with a form of AUTHENTICATION to prevent MiTM.

**Vulnerability of Diffie-Hellman** **Key Exchange** = **Man-In-The-Middle**

* A spy can intercept the Shared Secret between Alice and Bob + send both parties their own Fake Shared Secret
* Any message now received can be decrypted by using the Fake Shared Secret.
* The main problem is that Diffie-Hellman doesn’t AUTHENTICATE the participants of the exchange.

**RSA**

**RSA** is a public-key cryptographic algorithm.

It is based off the **Factoring Problem** = difficulty of finding the factors of the product of two prime numbers.

1. Choose two distinct prime numbers **p = 3** and **q = 11**
2. Compute **n = pq** (n will be used as the mod for both private and public keys)  
   **n = 33**
3. Compute **phi(n) = (p – 1) (q – 1)**  
   **phi(n) = 2 \* 10 = 20**
4. Choose an **e** such that **1 < e < phi(n)** ande / phi(N)are coprime (gcd(e,phi(n)) = 1) **e = 7**
5. Compute the secret exponent **d** such that **d \* e % phi(n) = 1** (as 7\*3 = 21 % 20 = 1) **d = 3**

**Public Key is (e, n) = (7, 33) Private Key is (d, n) = (3, 33)**

Suppose there is a plaintext m = 2.

* **USING PUBLIC KEY (e, n):** Encrypting plaintext m 🡪 cipher-text c  
  **c = 27 % 33 = 29**
* **USING PRIVATE KEY (d, n):** Decrypting cipher-text c 🡪 plaintext m **m = 293 % 33 = 2**

**Merkle Puzzles**

**Merkle Puzzles** were an early attempt at establishing a protocol for a public key cryptosystem.

* It works by one party generating a large number of puzzles such that each puzzle is
* Method:
  + Merkle writes a series of notes. Each note has a PUZZLE + KEY + IDENTIFIER
  + Merkle publishes all the notes it on the internet, so the public can see them.
  + We want to talk to Merkle, so we pick one of the notes and break the puzzle to obtain the IDENTIFIER + KEY
  + We encrypt our intended message, then pass our CIPHERTEXT + IDENTIFIER to Merkle
  + Merkle knows the PUZZLE/KEY/IDENTIFIER on each note, so he goes to the correct note to grab the KEY
  + This KEY = the PUBLIC KEY and can be used to decrypt all messages now.
* An attacker who intercepts the message has to crack the code on every piece of paper until they find an identifier corresponding to the one the two parties are using.
* On average, they have to look at HALF the notes.

**Merkle Puzzle Example QN:**

* Alice wishes to establish a shared secret with Bob using a Merkle Puzzle system. If she uses 10,000 puzzles that are individually crackable in 2 hours, how long would it take for an attacker Eve to find the shared key?
  + For Bob, it would only take 2 hours to get the shared key.
  + For Eve, she would have to look at HALF the notes on average to get the shared key.  
    10,000 / 2 = 5,000 🡪 5,000 \* 2 hours = 10,000 hours.
* For M puzzles, each crackable in N time, **key generation = (m\*n) /2**

**Differences between Hashing and Encryption**

**1. Non-Reversable**

* A bad hash is reversible

**2. Fixed-Length**

* Encrypting a sentence 🡪 output book is ~length of sentence
* Encrypting a book 🡪 output length is ~length of book
* However, hash output is always the same length
  + E.g. 128bit | 64bit depending on the hashing algorithm

**3. Non-keyed**

* Encryption always has a key
* Hashes don’t have keys built into them.
* **Hash(document + key)** 🡪 key can be appended to the input message then hashed  
  VS
* **AES(document, key)** 🡪 takes in key as a parameter to be used in the encryption process

**HASHING / CRYPTOGRPAHIC HASH FUNCTION**

A **Hash Function** is a function that takes in a variable sized string of data **(a pre-image)** and maps it to a unique data packet of fixed size **(Hash-Value)**. Hash functions are typically only used as an INTEGRITY + AUTHENTICATION TOOL.

**Properties of a cryptographic hash function**

**1. Pre-image resistance (CAN’T FIND X, GIVEN Y)**

* Given an output y, it is computationally infeasible to find an x such that h(x) = y
* Given the hash, you can’t reverse it to find the input.

**2. Second pre-image resistance (CAN’T FIND X’, GIVEN X)**

* Given an input x, it is computationally infeasible to find any SECOND INPUT x’ such that h(x) = h(x’)  
  (Given a document, it is infeasible to find a second document that has the same hash output)
* Important because hashes are used in MAC’s (Message Authentication Codes) which give us **integrity / auth**
* If two documents can hash to the same output, then someone can just forge or falsify a document and send it to a victim, pretending it is the real thing.

**3. Collision resistance (a more generalised case of second pre-image)**

* Not given any input / in general, it is computationally infeasible to find any two distinct inputs x / x’ that hash to the same output.
* Because the size of hash outputs is fixed (128 or 64 bit), there are bound to be some duplicate inputs, as #inputs can be infinite. A good hash function will produce an output that is as evenly distributed as possible across all hash possibilities.
* A bad hash function may have a bias towards a specific hash output, therefore having a greater chance for collisions.

**High-Resistance Hash Function:** If your attacker has to use **Brute-Force** to solve the problem.

**Low-Resistance Hash Function:** If your attackers show any way to do it faster than Brute-Force.

**Pre-image attack** is where an attacker tries to find an input/message that outputs a specific hash value.

**Second Pre-Image attack** is an attacker tries to find a secondary input, such that it equals the same H.V as the original H.V

**Birthday attack** is where an attacker tries many random / pseudorandom inputs until a hash value is found more than once.

* Based off the Birthday Problem and the Pigeonhole Principle.

**Random Mapping**: A small change in the input (1bit) should ideally change on average half of the output.

**Cryptographic Hash Functions**

**MD5:** A widely used algorithm / hash function producing a 128bit hash value. (but with 56bits of security which is too small)

* Problems:
  + Hash size is too small, so anything protected with that hash is subject to to attacks.
  + E.g. Collision attacks can find collisions within seconds using a Pentium 4 processor (224.1 power)
  + #bits shaved off because of some weaknesses 🡪 #bits is not good enough 🡪 Easy to do a Birthday Attack on it to break it
* Not-collision resistant + Not suitable for SSL Certificates / Digital Signatures
* Say you’re given an MD5 hash (128bits) and someone asks you to find the thing that hashed this, you got no chance
  + Using an MD5 hash for your passwords is actually okay, its not bad to use it, because it is still hard to extract your password.
  + Birthday attack was used against some Certificate Authority who were using MD5 hashes for their SSL Certificates and a research company was able to generate a fake SSL Certificate because they found that they were able to do a birthday attack

**SHA1:**

* **Takes any length of string and outputs 160 bit value**
* Have a loop which takes 512 bit blocks of data, one at a time until the file is expended.
* Problem: Too small, already been broken. Collision has been found.

**SHA2:** Works same way as SHA1 but is stronger / generates a longer key

* **Takes any length of string and outputs 256 | 512 bit value**
* Problem: Small, but hasn’t been broken yet.

**SHA3**:

* Good because it goes up to very large sizes
* No known attacks on it yet

**AUTHENTICATION, TRUST AND PUBLIC KEY DISTRIBUTION**

**Non-repudiation**: You can’t claim that something didn’t happen when it did happen. E.g. signing contracts.

* Use asymmetric keys to prove authentication.
* Use **Digital Signatures** to prove authentication

**Horton Principle** is a design principle for cryptographic systems: **“Authenticate what is being meant, not what is being said”.**

* There is often difference between what people want to authenticate and what we actually do to check.
* E.g. Google.com vs. G0ogle.com
* This makes it possible for an attacker to substitute a message with a valid MAC but a different meaning.

**Public Key Infrastructure (PKI)**

**PKI** is a suite of procedures for managing and revoking **Digital Certificates** (AUTHENTICATION) and managing Public Key Encryption. Its purpose is to facilitate the secure electronic transfer of information for a wide-range of network activities, such as eCommerce, internet banking and emails.

**Management of Digital Certificates: CERTIFICATE AUTHORITIES (CA)**

* They hold a “bank” of Public Keys
* They verify the identity of an individual through extensive vetting.
* If satisfied, they provide a **Signed Certificate** confirming the identity of the individual, encrypted with a private key held by the CA. So people communicating with this individual can use the CA’s Public Key to decrypt the certificate + authenticate the individual.

Problems with PKI’s

* **Horton’s Principle:** With web-browsing, we are often only authenticating a URL. The person / company that we want to communicate with is not the URL we are actually authenticating.
* **Single Point of Failure**: If a CA gets hacked, attackers can just generate fake Certificates for their own use to authenticate malicious websites.

**SSL – Secure Sockets Layer** is a protocol that manages server / client authentication and encrypted communication between clients and servers.

* If a webpage requires an SSL connection, URL will change from HTTP 🡪 HTTPS + a padlock icon appears in the browser once the server has been authenticated.
* Typically, an SSL will contain:
  + Your domain name, company name, address, city, state, country.
  + Expiration date of certificate
  + Details of the Certificate Authority responsible for issuing your certificate

**TLS – Transport Layer Security** is a protocol that ensures privacy between communicating applications and their users on the internet.

* When a server / client communicates, TLS ensures that no third-party may eavesdrop or tamper with any message.
* TLS is the successor to SSL.

**HTTPS – HTTP Over SSL or HTTP Secure** is a protocol for secure communication over a computer network.

* HTTP is using SSL or TLS as a sublayer under regular HTTP application layering.
* Encrypts and decrypts user page requests + the pages that are returned by the Web-Server
* Provides AUTHENTICATION of a website, protects data INTEGRITY / eavesdropping and MiTM attacks.

**Pretty Good Privacy (PGP)**

**PGP** is an encryption program which provides AUTHENTICATION + CONFIDENTIALITY for data communication.

Since encrypting long messages using RSA can be time consuming, PGP uses a faster symmetric encryption method and encrypts that key using the recipients public RSA key.

It can be used for authentication as follows:

* A message is hashed 🡪 we get a **Message Digest** 🡪 this is encrypted using the sender’s private key to get a **Signature**
* The signature is concatenated with the message and sent to the recipient.
* The recipient reverses the process using the public key of the sender to get the message digest and hashes the message it received to ensure it matches the message digest (Authentication).

The weakness of PGP is that you have to be sure that the public key you’re using belongs to the person you’re trying to authenticate.

**3 Ways to Authenticate**

**1. Something you ARE =** This property is part of you as a person

* Biometrics: Fingerprints or Iris (eye scanning)
* Limitations: It is unchangeable. If someone gets a copy of your fingerprints, you cannot change your fingerprint.

**2. Something you HAVE** = This property is something that you have as a good

* Driver license, password, key card
* Limitations: These are forgeable (Fake ID / password) + It must be with you when you use it.

**3. Something you KNOW** = This property is something that only you would know, but no one else

* Passwords
* Limitations: It can be used by someone else / You can forget it / It can be discovered by brute force or packet sniffing / eavesdropping
* Protection against limitations
  + Add a password salt
  + Improving protocol to prevent sniffing / eavesdropping i.e. encrypting all network traffic.

**Simple Authentication Protocol Examples**

* **S/KEY** is a One-Time-Password system
  + Begins with a secret **W**
  + A cryptographic hash function **H** **is applied *n* times to W**, producing a **hash chain of *n* one-time passwords**
    - Result would be H(W) 🡪 H(H(W)) … 🡪 Hn(W)
  + The initial secret W is discarded
  + The **user keeps the list of *n* passwords**, while the **server only keeps the final hash Hn(W)**
  + To authenticate, the user provides the server with Hn-1(W)
    - The server computes Hn-1(W) and compares with its Hn(W) value.
    - If Hn-1(W) produces Hn(W), then authentication is successful.
  + Security of S/KEY relies on the **difficulty of reversing cryptographic hash functions**.
* **SKID**
  + Hk = a keyed MAC. Both A+B know k.
  + A chooses random no. Ra and sends to B
    - **A ----------- Ra ------------> B**
  + B chooses random no. Rb and sends to A, Hk(Ra, Rb, Bob’s name)
    - **A <---------- Rb, Hk(Ra, Rb, “BOB”) -------- B**Ra is needed to prevent a replay attack  
      Rb is needed to prevent a MiTM attack
  + A sends Hk(Rb, Alice’s name) to B
    - **A ------------- Hk(Rb, “ALICE”) --------> B**
  + Now each party knows their shared secrets
* **Interlock protocol (RSa)**
  + A encrypts her message with B’s key + sends half of the encrypted message to B.
  + B encrypts his message with A’s key + sends half of the encrypted message to A.
  + A sends 2nd half of her message to B // B sends 2nd half of his message to A
  + *Strength of this protocol lies in the fact that half of an encrypted message cannot be decrypted. Attacker needs both halves to read it.*

**GENERAL SECURITY CONCEPTS**

**210 bits = 10 bits of security 21 = 1 bit of security (the exponent = # bits of security)**

**AES 128bit** = 2128 permutations

* Offers 128 bits of security
* 128 is a “reasonable number” for security

**Finding bits of security on AVERAGE**

* Divide bits of security by 2 🡪 i.e. 24 bits of security / 2 = 23 bits of security

**How to determine if a password’s bits of security is reasonable?***Typical qn: “Calculate bits of security of X and calculate how long it will take to brute force it. Justify any assumptions.”*

* Guess how long it will take to brute force it. Assume a password with 48 bits of security.
* Assume everything: 1Ghz machine, which can do 1 billion operations per second.
* Therefore, it can do 1 billion password guesses per second. (10 billion = 109 = ~230)
* 248 passwords / 109 passwords per second 🡪 248 / ~230 = **218 seconds to brute force**
* Therefore, **on average it will take** **217 seconds to brute force (defender POV)**

Bits of security –consider Average or Worse case?

* **ATTACKER POV = Worst Case (because you want to know for sure / know specifics from an individual case)**
* **DEFENDER POV = Average Case (because you’re defending against a large majority of people)**

**What is an appropriate amount of bits of security?**

* Magic number = 128 bits of security
* Why? 2128 possible passwords = Takes longer than the age of the universe to brute force.

**Bits of Security Examples - ASSUME EVERYTHING ABOUT THE SCENARIO**

**Facial Recognition**

* 10 bits of security for breaking facial recognition / person \* 7 billion people

**Fingerprints**

* 7 billion people = 7 billion unique fingerprints = ~233 = **3 bits of security**

**Hardware**

* Each computer’s power = 235 = **3 bits of work / computing power**

**Password Brute Forcing #1**

* [A – Z] [a – z] [0 – 0] = 26 + 26 + 10 = 62 options
* Assuming it is a 8 character password, # options = 62 x 62 x 62 x 62 x 62 x 62 x 62 x 62 =628
* **62 = ~26**  **🡪 26 x 8 = 248 🡪 password has 48 bits of security**

**Password Brute Forcing #2**

Guess the 6 char password of someone – where the password is all in lower case, except possibly the first letter can be either upper or lowercase.

1. Think about the main variables in the question  
   **[A-Z + a-z][a-z][a-z][a-z][a-z][a-z]**
2. Make the appropriate calculations to find a approx. number.  
   **52 x 26 x 26 x 26 x 26 x 26 = ~617 million**
3. Go to the nearest number which is a power of 2 + convert to power of 2  
   **617 = ~512** **(closest power of 2 value)  
   1 million = 220**  
   **512 = 29**  
   **Therefore, 220 + 29 = 229** **bits of work**
4. To find the average case bits of work, divide by 2  
   **229 bits of work / 2 = 228 bits of work**

**Steganography** is the practise of concealing a file, message, image or video within another file, message, image or video.

* It is relying on **Security through Obscurity**: reliance on secrecy of the design / implementation as the main method of providing security for a system.

**Null Hypothesis** is when there is no statistical significance between two variables

**Type 1 Error:** When null hypothesis = TRUE (no significance) and you reject it. 🡪 No effect by accepting two meds as different

**Type 2 Error**: When null hypothesis = FALSE (significance exists) and you reject it. 🡪 Kill people by accepting two meds as similar

**Converting #bits to factorial**: E.g. 170 operations = 5 \* 4 \* 3 \* 2 \* 1 = 120 = requires 7bits to brute force.

**Moral Hazard**: When someone has incentive to act the wrong way even though they’re supposed to act the correct way

**Common Vulnerabilities and Exposures (CVE)**

* As vulnerabilities get found and reported, they are given ID no’s and certain companies / naming authorities such as Google, Microsoft can hand out these numbers, which are all put into a centralised database.

**Responsible Disclosure:** Disclose vulns to the software vendor and give them a period of time to do something about it.

**Perfect Forward Secrecy** is a property of secure communication protocols in which a compromise of a key does not compromise past session-keys. Encrypted communication and sessions recorded in the past cannot be retrieved / decrypted given a future compromised key.

**Whistle Blowing**

**Zero Knowledge Protocol**

**Zero Knowledge Protocol** is a method where one party (*prover*) can prove to another party (*verifier*) that a something is true, without needing to show any information.

* For proof, the protocol must require the verifier to present a challenge to the prover, who needs to respond to the challenge in a way which will convince the verifier that the statement is true.

It must satisfy three properties:

* **Completeness**: If the statement is true, the verifier will be convinced of this fact by a prover
* **Soundness**: If the statement is false, no cheating prover can convince the verifier that it is true, except with small %
* **Zero-Knowledge**: If the statement is true, no cheating verifier learns anything other than that the statement made by the prover is true. Just knowing the statement (not the actual secret) is sufficient enough to prove the statement.

**Why ASYMMETRIC > SYMMETRIC ENCRYPTION**

If there were a group of 500 spies, that all needed to communicate securely to one another, then using a symmetric key encryption would mean that each spy would need a unique key for the 499 other spies in the network. Using an asymmetric key would mean that a total of only 500 public/private key pairs would be needed. If using a symmetric system, you would need 499+498+..+2+1 = 499\*500/2 = ~125 000 keys.

**Why SYMMETRIC > ASYMMETRIC ENCRYPTION**

Communicating with one friend when there is an already established secret between the two of you. Saves both party time since they don’t have to perform a key exchange and also saves them computational time.

**Birthday Attack**

Randolf controls both the original and legitimate message (he was asked to digitise it) and he controls a fake message he may wish to use to fool his brother. By appending random noise to the end of the legitimate recording, and to the end of a fake recording, he can effectively perform a birthday attack by continuing to vary the legitimate and fake message every so slightly until he arrives at a hash collision. When that occurs, he can send the legitimate message (which now has the same hash as a fake message) on to his brother to be signed, confident in the knowledge that later down the track he can claim that the fake message was actually the original and legitimate message.

**Opal Cards and security? Recommendations?**

* Storing credit on the card VS. storing credit in a database
* Having a unique identifier VS. no unique identifier on the card.

**CONS of Electronic voting and security? Recommendations?**

* More easy to tamper with votes than paper voting
* Privacy issues with electronic voting
* Many ways to compromise entry points to an electronic vote (compromise someone’s email account, computer that would do the voting and change their vote via. Spoofing etc.)
* How to authenticate person voting? (Privacy issues however with having ID’s for people)

**Incidence Response – EXAM QN: If I were the head of IT and this attack occurred, what would I do?**

**Benefit of Hindsight**

* When you’re doing incidence response, you don’t know what the right response would be until the incident has alrea happened.

Have a **Measured Response**

* If you break the fire-alarm glass, would you save lives? VS. Would you just annoy your boss?
* What is the correct move?
* *You have to think about a measured action to take and the consequences of that action*.

**Timing of an attack**

* Best time to attack is when the victim is NOT prepared.