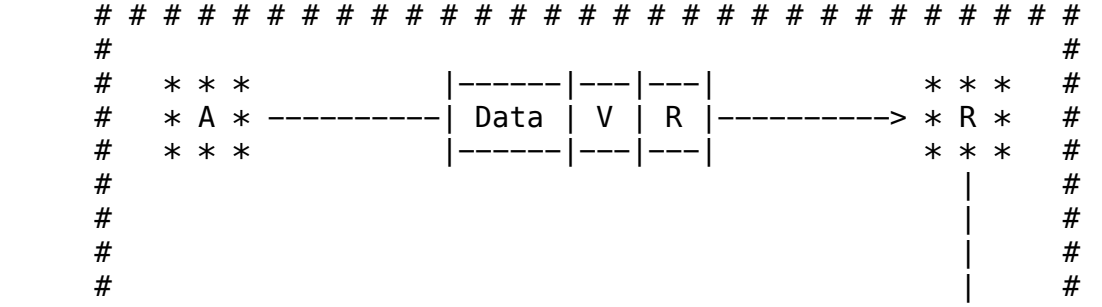
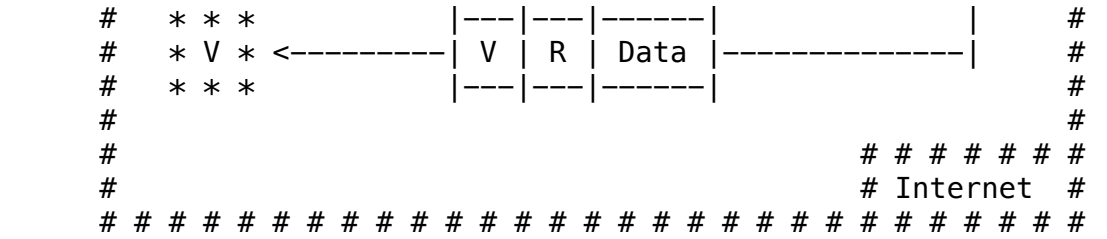
Week.12.txt

- March 29th, 2021
    - Recap
        - There are several types of network attacks, and each type has its own countermeasure(s)
        - Denial of service (DoS) attacks are typically done distributively
            - Typically, they are launched by attackers to prevent normal operations of a website or service provider
            - The main goal of an attacker performing a DDoS attack is financial/monetary
            - DoS attacks can cause a lot of damage
        - TCP SYN flood is one type of a DoS attack
    - Denial Of Service Attacks: Reflection
        - TCP SYN attacks are brute force in the sense that attackers will either directly launch the SYN attack, or flood SYNs from its victim nodes
            - This method can potentially expose the attacker, or the victim/compute node that the attacker has compromised
                - Victim nodes are machines that an attacker has managed to compromise and gain access to. The attacker has full control over the compromised machine
        - A sophisticated way of performing a denial of service attack is to use reflection
            - In a reflection-based attack, instead of attacking a victim the attacker tries to cause a non-compromised host to attack another host
                - i.e. If an attacker sends a DNS request message, or a TCP connection request message, with a spoofed source IP address, then the destination node will respond to the DNS message, or TCP message, to the victim node instead of responding to the attacker
                - If an attacker is able to generate a lot of spoofed messages with a victim's IP address as the source address, then the victim's computer may be over flooded with DNS reply messages or TCP response messages
        - i.e. Diagram of Reflection Attack



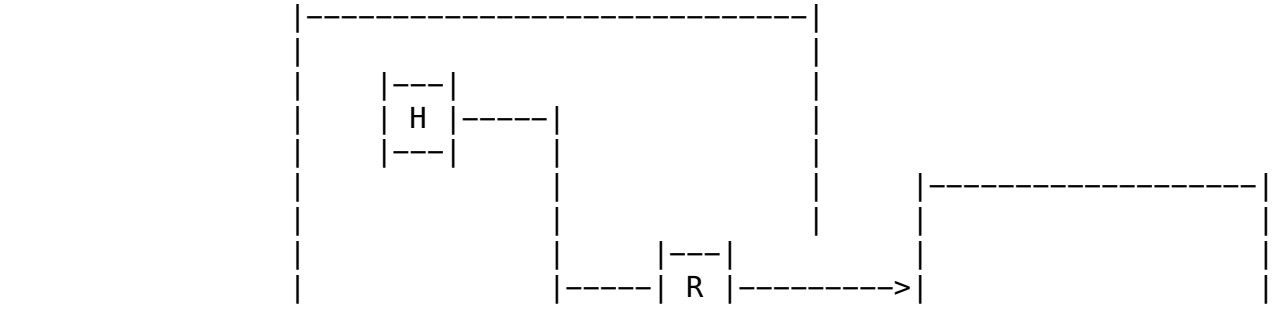


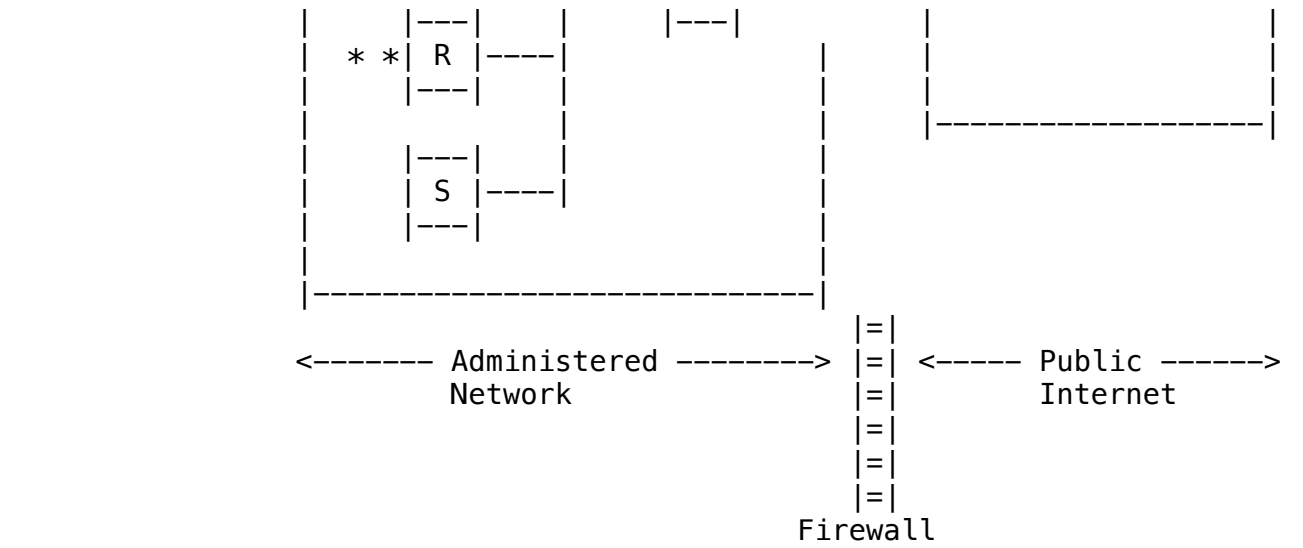
- 'A' represents the attacker
- 'R' represents the reflector node
  - This is the destination of the message that is sent by the attacker
- 'V' represents the victim node
  - This is the (spoofed) source of the message that is sent by the attacker
- The attacker (A) will send some data to a reflector node (R). The message will contain a spoofed source IP address, which corresponds to the victim's IP address. Once the reflector, or destination host, receives the initial message, it will respond with its own message. The response message can be a DNS reply or a SYN-ACK message in the second step of the TCP 3-way handshake. The destination IP address of the response message will correspond to the IP address of the victim node (V)

- A reflection attack is sinister, because the victim has no knowledge on what caused the attack, or flurry of messages
  - This is because the attacker spoofed the source IP address in its initial messages
- To summarize, a reflection attack is when a non-compromised host attacks another host
  - i.e. An attacker sends a DNS request, or TCP SYN, with source 'V' to server 'R'. Upon reception of the message, 'R' will send a reply to 'V'

- IP Traceback
  - A potential countermeasure to denial of service attacks, especially direct attacks, is IP tracebacks
  - IP tracebacks are not widely available in the Internet
    - However, routers have some kind of capability to tag packets in a probabilistic manner
      - Tagging every single IP packet is not feasible, because routers in the core network have to process packets at a data rate of multiple, or hundreds, of gigabits per second. Tagging every packet is too much computation, and potentially storage, overhead/ complexity
    - In the core network, routers spuriously and probabilistically tag (some) packets
      - The idea is that if enough packets have been tagged, then the originator of the attack can be pieced together by backtracing through routers that have

tagged similar packets
    - IP traceback is analogous to the story of Hansel and Gretel. When the kids, Hansel and Gretel, were lead into the forest by their stepmother, and left to fend for themselves, the kids were able to find their way back home by dropping small pebbles along the way
        - Since the kids only had a limited number of pebbles, they had to sparsely drop pebbles. In other words, they can't drop pebbles every 10 centimeters, rather, the pebbles need to be dropped a couple meters apart. Regardless, Hansel and Gretel were able to find their way back home by tracing the pebbles
            - This mechanism is very similar to IP traceback. If routers selectively tag packets from the source to destination host, then they will be able to figure out where the traffic is originating from, by tracing back the tags/pebbles
    - IP traceback is not an effective countermeasure to reflection-based attacks, because the response message originates from a victim, not the attacker
        - Hence, a more sophisticated mechanism is needed to counter reflection-based attacks
    - To summarize:
        - In IP traceback, routers probabilistically tag packets with an identifier. The destination can infer the path to the source after receiving enough packets
- Firewalls
    - Firewalls are a new form of defense, and are widely deployed
        - They are used by system administrators for networks, and by users for their systems
    - In general, firewalls help isolate an organization's internal network from the outside world. This is done by selectively allowing certain packets to pass through the firewall
        - Firewalls block packets coming from external networks, and they can block packets that originate from the internal network
    - The name 'Firewall' is self explanatory
        - A firewall serves as a filter for incoming and outgoing traffic
    - i.e. Diagram of Firewall



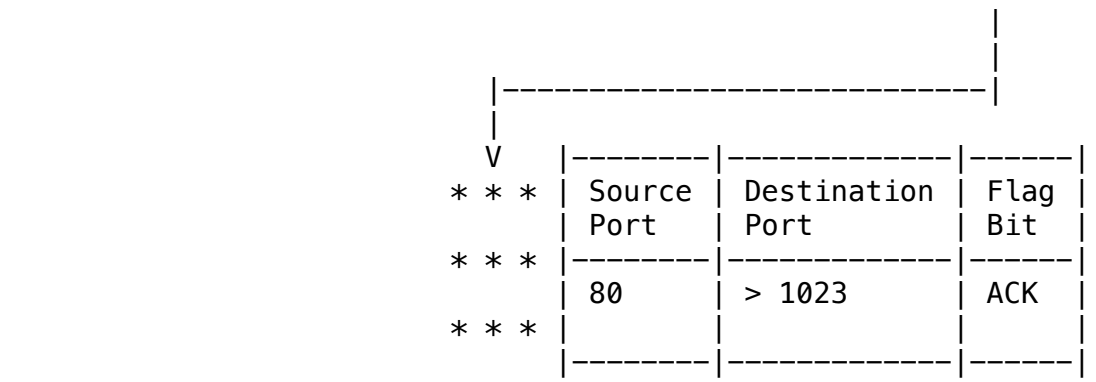


    - To summarize: A firewall isolates an organization's internal network from the Internet. It allows some packets to pass through, and blocks others
- Firewalls: Why
    - Firewalls can help prevent denial of service attacks
        - For instance, if the firewall can detect a SYN flooding attack, then it can selectively drop some of those bogus TCP connections
    - Firewalls can prevent the illegal modification or access of internal data
        - For instance, a firewall can be setup to only allow legitimate users to access the network or settings panel. If an attacker tries to gain access to an internal website to change its contents, then the firewall can deny access to the attacker. Thus, the attacker, or attackers, will not be able to modify or access the content
            - i.e. An attacker tries to replace CIA's homepage with something else, like a meme page, pwned page, or a redirection to something else
        - By setting up an application gateway, firewalls can only allow authorized users to access the internal network
    - Generally, there are 2 types of firewalls:
        1. Packet filtering firewall
            - Typically operates at the level of packets by looking at the information in the packet header at different layers of the protocol stack
            - There are two types of packet filtering firewalls, stateful and stateless. The key distinction is the decision of letting a packet pass or not is based on just the packet itself, or based on some kind of history of the packets that have been observed
        2. Application gateway
            - The decision regarding whether certain packets are

        let through or not is based on high level semantic information such as the user; whether the user has certain access/rights to certain content
      - Operates not only at the packet level of the TCP/IP protocol suite but also utilizes application layer knowledge
  - To summarize:
    - Firewalls are important because they can:
      - Prevent denial of service attacks
      - Prevent illegal modification/access of internal data
      - Allow only authorized access to inside network
    - There are 2 types of firewalls:
      1. Packet filtering
         - Stateless
         - Stateful
      2. Application-level
- Stateless Packet Filtering
  - The simplest kind of packet filtering firewall is stateless packet filtering
  - The key mechanism of stateless packet filtering is that the decision is made on a packet-by-packet basis
    - The router inspects the packet header, or maybe the packet's contents, and then decides whether it will forward the packet or drop it
    - The decision to drop or forward packets is based on:
      - Source IP address
      - Destination IP address
      - Source/Destination port number of UDP/TCP transport layer segments
      - ICMP message type
        - i.e. Drop all ping messages that an attacker may utilize to test whether certain hosts are available or not
      - TCP SYN/ACK bits
        - Firewalls can look at the flags in a TCP header to decide whether or not a packet is dropped
  - The decision in stateless packet filtering is typically binary. The packet is either forwarded or dropped
    - Packets are either forwarded from the external network to the internal network or vice versa; from the internal network to the external network
- Packet Filtering Example
  - Example 1: Block incoming and outgoing datagrams with IP protocol field = 17, and with either source or destination port = 23
    - This rule corresponds to blocking incoming and outgoing datagrams with IP protocol field 17. In addition, it will block any incoming and outgoing datagram with either source or destination port 23. What sort of packet may be blocked by this rule?

        - The first part of this rule blocks all UDP traffic. Recall that the IP header has a protocol field that is used for demultiplexing incoming IP datagrams to different transport layer services, and IP protocol field 17 corresponds to UDP
        - The second part of this rule disables telnet. This is because the port number utilized by telnet is 23. Since all incoming and outgoing datagrams with port = 23 are blocked, packets that correspond to Telnet are blocked
      - To summarize, this rule blocks all incoming and outgoing UPD traffic, and all telnet connections are blocked
    - Example 2: Block inbound TCP segments with ACK=0
      - This rule blocks all inbound TCP segments with an acknowledgement flag set to 0. The flag is disabled in the segment. Also, this does not refer to acknowledgement number. What does this imply? And what are the consequences of this rule? (Hint: What kind of TCP segment has acknowledgement flag set to 0?)
        - A segment with ACK=0 is a SYN flag/segment; it is the initial connection request from an external host. In a SYN segment, the SYN flag is 1, and the ACK flag is 0 (disabled). This rule prevents any (new) inbound TCP connections. It does not allow external hosts to initial a connection to the internal host. However, this rule does not prevent the internal host from initiating a connection to outside hosts or servers. Hence, all TCP connections have to be initiated from the internal host
    - Packet filtering examples are typically stored in an access control table. Control tables list the corresponding action, such as forward or drop, for packets with a particular attribute, such as IP address, port number, flag, etc.
- Stateful Packet Filtering (1)
    - The main problem with stateless packet filtering is that the decision is based on a packet-by-packet manner. It does not look at the history of the packet(s)
      - Often times it is difficult to have a good tradeoff. Stateless packet filtering applies a one size fits all rule to all packets. This can cause issues like dropping packets that should not be disallowed, or allowing a packet to come through but it does not make sense
    - i.e. Example of Access Control Table

Action	Source Address	Destination Address	Protocol	* * *
Allow	Outside of 222.22/16	222.22/16	TCP	* * *
				* * *

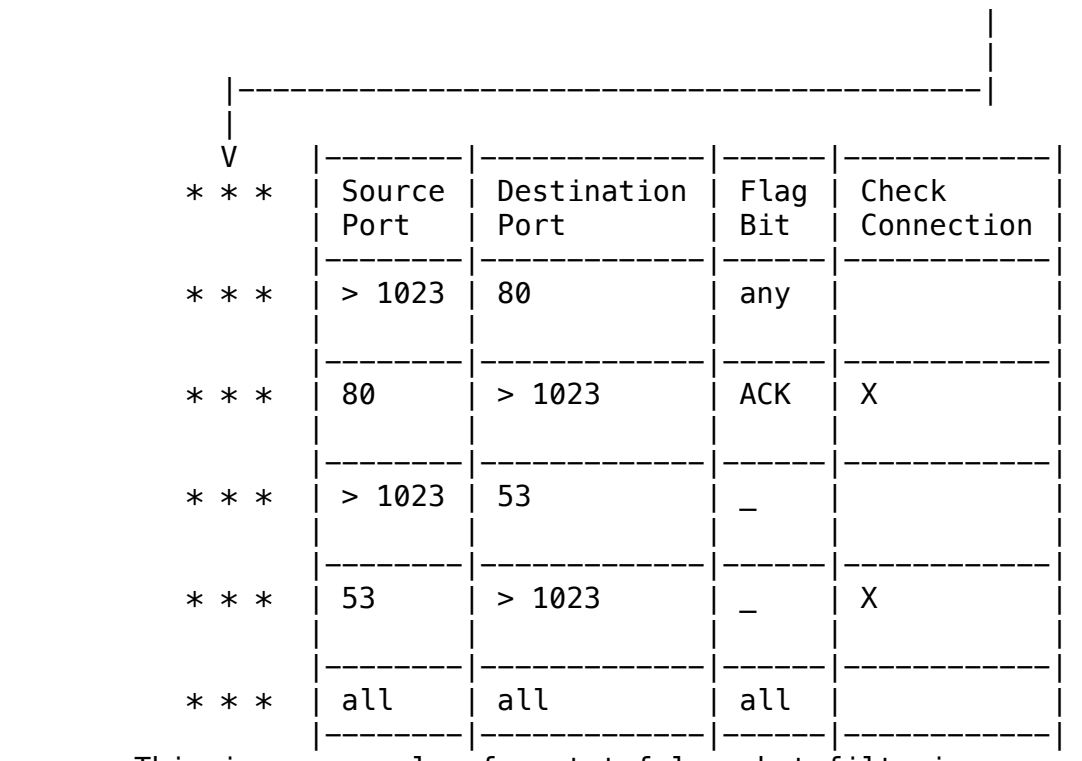


- This access control table contains a particular stateless packet filtering rule, which contains the corresponding allowable action
  - In other words, a packet is only allowed if it satisfies the condition in the access control table
- The access control table allows a packet whose source address is outside of the subnet. Most likely, this means that the host sending packets is in an external network. The protocol used must be TCP, the source port must be 80, the destination port must be greater than 1023, and the ACK bit must be enabled
  - Since the source port is 80, the server is outside of the network/subnet
    - Traffic coming from a web server is allowed to pass through the firewall
  - Any traffic coming into the network must be destined for an internal host
    - In other words, a packet is only allowed to enter the subnet, or LAN, if it originates from outside of the network
  - The destination port must be greater than 1023. This means that some port is utilized by, for instance, a client such as a web browser
  - The flag bit must be enabled. This means that it should be subsequent data transmission from the server, or it could be the second step of the TCP connection setup that originates from the connection setup request that comes from your own subnet
  - If an internal host initiates a connection to a web server, the incoming traffic will satisfy this rule, and the incoming packets will be allowed to pass through the firewall
- The problem with this rule is that the firewall is a stateless packet filter, and inspects traffic packet-by-packet. By enabling this rule, this can potentially be a problem and have negative consequences
  - It is possible that an attacker may generate a specific packet that satisfied this rule without any prior connection request from the internal host.

Thus, the attacker's malicious packet(s) will be able to pass the firewall, even though the packet does not correspond to any previous connection request from an internal host
  - An attacker can use this a stepping stone to launch more serious attacks like corrupting memory
- Stateless packet filtering is not always effective, because it inspects traffic packet-by-packet without looking at the history of the packet exchanges. Hence, it may allow some packet to get through that should not be there in the first place
  - In this example, if there is no prior connection request from the host, then the packet does not make sense and should be immediately dropped
    - To deal with this problem, stateful packet filters are used
- While a stateless packet filter inspects traffic packet-by-packet, a stateful packet filter can track the status of a TCP connection
  - Stateful packet filters can track the connection setup, connection tear down, and determine whether an incoming/outgoing packet makes sense
    - It can also timeout inactive connections on the firewall, so that no spurious packets coming from attackers can get through when the connection is no longer active

- Stateful Packet Filtering (2)
  - In stateful packet filtering, the access control list is augmented to indicate the need to check the connection state table before admitting/allowing a packet
  - i.e. Example of Access Control List

Action	Source Address	Destination Address	Protocol	* * *
Allow	222.22/16	Outside of 222.22/16	TCP	* * *
Allow	Outside of 222.22/16	222.22/16	TCP	* * *
Allow	222.22/16	Outside of 222.22/16	UDP	* * *
Allow	Outside of 222.22/16	222.22/16	UDP	* * *
Deny	all	all	all	* * *



	Source Port	Destination Port	Flag Bit	Check Connection
* * *	> 1023	80	any	
* * *	80	> 1023	ACK	X
* * *	> 1023	53	_	
* * *	53	> 1023	_	X
* * *	all	all	all	

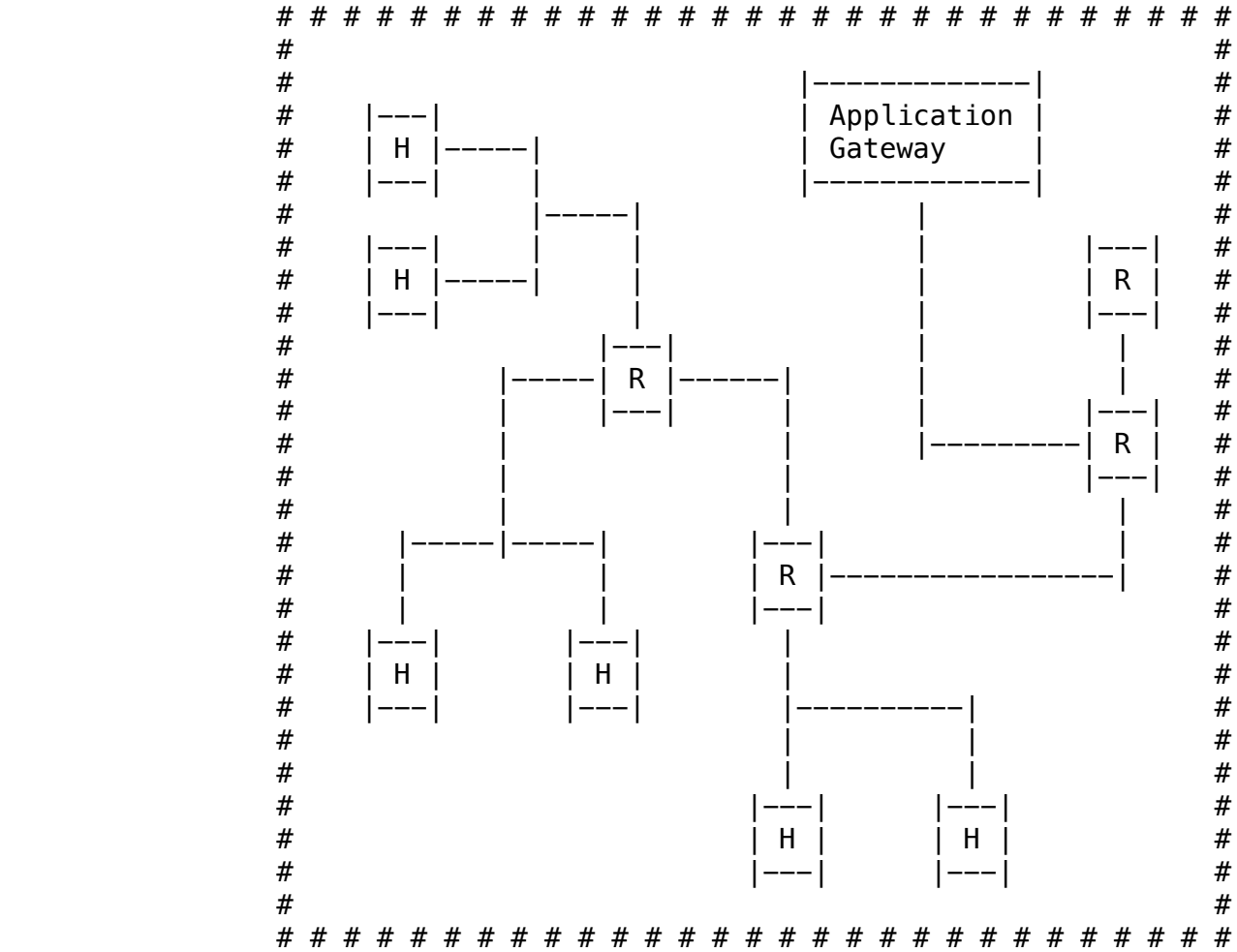
- This is an example of a stateful packet filtering access control list
- Compared to stateless packet filtering, the main difference is the last column that tracks/checks the connection status
  - The first few columns for stateless and stateful are the same. Both contain packet filtering rules such as action, source IP address, destination IP address, protocol, source/destination port, and flag bit
- In stateful packet filtering, an additional column is added to indicate whether the connection is checked or not
  - For instance, the 2nd row in the access control list contains an entry for the check connection field. This means that the firewall does not make decisions solely baesd on a packet's attributes, and whether it satisfied all other criterions or not. In addition to checking the fields/attributes, the firewall checks the connection of the packet
    - The packet is only allowed if its connection has a corresponding connection request, coming originally from an internal host or an already established connection. If a connection has already been established, or initiated, then the packet is allowed to pass through
  - This is the difference between stateful and stateless packet filtering; the check connection

field
- The UDP packet in row 4 is an indication of some message exchange; it is a response to some earlier request message that was sent by the internal host. Even though UDP is not a normal definition of a connection, because it does not require connection setup, its connection can still be tracked
  - The rule in row 4 allows packets from outside the subnet to enter the internal host if the packet is a UDP packet, has a source port of 53, destination port that corresponds to the application, and the flag bit is not enabled. However, the firewall will still check the connection of packets that fit this rule
    - The protocol utilizes port number 53. This corresponds to DNS. In this context, the check connection field ensures that the firewall does not accept UDP packets unless the DNS server has a prior connection with the internal host
      - In this scenario, the check connection field prevents an attacker from performing a reflection attack, because in order for a DNS packet to pass through the firewall, the DNS server must have a prior connection with the internal host. Meaning, the internal host needs to send the initial DNS request message, in order to accept any DNS response message
      - By pairing up the UDP messages with DNS, you can prevent an attacker from using a reflection attack to flood an internal host with lots of response messages
- When the check connection field is enabled, the firewall will check for any existing connection before admitting/ allowing the packet
  - If the check connection field is not turned on, then the stateful packet filter firewall will not keep track of, or check, the connection of the packet. In essence, it serves as a stateless packet filter firewall when check connection is not used

- Application Gateways
  - Application gateways differ from packet filtering firewalls, because they can utilize high level semantic information about users
    - For instance, a network administrator can use an application gateway to authenticate users, and make sure that those users are the only ones that are able to utilize certain system services, based on their role in the organization/company
      - This cannot be easily done via packet filtering

firewalls, because packet filtering has (typically) no information about what kind of user generates certain traffic. Packet filtering firewalls only look at the packet's headers, and in some cases the packet's payload. However, there is no information that discerns one person's traffic from another
  - For instance, packet filtering firewalls do not know if a professor in a certain department is trying to access grade information, versus a particular student trying to access the same information.
    - An application gateway can mitigate this problem that requires some kind of user access control in order to access certain applications

- i.e. Diagram of Application Gateway



  - 'H' represents a host
  - 'R' represents a router
- Assume that the network administrator does not allow every user to use Telnet, for security reasons, through the organization's network. Only a limited number of

users are authorized to use telnet and connect to an outside host. Recall that Telnet is dangerous to use, because it is not encrypted
        - To restrict access to Telnet, the system administrator needs to setup an application gateway that will serve as the intermediary.
            - Note: Sometimes the application gateway is also referred to as a proxy
            - If an internal host wants to access a remote service, such as Telnet, then the host has to first connect to the application gateway on the subnet
                - Then, the application gateway will be able to relay the data to the external server that the host does not have access to; for instance, a Telnet server. Before this is done, the user/host has to go through some kind of authentication process in order to show the gateway that it is capable and allowed to access the service in question
            - If a Telnet connection originates from a non-approved internal host, then the application gateway will block the request by dropping the corresponding packets. This can be done via a packet filtering firewall
        - An application gateway can restrict access to more than just Telnet. For instance, it can also perform some kind of filtering on web services. The application gateway can restrict access to certain web services to a subset of users
            - In this case, the system administrator will need to setup an HTTP proxy on the network's application firewall
                - i.e. All interns cannot access "YouTube", "Facebook", "Twitter", Etc.
    - To summarize:
        - Application gateways filter packets on application data as well as on IP/TCP/UDP fields
            - i.e. An application gateway can only allow selected users to use a service, like telnet, and connect to external hosts
                - In the case of telnet, the application gateway requires all telnet users to telnet through the gateway. For authorized users, the gateway sets up the telnet connection to the destination host. Once the connection is setup, the gateway relays data between both end-hosts. The router filter will block all telnet connections not originating from the gateway
- Limitations Of Firewalls & Gateways
    - Not all attacks can be mitigated or prevented by utilizing

  firewalls and application gateways
    - However, firewalls and application gateways are useful in a lot of situations, like enabling/enforcing explicit rules and making sure that the traffic is originating from internal host or external host
- If a host within the network performs IP spoofing, then a firewall or application gateway will not be helpful
    - If a host pretends to be a machine that does not fall into the range of IP addresses that are allowed or disallowed by the packet filtering firewall, then the corresponding rules in the access control table (ACL) will not be applicable
    - In general, a router or a firewall cannot really tell whether data actually comes from the true host that is associated with a particular IP address, or just a spoofed one
- Another problem with application gateways and firewalls is that they (typically) add more administration cost. If there are multiple applications that require special services, then each one will have its own application gateway
    - Big companies like Amazon and Azure offer cloud based application gateways for organizations. This offloads the task of setting up application gateways for the organization to the cloud
- In order to make an application gateway work, the system administrator needs to deploy client software
    - This needs to be setup in such a way that it can contact the gateway which allows the packet to eventually pass through and connect to the remote host
- The tradeoff between security and utility also exists for firewalls and application gateways
    - When dealing with networks, there is always the issue of the degree of communication that is allowed with the outside world - the Internet
        - From a user point-of-view, things need to be as flexible as possible. For instance, as a user I may want to setup a web server and allow others to reach, from an external network, and connect to my web services. However, this kind of traffic may be blocked by the firewall for security reasons
            - Hence, there is always a tradeoff between the degree of communication, or utility, and the level of security
- Despite the wide deployment of firewalls in organizations, as well as host computers, a lot of protected sites suffer from attacks
    - This indicates that firewalls or application gateways are not really the solution for solving network security problems
- To summarize:

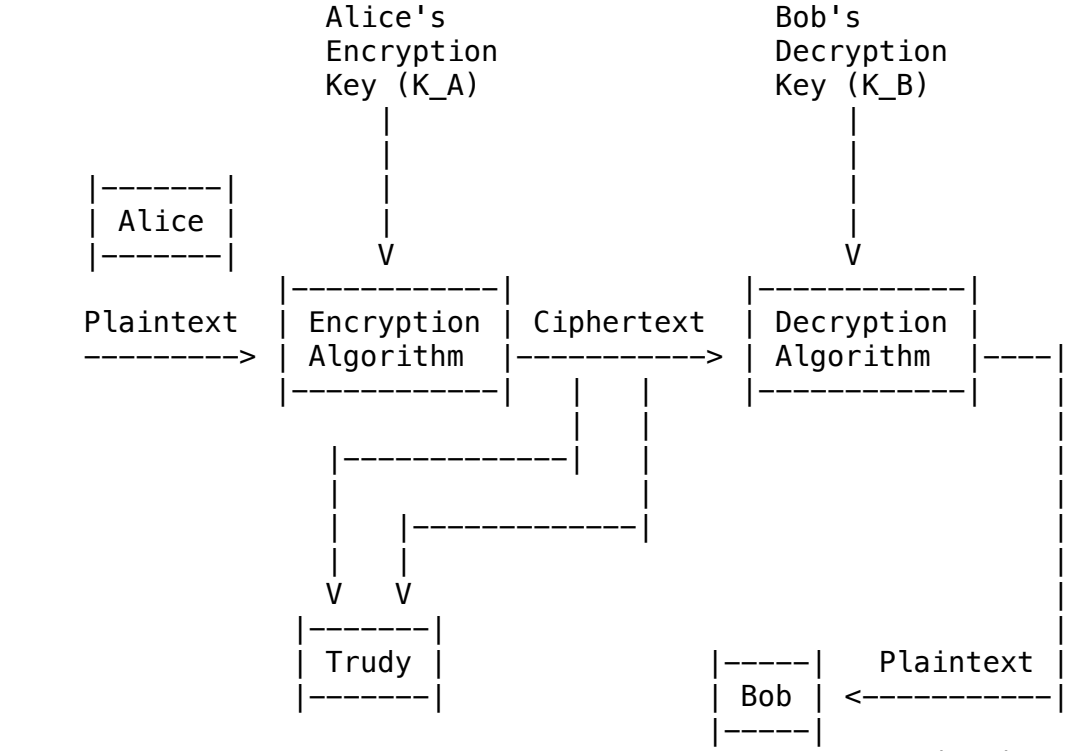
        - The router, or firewall, does not know if the data comes from the claimed source
            - i.e. The source IP address can be spoofed and none would be the wiser
        - Application gateways and firewalls create more overhead
            - i.e. If multiple applications need special treatment, then each will need to have its own gateway
        - For application gateways, the client software must know how to contact the gateway
            - i.e. Must set IP address of proxy in web browser
        - Filters often use an all or nothing policy for UDP packets
            - Bad implementation, because some UDP packets are necessary, like DNS
        - Setting up firewalls and application gateways has a tradeoff between degree of communication with the outside world and the level of security
        - Despite the wide prevalence of firewalls and application gateways, many sites still suffer from attacks
- Outline
    - Attacks and countermeasures
    - Security primer
        - This is the next part of the lecture
            - We will look at some basic concepts in cryptography that laid out the foundation for the discussion of security protocols in the TCP/IP protocol stack
            - If you have taken the information security course, you can skip this part of the lecture, because the material is essentially the same
                - However, it is a good idea to come back and review this material and listen to the lecture on security protocols in the TCP/IP stack
        - The security protocols in the TCP/IP stack are unique to network security
    - Security in different layers
- Q/A
    - Question: In a stateful packet filtering firewall, how is state information stored?
        - Answer: In stateful packet filtering, the state information can be stored in a database, or some other means. The information stored is past connections
    - Question: Are application gateways always a physical device?
        - Answer: They do not have to be physical devices. An application gateway can be some kind of virtual machine or application process that runs on a physical device
- Friends & Enemies: Alice, Bob, Trudy
    - The most well known example in the network security world is Alice, Bob, and Trudy. This example is used to introduce the basic concepts in cryptography, and gives rise to basic

primitives that are useful to achieve security goals such as confidentiality, message integrity, etc.
        - Alice and Bob are secret lovers that want to communicate "securely"
            - In this example, they are the good guys/people
        - Trudy is an intruder/attacker who poses a threat to the communication between Alice and Bob. She tries to compromise the communication between Alice and Bob
            - In this example, Trudy is the bad guy/person
    - The security requirements for (secure) communication between Alice and Bob are:
        - The messages exchanged between them are not leaked or eavesdropped by a third party; in this case it is Trudy
        - Trudy cannot compromise or modify the message that is exchanged between Alice and Bob
        - Trudy cannot prevent Alice and Bob from communicating by performing a denial of service (DoS) attack
- There Are Bad Guys Out There
    - Question: What can a bad guy do? In this context, what can Trudy do to Alice and Bob's communication? What are the things that Trudy can compromise?
    - Answer: A LOT!
        - Eavesdrop/Intercept messages
            - If Trudy is in the same local area network, she can eavesdrop on Alice and Bob's communication by intercepting their messages via Wireshark
                - However, intercepting messages does not mean that the message content is visible
        - Insert/Modify messages
            - Trudy can actively insert messages into the connection. This is accomplished via TCP hijacking; the attacker injects a falsified TCP segment into the ongoing connection between the two hosts. Then, this mechanism is utilized to launch a reverse shell, in order to have access to the victim's machine
                - This is one example of injection
            - Trudy can modify messages by changing the content of the message. In the case of a man-in-the-middle attack, an attacker can change the content of the exchanged messages
                - i.e. Instead of Alice transferred $30 to Bob, an attacker can change it to $30,000
        - Impersonation
            - Trudy can impersonate Alice or Bob by faking/ spoofing her source IP address in her packets. She can also change the MAC address in the frames she sends. Any field in the packet/frame can be changed to increase the illusion of impersonation
        - Hijacking

        - Hijacking is the process of taking over an ongoing connection by removing the sender/receiver and inserting yourself in their place
        - Trudy can hijack the connection between Alice and Bob via TCP hijacking
          - However, even with a successful TCP hijacking attack, Trudy will not be able to send anything from the terminal of the legitimate host. This is because you cannot type and send anything through telnet anymore
    - Denial of service (DoS)
        - An attacker, like Trudy, can prevent a service from being utilized by others, by launching a denial of service (DoS) attack
          - A denial of service (DoS) attack prevents legitimate users from accessing a service by overloading the server with bogus requests
- The Language Of Cryptography
  - For network security, the tools that we have at our disposal are primarily cryptography tools
    - Even though mechanisms like packet filtering are useful for mitigating or preventing certain types of attacks, cryptographic tools are needed to ensure message integrity, confidentiality, etc.
  - The basic concept of cryptography is converting plaintext into ciphertext. Without this conversion, if an attacker manages to compromise the communication channel, then the messages will be visible/readable
  - Cryptography methods use a pair of algorithms; one algorithm is for encryption, and the other algorithm is used for decryption
    - Encryption typically utilizes some key to encrypt plaintext into ciphertext. Without access to this key, an attacker will not be able to get any information pertaining to the plaintext that has been sent
    - The receiver (i.e. Bob) needs to use a decryption algorithm that uses the same or maybe a different key, depending on what kind of encryption mechanism is utilized, to recover the original plaintext
  - For an encryption mechanism to be successful, we need to make sure that there is no information that the intruder can gather from the ciphertext about the plaintext
    - This can be formalized in a more rigorous and probabilistic term
      - i.e. Given a ciphertext, the knowledge of the plaintext is the same as what you have before
        - So having access to ciphertext does not reduce the uncertainty that you have of the plaintext that is sent by the legitimate user
  - There are 2 types of cryptography methods:

1. Symmetric Key Crypto
    - The sender and receiver use the same key; the keys are identical
    - The key that is used to encrypt plaintext into ciphertext is also used to decrypt ciphertext into plaintext
2. Public Key Crypto
    - There are 2 keys
    - Each user has a key, called public key
        - The public key can be used to encrypt
    - In order to map from ciphertext to plaintext, and decrypt the message, a different key may have to be utilized; like a private/secret key
    - Public key crypto is used for more than just encryption and decryption, it is also used for digital signature
- The key difference between the 2 types of cryptography mechanisms is realizing whether the same key is utilized for encryption and decryption

- i.e. Diagram of Using Cryptography For Message Exchange



- In this example, Alice uses her encryption key (K_A) to convert plaintext into ciphertext, before sending it to Bob. When Bob receives the ciphertext, he converts it into plaintext with his decryption key (K_B)
    - If symmetric key crypto is used, then 'K_A' is the same as 'K_B'. Alternatively, if public key crypto is used, then 'K_A' and 'K_B' are different
- Trudy is the attacker/intruder, she is spying on Alice

        and Bob, and intercepting their messages
          - Fortunately, the messages she intercepts are encrypted and not readable
- Symmetric Key Cryptography
  - Between the 2 crypto mechanisms, symmetric key crypto is simpler and more intuitive to understand
  - i.e. Example of Symmetric Key Crypto

Cipher Table (Key)																									
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

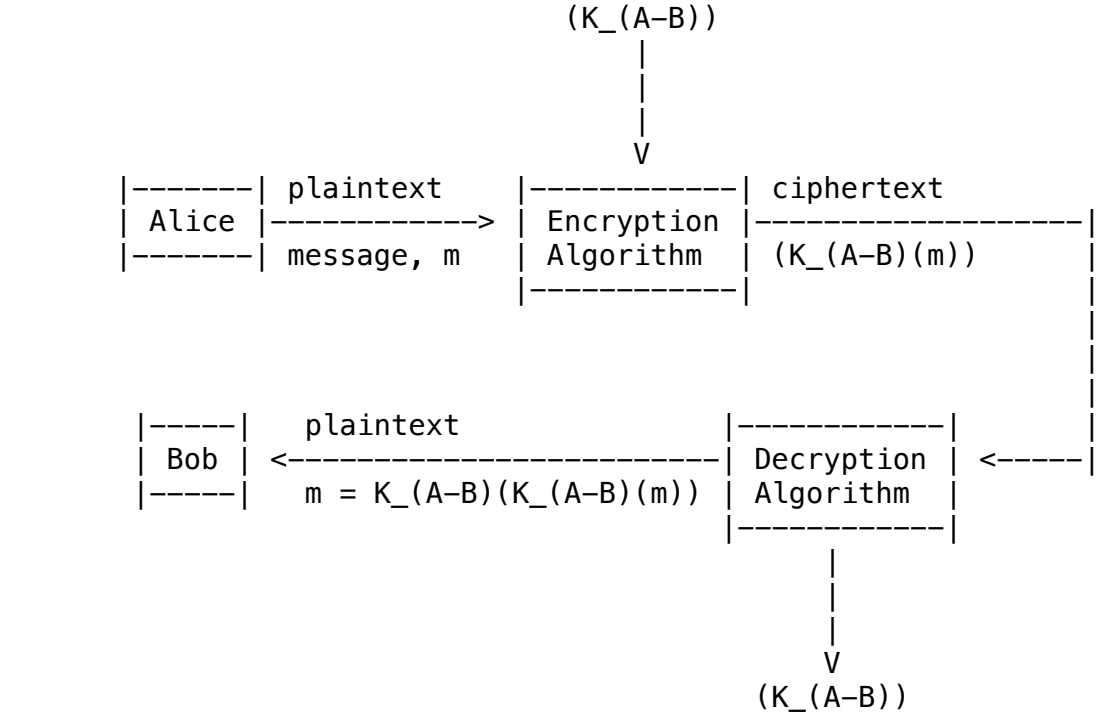
      - Message from Alice to Bob:
        - Plaintext: bob. i love you. alice
        - Ciphertext: nkn. s gktc wky. mgsbc
    - This example is not a very secure mechanism for symmetric key crypto. However, it is simple to understand
    - This example is a substitution cipher. It replaces letters in the message, or plaintext, with other letters according to a table
    - In the Cipher Table, the first row is the plaintext letter, and the second row is the corresponding substitution - the mapping of plaintext to ciphertext
      - i.e.
        - 'a' is replaced by 'm'
        - 'b' is replaced by 'n'
        - 'z' is replaced by 'q'
        - Etc.
      - When a plaintext message is sent, the program/ algorithm will look at the table and find the corresponding substitution
    - This example is a monoalphabetic cipher
    - If Alice sends a message to Bob that reads, "i love you. alice", then by applying this monoalphabetic cipher, the ciphertext will be "nkn. s gktc wky. mgsbc". This is because 'b' is replaced by 'n', 'o' is replaced by 'k', and so on and so forth
      - In order for Bob to recover the plaintext, he will need to reverse the process. If Bob has access to the same table and key as Alice, then he can map the letters back from 'n' to 'b', 'k' to 'o', and so on and so forth
    - This example is a simple implementation of symmetric key cryptography. It demonstrates that the key does not have to be some kind of algebraic operation; it can even be a table that requires lookups for encryption/decryption
    - The key idea here is that this example is symmetric key

cryptography. Since both Alice and Bob have access to the same table, they apply the same key in order to encrypt and decrypt the message
- Question: How can someone break this simple cipher? If Trudy did not have access to the table nor the key, how can she convert ciphertext into plaintext?
  - Answer: One way to break this cipher is to brute force it; try every single substitution until decryption makes sense. Also, it is important to think about what is the cost for performing a brute force attack? How many combinations are there? How many different tables can we possibly have? Since every letter will be uniquely mapped to another letter, the total possible number of combinations is 26 factorial (26!). This is because 'a' can map to 26 letters, and 'b' can map to the remaining 25 letters, and so on and so forth. Hence, the possible number of tables we can have is (26!). Then, we can try out every one of them and see whether the decryption produces a legitimate english word or sentence
    - This approach works as long as you are using a decently fast computer. In fact, 26! is not that big of a number in today's world of high performance computing and GPU power
      - However, brute force is not a good approach in general.
    - A more efficient, and better, approach is to look for patterns in the text. In the English language, we know that there are different letters and different frequencies. For example, 'e' may appear a lot more frequently than 'z'. Since the monoalphabetic cipher above has a one-to-one mapping between letters, you can analyze the frequency of particular letters that appear in the ciphertext. This will allow you to narrow down the possible options; you can reduce 26! to something more manageable. The final step is to see if the output is a meaningful sentence
      - This style of attack is referred to as frequency attack analysis, and reduces the overhead in decrypting ciphertext to plaintext
- To summarize, a substitution cipher substitutes one thing for another. A monoalphabetic cipher is a subset of a substitution cipher, and it substitutes one letter for another

- March 31st, 2021
  - Symmetric Key Cryptography (1)

- There are 2 types of crypto mechanisms that are commonly used nowadays:
  1. Symmetric Key Crypto
     - The sender and receiver share a common secret
       - This is why it is called symmetric
     - An example of a very naive symmetric crypto mechanism is called substitution cipher. It works by substituting, or mapping, letters according to a table. The table contains a mapping of letters
       - In the absence of the key and mapping, it may be very difficult for an attacker to decipher the ciphertext. However, if the attacker has sufficient computational power, then a simple brute force attack can be performed to obtain the plaintext. Alternatively, the attacker can also analyze the frequency of letters in the text, and gain more knowledge about the mapping. This approach is more efficient than a simple brute force
  2. Public Key Crypto

- Symmetric Key Cryptography (2)
  - i.e. Illustration of Symmetric Key Crypto



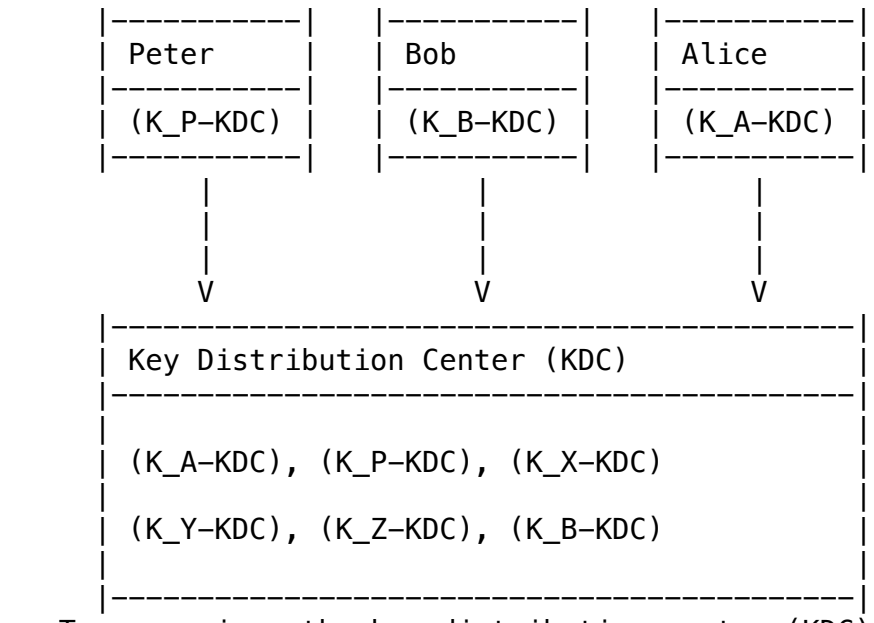
  - In symmetric key crypto, Bob and Alice share the same (symmetric) key: K_(A-B)
    - i.e. Key is knowing substitution pattern in mono-alphabetic substitution cipher
- It is very important to come up with strong symmetric key crypto mechanisms
  - Strong crypto mechanisms ensure message integrity,

        confidentiality, etc.
    - The standard committee have used several different symmetric key crypto standards
- Symmetric Key Crypto: DES
    - In 1993, the U.S. National Institute of Standardization (NIST) published an encryption standard called DES
        - DES stands for Data Encryption Standard
    - Data Encryption Standard (DES) uses a 56-bit symmetric key to encrypt a 64-bit plaintext input
        - If the input is greater than 64-bits, then the plaintext is broken down into blocks of at least 64-bits
    - How secure is DES?
        - After the standard was published in 1993, it was widely used for a while, but then later shown to be quite vulnerable during DES Challenge III
            - DES Challenge III was a joint effort between distributed.net and Deep Crack. It consisted of multiple people trying to obtain plaintext from ciphertext encrypted by DES
        - In January of 1999, participants in the third challenge of DES were able to find the key within less than a day; 22 hours and 15 minutes to be precise
            - The plaintext was "See you in Rome", and it is where the second AES conference took place; on March 22th 1999
        - DES, by itself, is not a secure enough encryption mechanism in the face of a very powerful attacker
    - DES can be made more secure by:
        - Instead of using 1 key, use 3 keys sequentially (3_DES) on each datum. As a result, the message is encrypted by 3 keys, one after another
        - Utilize cipher-block chaning
            - This trick is used in many modern cryptography mechanisms. Essentially, the encryption of the next block depends on the key that is generated from the outcome; the outcome is the encrypted text from the previous block. As a result, in order to decrypt a particular block within the data, an attacker needs to successfully decrypt the previous data as well
                - By utilizing this mechanism, the security properties of the resulting ciphertext are strengthened
    - Regardless of what tactic is used to strengthen DES, fundamentally it is a weak cipher
- AES: Advanced Encryption Standard
    - In November of 2001, the National Institute of Standardization (NIST) published a new encryption standard called AES
        - AES stands for Advanced Encryption Standard
        - AES replaced DES, which is cryptographically less secure

    - The Advanced Encryption Standard (AES) processes data in 128-bit blocks
        - The keys can be of different sizes, such as 128, 192, or 256 bits
            - The size of the key can be decided by the communicating parties that are utilizing AES as their encryption mechanism
    - Since its standardization by NIST, AES has not been subject to any practical attack that can allow someone without the knowledge of the key to read data encrypted by AES
        - Note: This assumes that the encryption mechanism is properly implemented (i.e. Bug-free)
        - It has been estimated that a brute force dictionary attack on AES will take 149 trillion years, to decipher the key
            - In comparison, it takes 1 second to decrypt ciphertext that is encrypted via DES. The ratio between DES and AES is 1 second to 149 trillion years. Thus, AES is a very strong crypto mechanism
        - However, there are other ways for an attacker to extract/obtain the key from unsuspecting victims
            - i.e. An attacker can execute a social engineering attack to trick the victim into revealing the key
            - i.e. An attacker can use a type of side channel attack to find other ways to obtain the key
        - An attacker cannot use ciphertext encrypted via AES to obtain the key or the original plaintext
            - Researchers have proven AES to be nearly impossible to decrypt without the key, provided that the encryption mechanism is correctly implemented
    - AES is widely utilized in network communications and securing data/information
- How Do Bob & Alice Agree On Key Value
    - In symmetric key crypto, the basic idea is that once Alice and Bob have obtained the shared secret key, then they can use this key for encrypting messages
        - Bob and Alice use the shared secret key to encrypt and then exchange messages
        - As an example, if Alice wants to send a message, 'm', to Bob, then she will encrypt the message by utilizing the shared secret key that has been pre-established prior to the start of communication with Bob. The plaintext is encrypted/converted into ciphertext by applying the key and a crypto mechanism/algorithm
            - Upon reception of the message, Bob can apply the same key to decrypt the message
        - Alice and Bob encrypt and decrypt messages with the same key. Hence, it is called symmetric key crypto
    - In cryptography, it is very important for communicating

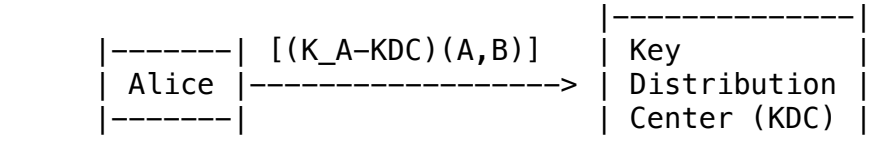
parties to be able to exchange secret keys. It is also important for parties to come to an agreement of a shared secret key
  - One way to exchange a shared key is to communicate out of channel
    - i.e. If you want to establish a secret with your friend that both of you will use to encrypt messages/emails, then you can call your friend and, over the phone, tell them what the shared secret key is; like "Have a nice day"
    - However, this is generally regarded as bad practice and we should not rely on any kind of out of band mechanism; manually exchanging keys is bad practice
      - Ideally, communicating parties need to be able to automatically agree on a shared secret key, prior to exchanging messages
- For symmetric key crypto, there are 2 ways for communicating parties to agree on a shared secret key:
  1. Rely on a trusted third party center, called key distribution center (KDC)
  2. Utilize Diffie-Hellman key exchange protocol
      - This can be done entirely between the communicating parties, and does not need or require a trusted third party. The Diffie-Hellman protocol has some clear advantages
- Key Distribution Center (KDC) (1)
  - The job of the key distribution center (KDC) is to act as a trusted party/entity and store pre-shared symmetric keys for each communicating party
    - This procedure is a one-time cost
    - For instance, the key distribution center will store Alice and Bob's symmetric keys
  - In the case of Alice and Bob, they can use some kind of out-of-band mechanism to establish their symmetric keys with the key distribution center (KDC). So, Alice will share her key with KDC, and Bob will share his key with KDC
    - Once this information has been established, Alice and Bob can store their own keys, and the key distribution center (KDC) will also store both of their keys
  - With the help of the key distribution center (KDC), communicating parties, such as Alice and Bob, can establish a shared secret key
    - The secret key can change over time. If Alice and Bob want to communicate at a later date, they can use a different key
      - This type of key is called a session key, and it tends to be dynamic
        - Session keys are the reason why it is important to have an automated mechanism that can establish secret keys between communicating

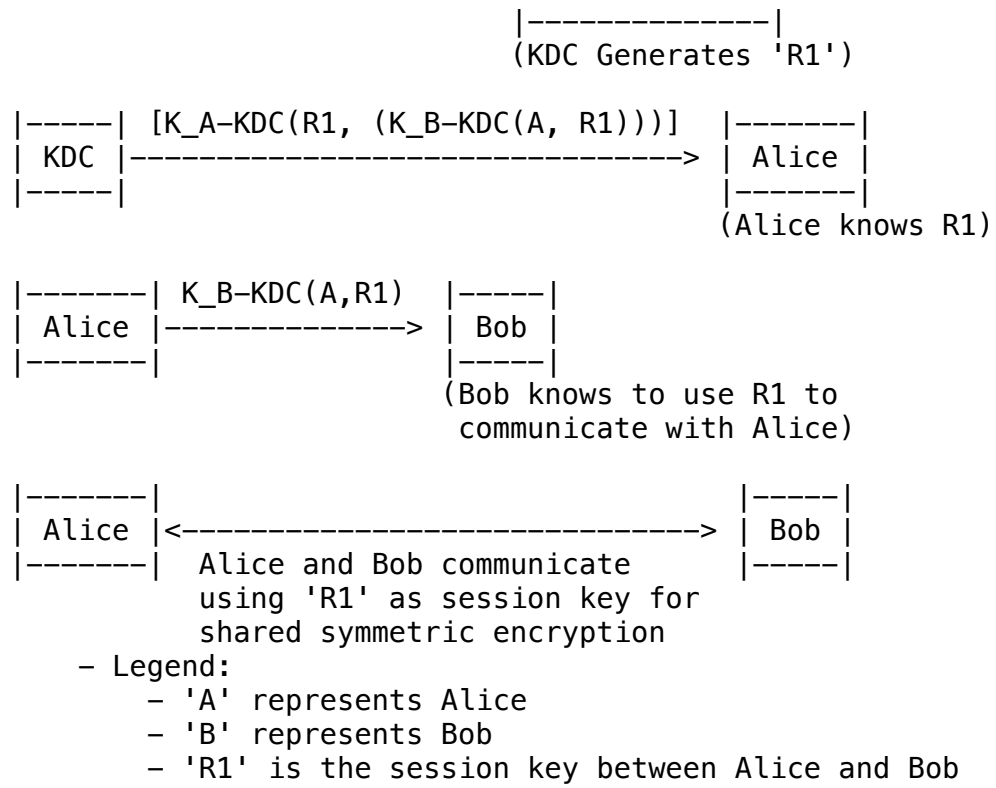
parties
    - The assumption here is that all communicating parties already have their shared secret key stored on the servers of the key distribution center
      - So, the key distribution center (KDC) will store a secret key between:
        - Alice and KDC
          - Denoted as (K_(A-KDC))
        - Bob and KDC
          - Denoted as (K_(B-KDC))
      - Alice and Bob know their own symmetric keys for communicating with the key distribution center (KDC)
- i.e. Diagram of Relationship Between Users & Key Distribution Center



- To summarize, the key distribution center (KDC) shares different secret key with each registered user, and there are many users

- Key Distribution Center (KDC) (2)
  - The key distribution center (KDC) helps parties that want to communicate with one another generate a shared key. However, the problem is that the key distribution center (KDC) cannot send the generated shared key in plaintext
    - It needs to somehow encrypt this information, and ensure that only the corresponding communication entities that are trying to establish a connection with one another are allowed to receive the corresponding key
  - i.e. Diagram of Key Distribution Center Facilitating Communication Between Parties (Alice & Bob)





- Legend:
  - 'A' represents Alice
  - 'B' represents Bob
  - 'R1' is the session key between Alice and Bob
  - 'K_A-KDC' is the pre-shared key between Alice and the key distribution center (KDC)
  - 'K_B-KDC' is the pre-shared key between Bob and the key distribution center (KDC)
- This approach to secure communication assumes that the communicating parties have established their identity and shared secret key with the key distribution center (KDC)
- Assume that Alice wants to send a message to Bob. Before any message can be sent, Alice and Bob need to come to an agreement on the shared secret key

- The first step is for Alice to use the pre-established shared key with the key distribution center (KDC) to encrypt the information of the entity of Alice and Bob
  - This is the first message; it is an encrypted message that uses a shared secret key between Alice and the key distribution center (KDC). The message may contain information such as Alice's IP address, Bob's IP address, etc.
- Once the key distribution center (KDC) receives the initial message from Alice, KDC will generate a random number, 'R1'; a sufficiently random number that an attacker cannot easily guess. Then, KDC will reply to Alice with a message that includes the newly generated random number, 'R1'
  - 'R1' is important, because it will be utilized as a shared key

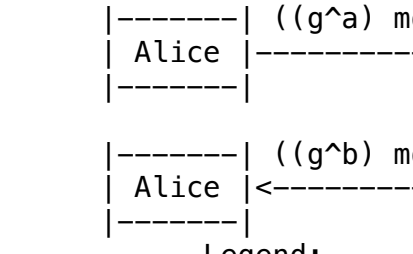
    - Since the key distribution center (KDC) and Alice have a shared secret key, KDC will be able to decrypt Alice's initial message and extract the plaintext
      - The plaintext contains the identities of the communicating parties. Now, the key distribution center (KDC) also has Bob's shared key
- In response to Alice's initial message, the key distribution center (KDC) will use the shared secret key and encrypt Alice's identity as well as the newly generated random number, 'R1'
    - From an attacker's point of view, as long as an attacker does not have the shared key between Alice and the key distribution center (KDC), then the attacker will not be able to decrypt any message exchanged between the communicating parties. In additional, an attacker will not be able to decrypt what is included in 'A' and 'R1'
- Once Alice receives the initial message from the key distribution center (KDC), then Alice will use the shared secret key to decrypt the message and retrieve the corresponding information
    - Alice is able to learn the value of 'R1'. This randomly generated number will be used for subsequent communication with Bob
    - The second field in this message, (K_B-KDC(A,R1)), cannot be decoded by Alice without the shared key between Bob and KDC. This message is sent from Alice to Bob, and is used as proof/evidence that the correct parties are communicating with one another
- Alice will send the second field, the inner message, to Bob. At this step, the key distribution center (KDC) is no longer involved
    - The message Alice will send to Bob is: (K_B-KDC(A,R1))
- Upon reception of the message, from Alice to Bob, Bob will decode the message using his shared key with the key distribution center (KDC). Once the message is decrypted, Bob is able to retrieve the shared secret session key, 'R1' between Alice and Bob
    - Alice and Bob are now able to securely communicate with one another without the help of any third party. An attacker will have a hard time decrypting their messages
- This entire process between Alice and Bob accomplishes two things:
    1. Alice and Bob are able to establish their shared secret key, 'R1'
    2. Since Alice and Bob have their own shared key with the key distribution center (KDC), they will know

that they are communicating with each other, or the correct person
- Bob knows that he is establishing a shared secret key with Alice, and Alice knows that only Bob is able to decode the subsequent message. Thus, whoever has the session key, 'R1', must be Bob
- Note: This entire process relies on the help of a trusted authority, like a key distribution center (KDC). The communicating parties must have a pre-shared key with the key distribution center, before they can establish session keys with each other
- To better understand this approach/protocol, think about hypothetical scenarios such as:
- Can this procedure be done by sending an encrypted message with 'R1' to Alice and then to Bob, respectively
- Would this work?
- How does KDC allow Alice and Bob to determine the shared symmetric secret key to communicate with each other?
- Question: How does Bob know 'R1'?
- Answer: In the second step, Alice receives a message from the key distribution center (KDC). This message contains 'R1', and (K_B-KDC(A,R1)). Alice is able to decrypt the outer message with her pre-shared key with KDC and retrieve 'R1'. However, the inner message is meant for Bob, and only Bob can decrypt it with his pre-shared key with KDC. Once Alice sends this message to Bob, then Bob will be able to decode it with his pre-shared key with KDC, and figure out 'R1' and who this shared key must be used with
- This approach is provably correct. However, utilizing a trusted third party creates additional overhead. In fact, often times users may need to pay the key distribution center to store their key
- The Diffie-Hellman key agreement protocol is a clever way to completely eliminate the need for a trusted third party

- Diffie Hellman Key Agreement Protocol
- The purpose of Diffie-Hellman is to establish a shared secret key between two communicating parties without involving a third (trusted) party
- i.e. Alice and Bob can communicate with each other without the help of a key distribution center (KDC)
- At its core, Diffie-Hellman utilizes the properties of algebra; notably modular arithmetic
- Prior to establishing the shared key via Diffie-Hellman key agreement protocol, we assume that we have 2 numbers. The first number, 'P', is a large prime number. The second number, 'G', is a base

    - Both numbers are public, and anyone can see them
    - For communicating parties, such as Alice and Bob, each party needs to have their own secret key
  - Keys are computed in the following manner, assume that Alice and Bob are the communicating parties:
    - Alice will pick a random number, 'a', and Bob will pick his own random number, 'b'
      - Alice's secret number is 'a', and Bob's secret number is 'b'
    - Alice and Bob will use the publicly known numbers, 'p' and 'g', and apply their own secret key/number to it
      - The computation Alice performs is: [(g^a) mod p]
      - The computation Bob performs is: [(g^b) mod p]
    - After both parties compute their secret with the public numbers, they will exchange their results
      - An attacker can sniff these messages, and discover the results. However, the assumption is that as long as 'p' is sufficiently large, it will be very hard for an attacker to resolve the value of 'a' and 'b' from the encrypted messages that are exchanged after the secret keys are established
        - This is only possible because of the property of prime numbers. Although, in a post-quantum world, quantum computers may be able to crack this encryption mechanism and extract the value 'a' and 'b' from the messages. But for this example, let's assume that this is strong enough
        - Put simply, just by looking at the transmitted messages, an attacker will not be able to find or determine the secret key
    - Once Alice and Bob have received each other's results, they perform additional computation to establish the shared secret. The secret key is established via the following property: K = [(((g^a) mod p)^b) mod p]
      = [(g^ab) mod p]
      = [(((g^b) mod p)^a) mod p]
      - Upon reception of the message from Alice, Bob performs the operation of raising the value sent from Alice, (g^a mod p), to the power of 'b', its own secret, and then applying modular 'p'
        - In short, Bob performs the operation: [((g^a mod p)^b) mod p]
          - This entire operation is performed by Bob
      - Upon reception of the message from Bob, Alice takes the number, (g^b mod p), received from Bob, raises it to the power of 'a', her own secret number, and then applies mod 'p' to the result
        - In short, Alice performs the operation: [((g^b mod p)^a) mod p]
      - Mathematically, it can be shown that the operation

performed by both Alice and Bob yields the same result
- The result is simply: [g^(ab) modular p]
- This is the shared secret
- i.e. Diagram of Alice & Bob Using Diffie-Hellman Key Agreement Protocol To Establish Their Shared Secret

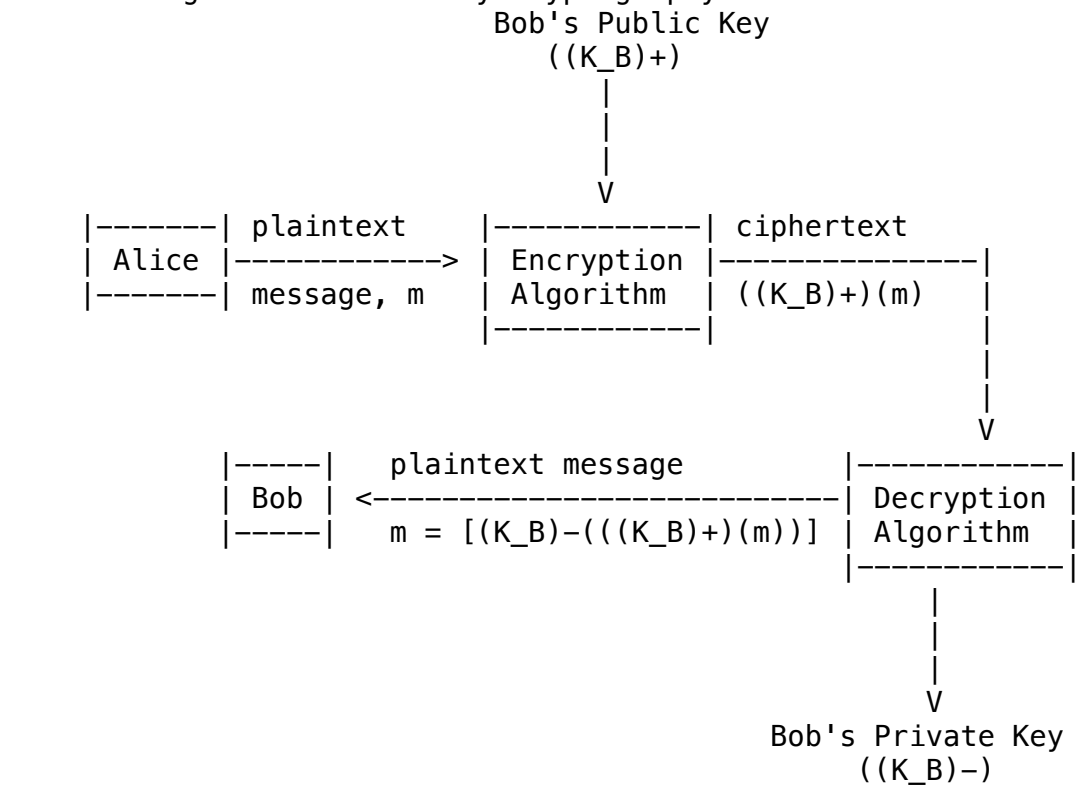


- Legend:
- 'g' is a base number
- 'p' is a prime number
- 'a' is Alice's secret number/key
- 'b' is Bob's secret number/key
- Alice will pick a random secret/number, 'a'. Then, she will compute and send ((g^a) mod p)
- Bob will pick a random secret/number, 'b'. Then, he will send ((g^b) mod p)
- Diffie-Hellman allows Alice and Bob to agree on a shared secret in a public channel against passive (i.e. Eavesdropping) adversaries
- The following mathematical formula is used by communicating parties to derive the secret key:
K = [(((g^a) mod p)^b) mod p]
= [(g^ab) mod p]
= [(((g^b) mod p)^a) mod p]
- In the entire Diffie-Hellman key agreement protocol, there is no need to involve a (trusted) third party. If Alice and Bob want to communicate with each other, then they only need to exchange some information that is not decodable, in terms of an attacker extracting the secret keys from the exchanged messages
- Communicating parties can arrive at a common value that will be used as a shared secret without the help of a key distribution center (KDC)
- The Diffie-Hellman key agreement protocol is used in security protocols in the TCP/IP protocol stack

- Diffie Hellman Example
- Assume that Alice and Bob want to securely communicate with one another. The process of establishing secret keys via Diffie-Hellman is as follows:
- First of all, Alice and Bob need to agree on 2 pre-agreed numbers, 'p' and 'g'
- Both parties agree on 'p' = 23, and 'g' = 5
- 'p' has to be a prime number, but 'g' doesn't
- Alice chooses her secret number to be 6, and performs

the following computation: ((5^6) mod 23). The result of this calculation is 8, and Alice sends it to Bob
        - Bob chooses his secret key to be 15, and performs a similar computation to Alice's. Bob does the following: ((5^15) mod 23), obtains 19 and sends it to Alice
        - Upon reception of each other's messages/numbers, each party performs the following computation:
            - Alice computes: ((19^6) mod 23)
                - Alice takes the received number, raises it to the power of her secret number, 6, and then takes the modulus of the result
                    - The result is 2
            - Bob computes: ((8^15) mod 23)
                - Bob takes the received number, raises it to the power of his secret number, 15, and then takes the modulus of the result
                    - The result is 2
            - Note: Both Alice and Bob arrive at the same number, which is the secret key, and is used to encrypt subsequent messages
        - Diffie-Hellman is hard to crack because the assumption is that finding ((5^??) mod 23) = 8 and ((5^??) mod 23) = 19 are very hard
    - If Diffie-Hellman is correctly implemented, then it is hard for an attacker to find out each party's individual secret key/number from the messages exchanged
        - For this to work, 'p' needs to be a very large prime number; this can be proved through the properties of prime numbers
- Public Key Cryptography (1)
    - The main difference between public key and shared key crypto lies in whether or not the key used for encryption and decryption are the same
        - In symmetric key crypto, a single key is used for both encryption and decryption
        - In public key crypto, two different keys are used
    - In public key cryptography, the two different keys used are:
        1. Public Key
            - The public key key is known to everyone
                - i.e. When logging into a bank, the public key is sent over the network to the browser to use for authenticating the web page
        2. Private Key
            - The private key is only known to the user/receiver
                - This information is private
- Public Key Cryptography (2)
    - To ensure the confidentiality of messages exchanged via public key crypto, the sender uses the receiver's public key to encrypt the message
        - At the other end, the receiver applies his own private

key to decrypt the message
  - This entire mechanism is different from symmetric key crypto, where a single shared key is used to encrypt and decrypt the message
- i.e. Diagram of Public Key Cryptography



  - The notation ((K_B)+) is used to indicate Bob's public key
    - This is read as: 'K' subscript 'B', and superscript '+' (plus)
    - The 'B' indicates the receiver; it is Bob
    - The '+' indicates that the public key is being used
  - The notation ((K_B)-) is used to indicate Bob's private key
    - This is read as: 'K' subscript 'B', and superscript '-' (minus)
    - The 'B' indicates the receiver; it is Bob
    - The '-' indicates that the private key is being used
- This is the procedure for sending a plaintext message from Alice to Bob, using Bob's public and private key
  - Since Bob's public key is known to everyone, including Alice, Alice will encrypt messages she sends to Bob using Bob's public key
  - When Bob receives a message, he will decrypt using his private key
    - Note: Only Bob knows his private key

        - By applying Bob's private key to the encrypted message, Bob can retrieve the original message
            - Note: The message is encrypted with Bob's public key
    - For public key crypto to work, the following equality needs to hold: `m = [(K_B)-(((K_B)+)(m))]`
        - When designing a public key crypto mechanism, this equality needs to hold
            - This is very non-trivial to do
- Public Key Encryption Algorithms
    - The following property needs to be ensured by public key encryption algorithms: m = [(K_B)-((K_B)+(m))]
        - Additionally, the private key should not be able to be derived/computed from the public key ((K_B)+), which, by definition is known by everyone/anyone
            - A successful public crypto algorithm must satisfy the properties listed above
        - This property ensures equivalence; applying the encryption or decryption algorithm will yield the same message
            - In fact, this holds true even if the public key, ((K_B)+), and private key, ((K_B)-) are exchanged/swapped. For instance, you can encrypt a message with Bob's private key, ((K_B)-), and then decrypt with Bob's public key, ((K_B)+). In the end, the plaintext is the same
                - This has applications in digital signatures
    - Normally, messages are encrypted with the receiver's public key, and decrypted with the receiver's (own) private key
        - This ensures confidentiality
    - The private key must always be private, and there is no feasible way to determine the private key from the public key or the encrypted message
        - This is the second property of public key encryption algorithms
    - The most famous public key crypto algorithm is called 'RSA'
        - 'RSA' is heavily utilized in network communication
        - The term 'RSA' comes from the initial of its 3 inventers/founders; Rivest, Shamir, Adleman
            - In 2002, the inventors of RSA algorithm received a Turing Award for their contribuion to computer science
                - The Turing Award is the Nobel Price Award of computer science
    - To summarize, public key encryption algorithms must satisfy the following requirements:
        1. The private key, (K-), and the public key, (K-), need to satisfy the following property: [(K-)(K+(m))] = m
        2. Given the public key, (K+), it should be impossible to compute the private key, (K-)
- RSA: Choosing Keys

    - RSA is a very clever, and non-trivial idea
        - It is easy to understand, but hard to invent/discover
    - RSA works in the following manner:
        1. Each communicating party needs to choose 2 (very) large prime numbers, called `p` and `q`
            - This decision is independently made by each party
            - A very large number is greater than 1024 bits
        2. For each set of numbers from step 1, `n` and `z` are computed using the following equations:
            a. n = p * q
                - `n` is simply the product of `p` and `q`
            b. z = (p - 1)(q - 1)
                - `z` is the product of `p` minus ` and `q` minus 1
        3. Choose a value, `e`, that is less than `n`, and has no common factors with `z`. In other words, `e` and `z` are relatively prime
            - The public key is [(n,e)]
                - i.e. ((K_B)+)
        4. Choose a value `d` such that ((e * d) - 1) is exactly divisible by `z`. In other words, since ((e * d) - 1) is divisible by `z`, it implies that:
           (((e * d) mod z) = 1)
            - The private key is [(n,d)]
                - i.e. ((K_B)-)
        5. To prove that this works, we need to show that the following property holds for both the public key and private key: m = ((((m^e) mod n)^d) mod n)
            - This is the first requirement/property of public key encryption algorithm. It proves that the following is true: m = [(K-)(K+(m))]
            - Proving the second requirement/property is really difficult. It is hard to show that the private key cannot be derived from the public key
- RSA: Encryption, Decryption
    - To encrypt a message, perform the following computation on the plaintext; the plaintext is referred to as `m`:
      [c = ((m^e) mod n)]
        - The result is ciphertext, `c`
    - To decrypt a message, perform the following computation on ciphertext; the ciphertext is referred to as `c`:
      [m = ((c^d) mod n)]
        - The result is the original plaintext, `m`
    - It can be proved/shown that the ciphertext, ((m^e) mod n), raised to the power of `d`, and then mod `n`, results in the original plaintext, `m`
        - The mathematical property is:
          [m = ((((m^e) mod n)^d) mod n)]
            - An alternative to this property is:
              ((((m^d) mod n)^e) mod n)

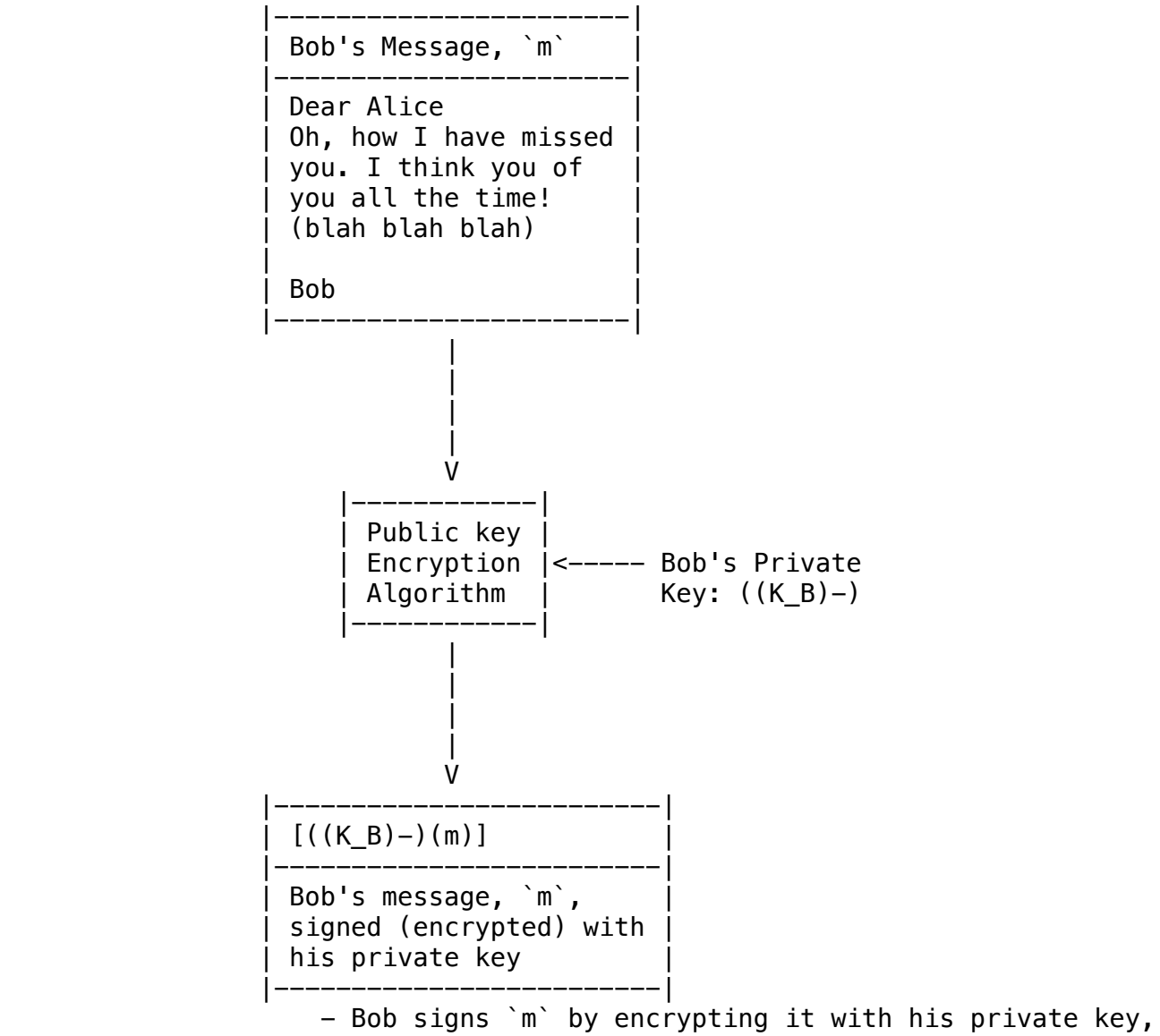
        - These mathematical properties only hold if the procedure used to derive the public and private keys are done according to the encryption algorithm's specification
    - To summarize, the steps for encryption and decryption, for the RSA algorithm are:
        0. Derive the public key, [n,e], and the private key, [n,d] as per RSA algorithm's specification
        1. To encrypt a bit pattern, `m`, compute the following: [c = ((m^e) mod n)]
            - `c` is the remainder when `m^e` is divided by `n`
        2. To decrypt a received bit pattern, `c`, compute the following: [m = ((c^d) mod n)]
            - `m` is the remainder when `c^d` is divided by `n`
- RSA Example
    - Assume that Bob chooses two numbers, `p = 5` and `q = 7` to derive his public and private keys
        - To calculate `n`, take the product of `p` and `q`: [n = (p * q) = 5 * 7 = 35]
        - To calculate `z`, take the product of `p - 1` and `q - 1`: [z = (5 - 1) * (7 - 1) = 24]
        - Pick a value, `e`, such that `e` and `z` are relatively prime. This means that `e` and `z` do not have any common factors
            - i.e. e = 13, 17, etc.
            - Also, `e` needs to be less than `n`
                - i.e. (e < n)
            - In this example, `e` is chosen to be 5
                - This forms the public key: [35, 5]
        - Pick a value `d` such that ((e * d) - 1) is exactly divisible by `z`
            - In this example, the value 29 is selected to satisfy this property
                - This forms the private key: [35, 29]
    - Using the public and private key from above, assume that Alice encrypts data using the public key of the receiver. The process for this is:
        - Take the plaintext, in this example it is the letter 'L' which corresponds to ASCII representation of numeral 12, and compute (m^e). This will result in a large number, 1524832. Then, modular `n` is computed on the large number, and the result is 17. This is the ciphertext, 17
        - At Bob's side, he will take the ciphertext and apply his own secret key, `d`. Bob will compute (c^d), which is a very large number, 48196857210675091509141182522307169 7, and then perform modular `n` on the very large number. The result is 12, which corresponds to the letter 'L' in the ASCII table
            - Bob gets the same value as the original plain text sent by Alice; the letter 'L' was encrypted by Alice, and then decrypted by Bob

    - If you are interested in learning more, you can go over the proof for the RSA algorithm
      - The proof does not require a very deep understanding of number theory or algebra
    - The RSA example, above, is summarized below:
      - Bob chooses p = 5, and q = 7. Then, n = 35 and z = 24
        - Next, `e` is chosen to be 5, and `d` is 29
          - `e` and `z` are relatively prime
          - ((e * d) - 1) is exactly divisible by `z`
      - During encryption:
        1. The plaintext/letter is L
        2. The message is 12
           - The plaintext is 12
        3. (m^e) = 1524832
        4. c = ((m^e) mod n) = 17
           - 17 is the ciphertext
      - During decryption:
        1. The ciphertext is 17
        2. (c^d) = 481968572106750915091411825223071697
        3. m = ((c^d) mod n) = 12
           - This is the plaintext
        4. In the ASCII table, the letter 'L' corresponds to the number 12
- RSA: Another Important Property
    - An important property that RSA has is: ((K-)(K+(m))) = m = ((K+)(K-(m)))
      - On the left side of the equation, a user can encrypt text with a public key, and then decrypt it with a private way
        - In other words, use public key first, followed by private key
        - This property is used to ensure message confidentiality
      - On the right side of the equation, a user can encrypt text with a private key, and then decrypt it with a public key
        - In other words, use private key first, followed by public key
        - This property is used for generating digital signatures
      - Regardless of which key is used for decryption or encryption, the result is the same
- Digital Signatures (1)
    - The purpose of digital signatures is to act as real life signatures. The goal is to be able to verify whether a particular document is actually signed by the intended entity
      - For instance, if John is signing a lease with his landlord, both parties want to ensure that the signatures are coming from the correct entity, and the

signed document that is sent to John from his landlord is in fact sent by the landlord, and not an imposter
        - In real life, people sign a piece of paper. Due to COVID, everyone pastes their signature on a PDF. However, both of these approaches are easy to forge. For instance, someone can forge John's signature or mimic his handwriting. It is also possible for someone to copy-paste a previous signature on an old document to a new document. For instance, if John has access to lease documents from a previous year, then he can copy-paste his landlord's signature onto the current lease document
            - From a security perspective, this is not good, nor is it safe
- Digital signatures are used to achieve verifiability. This means that the recipient should be able to prove that a particular document was signed by a particular person. For instance, Bob signed the lease documents, because his digital signature matches the one attached to the document
- Digital signatures should be unique. John's digital signature should only belong to John, Bob's signature should only belong to Bob, and so forth. This helps with achieving verifiability, and prevents forging of signatures; no one else can sign with someone else's digital signature
    - This property allows people to take documents and digital signatures to court, and show that a particular person signed a particular documen and owes money or assets
        - i.e. Alice digitally signed a contract stipulating that she will pay back Bob's $5000 loan after 2 years; but she has not done so. The court system can force Alice to pay back Bob, based on the legitimacy of the contract and digital signature
- The second half of the RSA equation, [m = (K+)(K-(m))], can be used to generate digital signatures
    - Assume that a document, `m`, needs to be digitally signed so that it has the properties of verifiability and non-forgeability
        - `m` being visible to third parties is not a requirement for this example; confidentiality does not need to be upheld
        - Public key crypto allows a user to sign `m` with their own private key to associate the document to their identity
- To summarize:
    - Digital signatures are analogous to hand-written signatures
    - If a sender digitally signs a document with their private key, then they are establishing themselves as

  the owner/creater of the document
- Digital signatures need to be verifiable and nonforgeable
  - i.e. Alice should be able to prove to a third party that Bob, and no one else, signed the corresponding document

- Digital Signatures (2)
  - i.e. Diagram of Simple Digital Signature

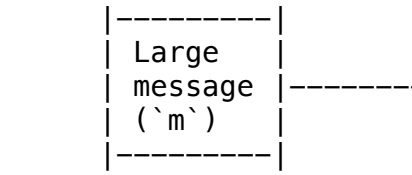


  - Bob signs `m` by encrypting it with his private key, ((K_B)-), which creates a "signed" message referred to as: [((K_B)-)(m)]
- Assume that Bob has sent a message, `m`, to Alice, and Bob wants to attach his own key. Bob can accomplish this by encrypting `m` with his private key, ((K_B)-), and attaching his digital signature, [((K_B)-)(m)], with it
  - Bob will send both `m` and [((K_B)-)(m)] to Alice

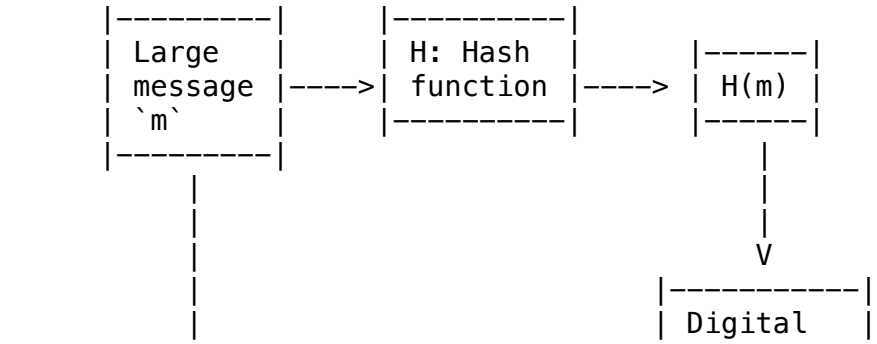
- Digital Signatures (More)

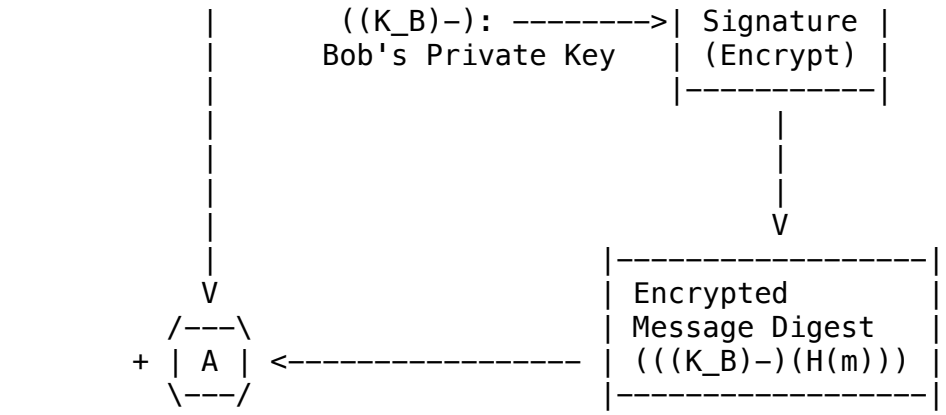
    - Assume Alice receives a message, `m`, and a digital signature, ((K_B)-)(m), from Bob
        - Since Alice has Bob's public key, ((K_B)+), she can verify that `m` is signed by Bob by applying his public key, ((K_B)+), to the digital signature, ((K_B)-)(m)
            - According to the property of the RSA algorithm, the original plaintext should be recovered, because: [((K_B)+)(((K_B)-)(m))] = m
        - Since the document is also sent in the message, Alice can easily verify that the two pieces of information sent by Bob are identical or not. By performing a simple equality check, Alice can verify that Bob is the person who signed `m`, and not a third party, because only Bob knows his own private key
            - If [((K_B)+)(((K_B)-)(m))] = m, then `m` was signed with Bob's private key. If only Bob has access to his private key, then only Bob can sign the sent messages
        - Furthermore, due to the properties of public key crypto, Alice can verify that:
            - Bob signed `m`
            - No one else signed `m`
            - Bob signed `m` and not `m'`
                - If Bob sends `m'`, then the size of `m` that Alice receives, and the signature of `m'` would have some kind of disagreement
    - Digital signatures have a useful property of non-repudiation that allows users to take signatures to court, and prove that the defendant did in fact sign the papers
        - i.e. Alice can take `m` and signature [((K_B)-)(m)] to court and prove that Bob, and only Bob, signed `m`
    - Digital signatures have a slight problem. Applying a public key crypto mechanism to a large document to sign is computationally expensive and the complexity is very high
        - To solve this issue, message digests are used
- Message Digests
    - Instead of using public key crypto to sign large documents, which is computationally expensive and complex, message digests allow users to hash the large message into a much shorter fixed length message, and then the corresponding fixed length message is signed
        - Since the private key only applies to a shorter message, compared to the original message/document, the computation complexity can be reduced
    - Message digests only work if a large message can be hashed to a shorter message
        - In addition, the hash collision rate needs to be extremely low
    - Hashing is a many-to-one operation; it can be used to produce a fixed size message digest

        - For example, a very naive hash mechanism can take all the bytes in a message, add them up together, and obtain the 1's complement sum. The hash value will be one byte
            - This is a very bad approach, because the hash collision rate will be very high
    - i.e. Diagram of Message Digest Procedure

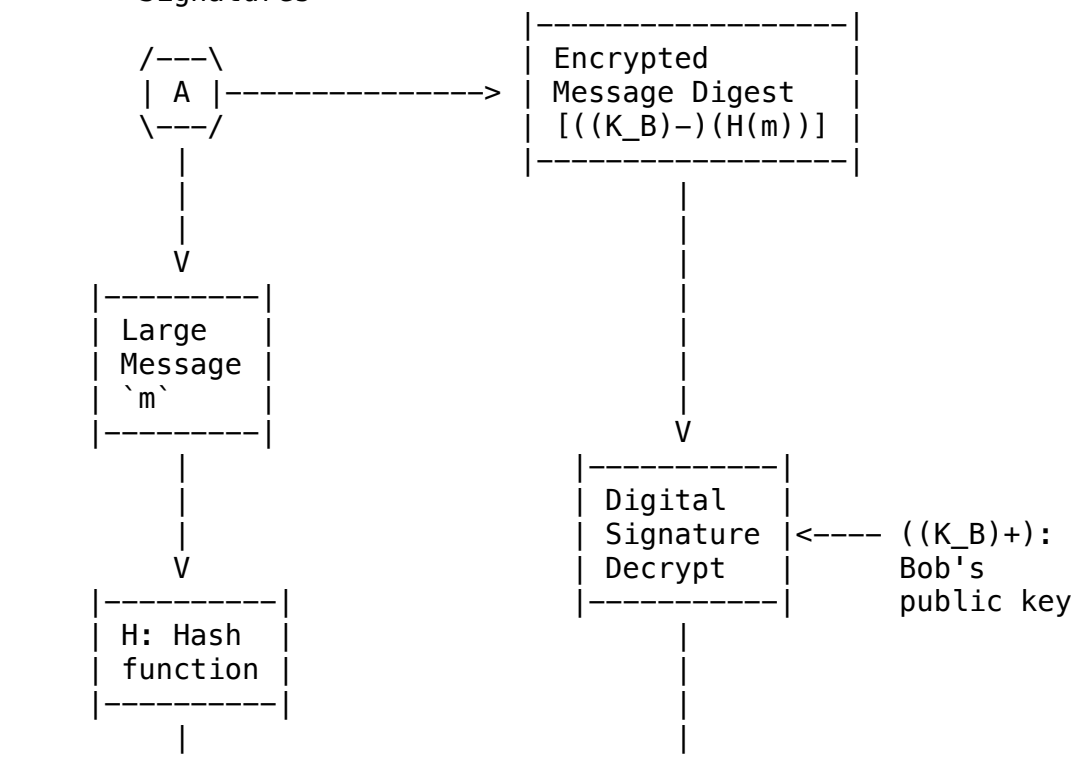


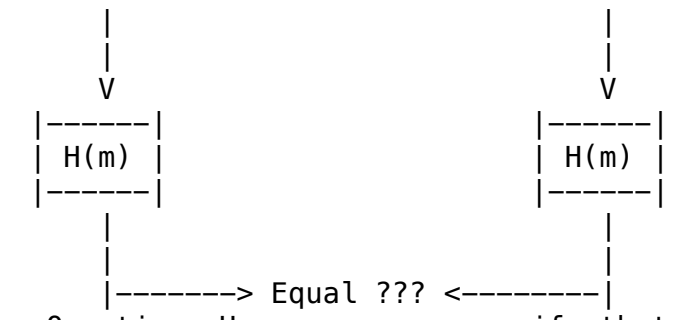
        - A large message is provided as input to a hash function. Upon successfully completing the hashing algorithm, a message digest is created. The message digest is like a unique fingerprint
    - To summarize:
        - It is computationally expensive to use public key crypto for long messages/documents
        - The goal of a message digest is to produce a fixed-length, easy-to-compute digital fingerprint
            - The fingerprint must be unique for each message
            - i.e. Apply hash function 'H' to `m`, to get a fixed size message digest, `H(m)`
        - A hash function must have the following properties:
            - Many to one
            - Produces fixed size message digest, regardless of the size of the input
                - This is a unique fingerprint
            - Given a message digest, `x`, it should be computationally infeasible to find `m` such that `x = H(m)`
- Digital Signature Equals Signed Message Digest
    - By combining the notion of hashing with digital signature, the complexity of using public key crypto on large messages/documents can be greatly reduced
        - In addition, verification of messages can still be carried out
            - i.e. A particular message was signed by Bob, and only Bob
    - i.e. Diagram of Digitally Signing Messages





- Encryption is done at the sender's side
- Assume that Bob wants to send a message to Alice. The message he wants to send is an original document
  - The first step is to apply a hash function, and generate the message digest for the message/document
  - Then, Bob applies his private key to the message digest, producing his digital signature in the process
  - Finally, Bob will send the original message as well as the digital signature to Alice
    - The digital signature is the encrypted form of the message digest
    - Note: Bob does not send the hashed message to Alice. She will compute the hash herself
- i.e. Diagram of Verifying The Integrity of Digitally Signed Signatures





- Question: How can a user verify that an incoming document/message is actually signed by the person indicated?
  - i.e. How can Alice verify that the document/message sent by Bob is actually signed by Bob
- Assume that Alice receives Bob's message/document
  - When Alice receives the message as well as the digitally signed message digest, she will use Bob's public key to decrypt the signed message digest, which contains the hash value of the original message/document. Alice will also apply the same hash function on the large message, in an attempt to generate the same hash value achieved by Bob
    - Alice wants to verify whether the hashed value from the original message and the decrypted message digest are the same or not. If they are the same, then Alice can verify that the message was indeed sent by Bob
    - The assumption here is that both agree upon what kind of hash function to apply so that Alice can verify whether the hashed value from the original message and the decrypted digest message are identical or not
      - If the hash values are the same, then Alice can verify that the message was indeed sent and signed by Bob
- To summarize, Alice verifies the signature and integrity of the digitally signed message