Cybersecurity

Professor: F. d'Amore

A.Y. 2022/23

Table of Contents

List of Figures

List of Tables

1 Introduction and terminologies

2 Symmetric Encryption

Secret key cryptography involves the use of a single key. Given a message (the plaintext) and the key, encryption produces unintelligible data which is about the same length as the plaintext was.
Secret key cryptography is sometimes referred to as **conventional cryptography** or **symmetric cryptography**.
Secret key encryption schemes require that both the party that does the encryption and the party that does the decryption share a secret key. We will discuss two types of secret key encryption schemes:

- **Stream Ciphers**: This uses the key as a seed for a pseudorandom number generator, produces a stream of pseudorandom bits, and $\oplus$ s (bitwise exclusive ors) that stream with the data. Since $\oplus$ is its own inverse, the same computation performs both encryption and decryption.
- **Block Ciphers**: This takes as input a secret key and a plaintext block of fixed size (older ciphers used 64-bit blocks, modern ciphers use 128-bit blocks). It produces a ciphertext block the same size as the plaintext block. To encrypt messages larger than the blocksize, the block cipher is used iteratively with algorithms called *modes of operation*. A block cipher also has a decryption operation that does the reverse computation.

So, block ciphers encrypt data in blocks of set lengths, while stream ciphers do not and instead encrypt plaintext one byte at a time. The two encryption approaches, therefore, vary widely in implementation and use cases.
Here we will have some prerequisite concepts and then we will dive into these two encryption algorithms

2.1 Finite Fields

2.2 Per-Round Keys

2.3 S-boxes and Bit Shuffles

Write S-boxes and Bit Shuffle

2.4 Feistel Cipher

It is very important to make the encryption algorithm revered so that the decryption is possible. One method (used by AES) of having a cipher is to make all components reversible. With DES, the S-boxes are clearly not reversible since they map 6-bit inputs to 4-bit outputs. So instead, DES is designed to be reversible using a clever technique known as a Feistel cipher.

A Feistel cipher builds reversible transformations out of one-way transformations by only working on half the bits of the input value at a time. let us assume a 64-bit input block. (Figure 1) shows both how encryption and decryption work. In a Feistel cipher there is some irreversible component that scrambles the input. We'll call that component the mangler function.

In encryption for round n, the 64-bit input to round n is divided into two 32-bit halves called L_n and R_n. Round n generates as output 32-bit quantities L_{n+1} and R_{n+1}. The concatenation of L_{n+1} and R_{n+1} is the 64-bit output of round n and, if there's another round, the input to round n+1. L_{n+1} is simply R_n. To compute R_{n+1}, do the following. R_n and K_n are input to the mangler function, which takes as input 32 bits of data plus some bits of key to produce a 32-bit output. The 32-bit output of the mangler is XORed with L_n to obtain R_{n+1}.

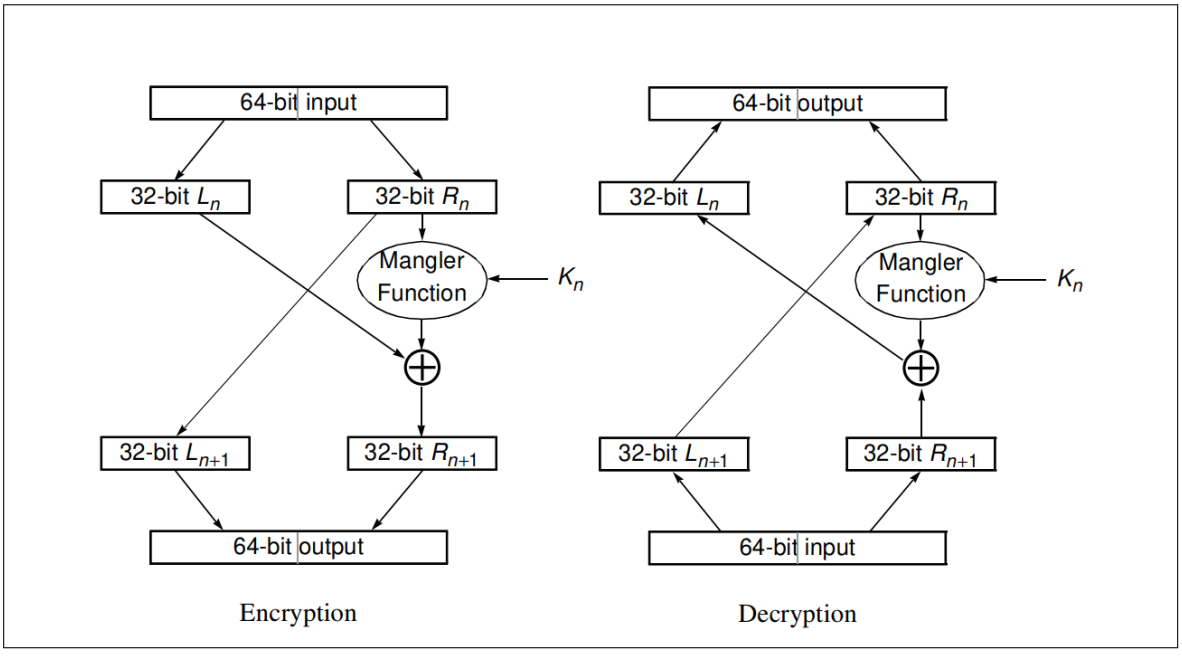

Figure 1: Feistel Cipher

2.5 Padding and Ciphertext Stealing

Padding and ciphertext stealing are techniques used in cryptography to handle messages or plaintext that are not an exact multiple of the block size in block cipher algorithms. Both techniques serve the purpose of ensuring proper encryption and decryption while maintaining the integrity and length of the ciphertext.

Padding:
Padding is the process of adding extra bits or bytes to the plaintext before encryption to ensure it aligns with the block size required by the encryption algorithm. The most commonly used padding scheme is called PKCS#7 (Public Key Cryptography Standard #7).
In PKCS#7 padding, if the plaintext is shorter than the block size, padding bytes are added to the end of the plaintext. The value of each padding byte is set to the number of padding bytes added. For example, if two bytes are needed to reach the block size, both padding bytes will have a value of 0x02.
During decryption, the receiver knows the number of padding bytes based on the value of the last byte in the decrypted block. The padding bytes are then removed to obtain the original plaintext.
Padding is effective when the plaintext length is an exact multiple of the block size, but it introduces additional bytes to the ciphertext, which might impact certain protocols or applications where maintaining the exact plaintext length is crucial.

Ciphertext Stealing:
Ciphertext stealing is an alternative approach to handle the last partial block of plaintext without explicitly padding it. It is commonly used when the plaintext length is not a perfect multiple of the block size.
In ciphertext stealing, the last block of plaintext is divided into two parts: a truncated part and a stolen part.
The truncated part consists of the initial bytes of the last block that can fill up a complete block size. This truncated part is encrypted like any other block and becomes

the second-to-last block of ciphertext.

The stolen part consists of the remaining bytes in the last block. It is then combined with the second-to-last block of ciphertext to create the final block of ciphertext.

During decryption, the last block of ciphertext is decrypted as usual. The stolen part is separated from the decrypted last block, and it is combined with the second-to-last block of plaintext to recover the original message.

Ciphertext stealing allows for encryption and decryption of messages of varying lengths without explicitly padding the last block. It ensures that the ciphertext remains the same length as the original plaintext, which can be desirable in certain scenarios.

Figure (Figure 2) below illustrates the CBC-CS1-Encrypt algorithm for the case that P_n^* is a partial block, i.e., d $\leq$ b. The bolded rectangles contain the inputs and outputs. The dotted rectangles provide alternate representations of two blocks in order to illustrate the role of the "stolen" ciphertext. In particular, the string of the b-d rightmost bits of C_{n-1} , denoted P_{n-1}^{**}, becomes the padding for the input block to the final invocation of the block cipher within the execution of CBC mode. The ciphertext that is returned in Step 5 above omits C_{n-1}^{**}, because it can be recovered from C_n during decryption.

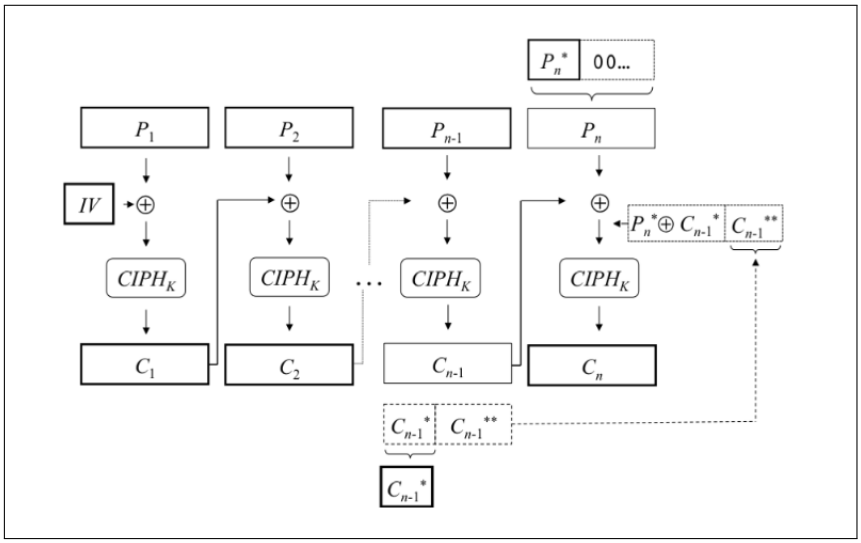

Figure 2: CBC CS1 Encrypt

2.6 Stream Ciphers

Idea: try to simulate one-time pad

A stream cipher encrypts a continuous string of binary digits by applying time-varying transformations on plaintext data. Therefore, this type of encryption works bit-by-bit, using keystreams to generate ciphertext for arbitrary lengths of plain text messages. The cipher combines a key (128/256 bits) and a nonce digit (64-128 bits) to produce the **keystream** — a pseudorandom number XORed with the plaintext to produce ciphertext. While the key and the nonce can be reused, the keystream has to be unique for each encryption iteration to ensure security. Stream encryption ciphers achieve this using feedback shift registers to generate a **unique nonce (number used only once) to create the keystream**.

Encryption schemes that use stream ciphers are less likely to propagate system-wide errors since an error in the translation of one bit does not typically affect the entire plaintext block. Stream encryption also occurs in a **linear, continuous manner**, making it simpler and faster to implement. On the other hand, stream ciphers lack diffusion since each plaintext digit is mapped to one ciphertext output. Additionally, they do not validate authenticity, making them vulnerable to insertions. If hackers break the encryption algorithm, they can insert or modify the encrypted message without detection. Stream ciphers are mainly used to encrypt data in applications where the amount of plain text cannot be determined and in low latency use-cases.

2.7 Types of Stream Ciphers

Stream ciphers fall into two categories:

Synchronous Stream Ciphers:

- In a synchronous stream cipher, the keystream block is generated independently of the previous ciphertext and plaintext messages. This means that **each keystream block is generated based only on the key** and does not depend on any previous blocks.
- The most common stream cipher modes use pseudorandom number generators (PRNGs) to create a string of bits, which is combined with the key to form the keystream.
- The keystream is then XORed with the plaintext to generate the ciphertext.

Self-Synchronizing/Asynchronous Stream Ciphers:

- A self-synchronizing stream cipher, also known as ciphertext autokey, **generates the keystream block as a function of both the symmetric key and the fixed-size (N-bits) previous ciphertext block**.
- By altering the ciphertext, the content of the next keystream is changed. This property allows self-synchronizing ciphers to detect active attacks because any modification to the ciphertext will affect the decryption of subsequent blocks, making it easier to detect tampering.
- Asynchronous stream ciphers also provide limited error propagation. If there is a single-digit error in the ciphertext, it can affect at most N bits (the size of the previous ciphertext block) in the next keystream block

.

2.8 Stream Ciphers in practice

popular encryption schemes that use stream ciphers include:

2.8.1 A5/1

2.8.2 Rivest Cipher (RC4)

2.8.3 Salsa20

2.9 Block Ciphers

Block ciphers convert data in plaintext into ciphertext in fixed-size blocks. The block size generally depends on the encryption scheme and is usually in octaves (64-bit or 128-bit blocks). If the plaintext length is not a multiple of 8, the encryption scheme uses **padding** to ensure complete blocks. For instance, to perform 128-bit encryption on a 150-bit plaintext, the encryption scheme provides two blocks, 1 with 128 bits and one with the 22 bits left. 106 Redundant bits are added to the last block to make the entire block equal to the encryption scheme's ciphertext block size.

While Block ciphers use symmetric keys and algorithms to perform data encryption and decryption, they also require an **initialization vector (IV)** to function. An initialization vector is a pseudorandom or random sequence of characters used to encrypt the first block of characters in the plaintext block. The resultant ciphertext for the first block of characters acts as the initialization vector for the subsequent blocks. Therefore, the symmetric cipher produces a unique ciphertext block for each iteration while the IV is transmitted along with the symmetric key and does not require encryption.

Block encryption algorithms offer **high diffusion**; that is, if a single plaintext block were subjected to multiple encryption iterations, it resulted in a unique ciphertext block for each iteration. This makes the encryption scheme relatively tamper-proof since it is difficult for malicious actors to insert symbols into a data block without detection. On the other hand, block ciphers have a **high error propagation rate** since a bit of change in the original plaintext results in entirely different ciphertext blocks.

2.10 Block Ciphers in practice

2.10.1 Data Encryption Standard (DES)

DES is a symmetric block cipher using 64 bit blocks and 56 bits key.
The choice of using 56 bits for keys in DES was a compromise between security

and practicality. While a longer key would provide stronger security, the designers of DES also needed to consider the limitations of computing technology at the time. The inclusion of 8 parity bits reduced the effective key size to 56 bits, which some experts even then considered inadequate for robust security. However, the decision to use a 56-bit key was likely influenced by a balance between security requirements and the feasibility of implementing and processing longer keys with the available hardware during that era.

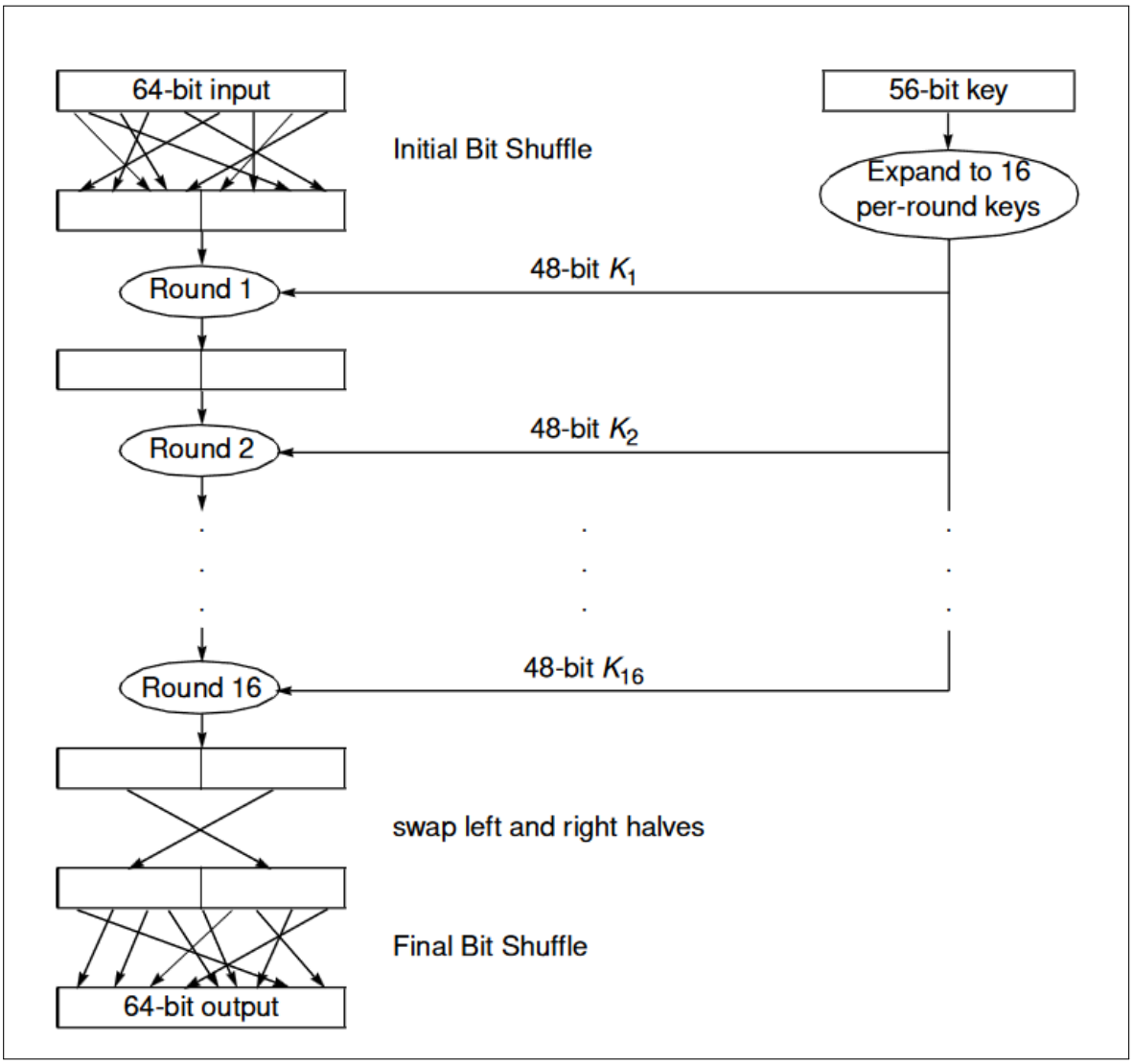

Figure 3: Basic Structure of DE

2.10.2 DES Overview

First, the 64-bit input is subjected to an initial bit-shuffle. Then, 56-bit key is expanded into sixteen 48-bit per-round keys by taking a 48-bit subset of the 56-bit

input key for each of the keys. Each round takes a 64-bit input with a 48-bit key and produces a 64-bit cipher block. After the sixteenth round, the output has its halves swapped and then is subjected to another bit-shuffle which happens to be the inverse of the initial bit-shuffle. This swapping after the final round does not add any cryptographic strength but has a side benefit. As noted in Feistel Ciphers, Swapping the output make the encryption and description identical except for the key schedule

2.10.3 DES Complementary Notes

The Mangler Function

Undesirable Symmetries

DES Variants

Add DES Mangler Function

Add DES Undesirab

Add DES Variantss

2.10.4 Advanced Encryption Standard (AES)

AES is a symmetric block cipher using 128 bit blocks and 128, 192 or 256 bits key with the resulting variants called AES-128, AES-192, and AES-256.

AES is similar to DES in that there is a key expansion algorithm which takes the key as an input and expands it into a bunch of round keys, and the algorithm executes a series of rounds that mangle a plaintext block into a ciphertext block. (Figure 4). With DES each round it takes a 64-bit input with a 48-bit key and outputs 64-bit that (along with the next round key) is fed to the next round. With AES, each round takes a 128-bit input and a 128-bit key and produces a 128-bit output which is the input to the next round. Unlike DES, AES is not Feistel cipher. (The Feistel cipher structure is characterized by the division of the plaintext into two halves and the repeated application of a round function that involves both halves. Read more here: 2.4)

In accordance to do as many rounds as needed to make the exhaustive search cheaper than any other form of cryptanalysis, AES does more round with bigger keys. AES-128 has 10 rounds, AES-196 has 12 rounds and AES-256 has 14 rounds. If AES had a 64-bit version, it would have eight rounds, which would be comparable to the sixteen rounds in DES.

Another diffference between DES and AES is that DES operates on bits where AES

operates on octets (bytes)

In AES, the key (128, 192, or 256 bits) is expanded into a series of 128-bit round keys, where there is one more round key than there are rounds. AES encryption or decryption consists of $\oplus$ing a round key into the intermediate value at the beginning of the process, at the end of the process, and between each pair of rounds.

In terms of Substitution and Permutation and cryptographic transformations, DES relies on predefined tables for S-boxes and P-boxes, which may appear arbitrary without understanding their rationale. In contrast, AES provides simpler mathematical formulas for substitutions and permutations, openly stating the reasons behind their design choices.

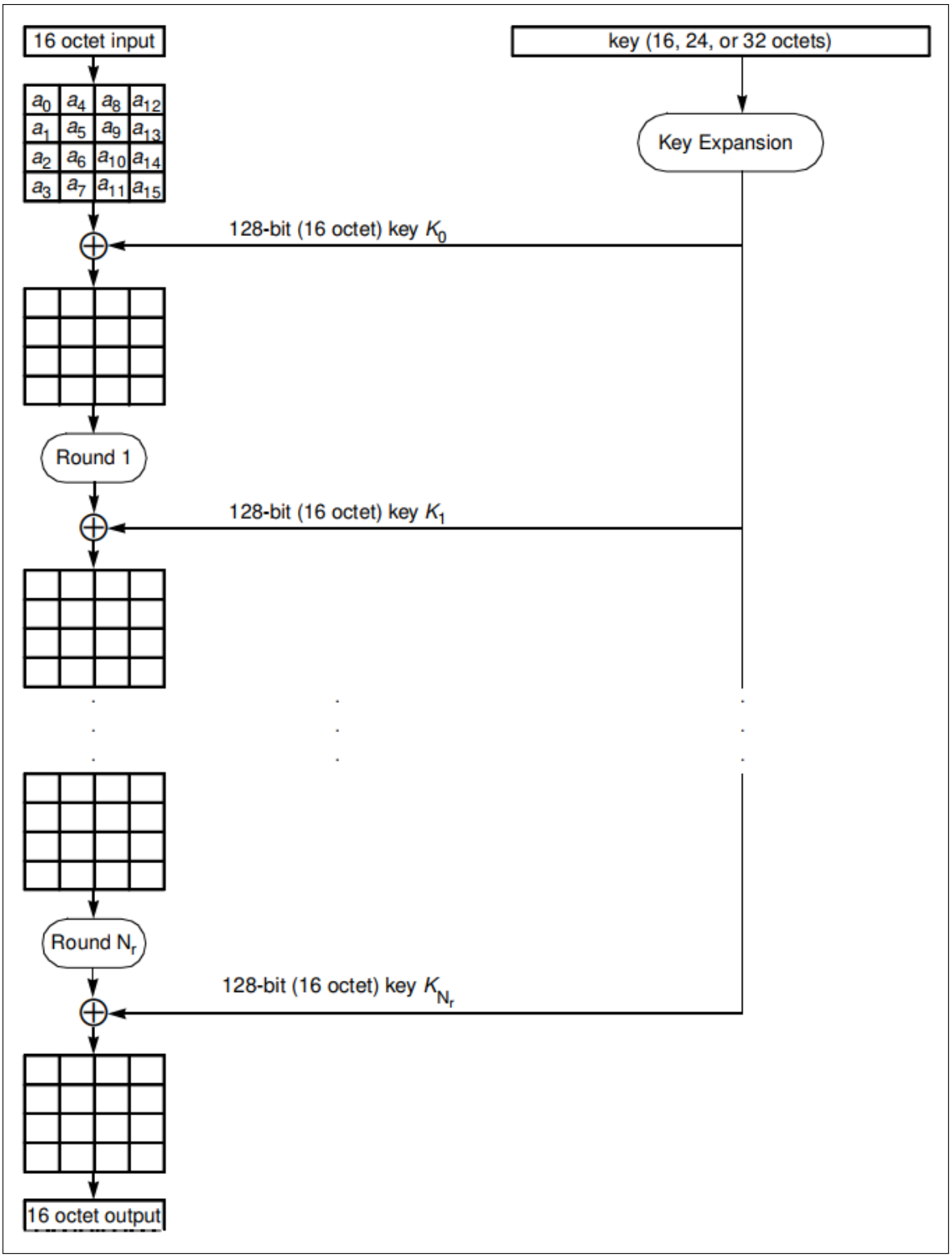

Figure 4: Basic Structure of AE

2.10.5 AES Complementary Notes

(AES's S-box)

Key expansion

Add AES Box

Add AES Key expansion

2.11 Block Ciphers Modes of Operations

Block ciphers operate on fixed-length blocks, typically 64 or 128 bits. The reason for using fixed-length blocks is twofold: first, messages can have varying lengths, and second, encrypting the same plaintext with the same key should always produce the same output.

To address the need for encrypting messages of **arbitrary** lengths, several modes of operation have been invented. These modes allow block ciphers to provide confidentiality for messages of any length. These modes define how the block cipher is applied to multiple blocks and how they are combined to encrypt or decrypt the entire message.

2.11.1 Electronic Cook Book (ECB)

a full 128 bits), and encrypt each block with the secret key (Figure 5). The other side receives the encrypted blocks and decrypts each block in turn to recover the original message (Figure 6).

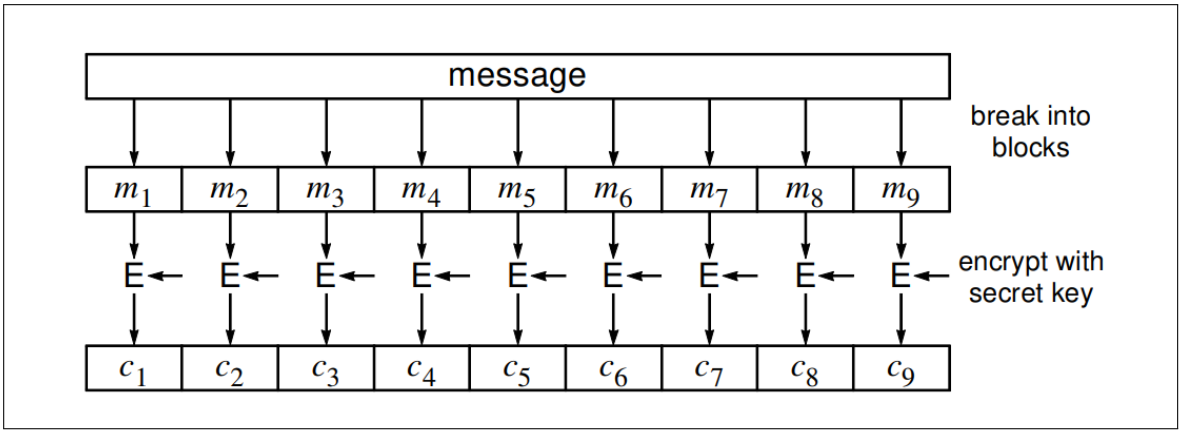

Figure 5: Electronic Code Book Encryption

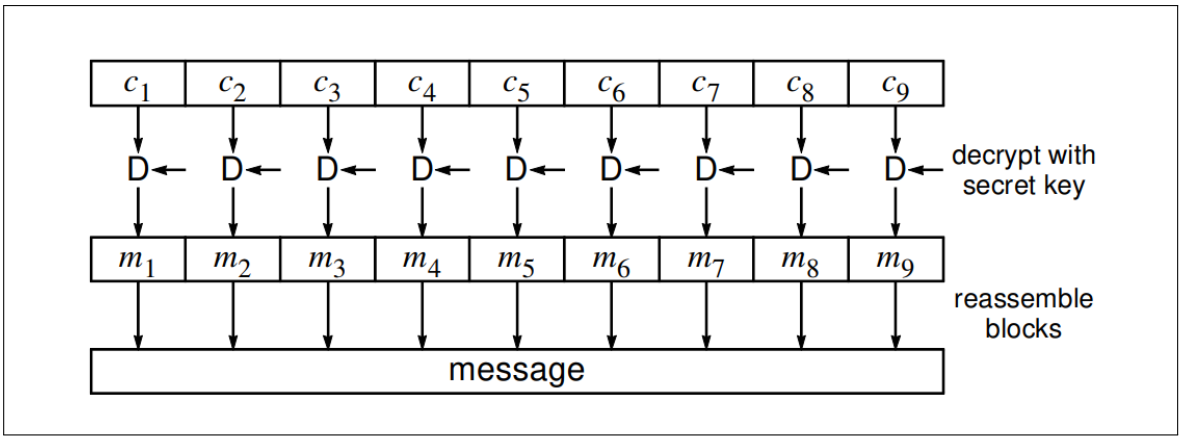

Figure 6: Electronic Code Book Decryption

ECB has two serious flaws. Patterns in the ciphertext, such as repeated blocks, leak information, and nothing prevents someone from rearranging, deleting, modifying, or duplicating blocks.

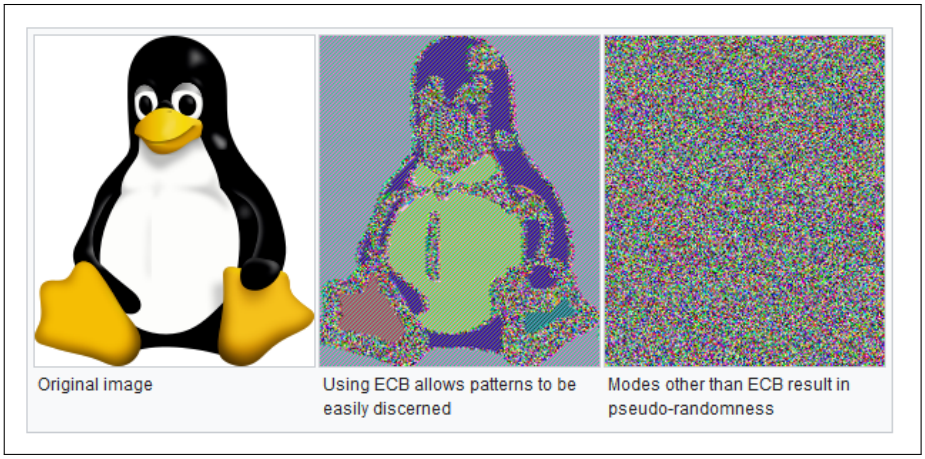

Figure 7: ECB Lack of Diffusion

2.11.2 Cipher Block Chaining (CBC)

CBC generates its own "random numbers" for all but the first block. It uses c_i as r_{i+1}. In other words, it takes the previous block of ciphertext and uses that as the "random number" that will be $\oplus$'d into the next plaintext block. To avoid leaking the information that two messages encrypted with the same key have the same first plaintext blocks, CBC selects one random number that gets $\oplus$'d into the first block of plaintext and transmits it along with the data.

This initial random number is known as an **IV (initialization vector)** A randomly chosen IV guarantees that even if the same message is sent repeatedly, the ciphertext

will be completely different each time.

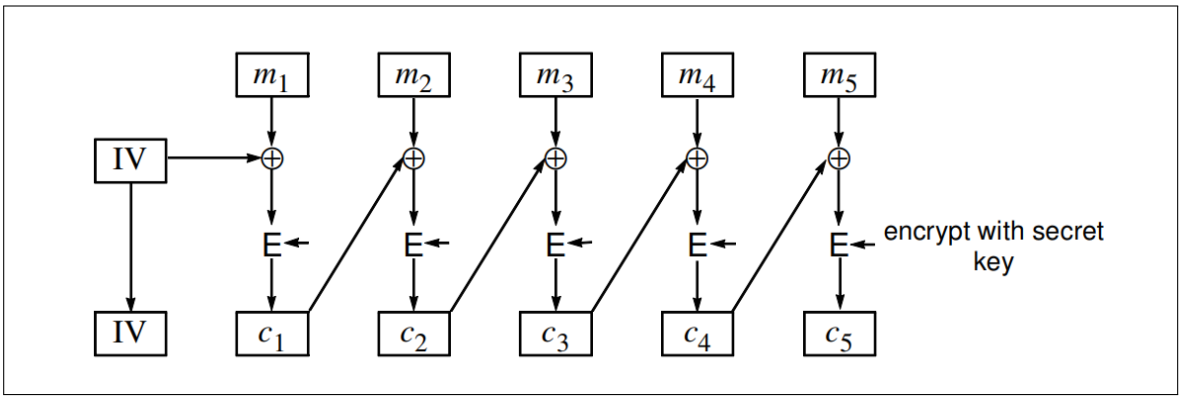

Figure 8: Cipher Block Chaining Encryption

Properties of CBC:

- What would happen if they changed a block of the ciphertext, say, the value of ciphertext block c_n? Changing c_n has a predictable effect on m_{n+1} because c_n gets $\oplus$'d with the decrypted c_{n+1} to yield m_{n+1} For instance, changing bit 3 of c_n changes bit 3 of m_{n+1}. Modifying c_n also garbles block m_n to some unpredictable value.
- Asynchronous stream cipher
- Conceals plaintext patterns
- Plaintext cannot be easily manipulated
- No parallel implementation known for encryption
- message must be padded to a multiple of the cipher block size (or we may use Ciphertext Stealing)
- a plaintext can be recovered from just two adjacent blocks of ciphertext
  as a consequence, decryption can be parallelized. usually a message is encrypted once, but decrypted many times

2.11.3 XEX (XOR Encrypt XOR)

Write dow
XEX

2.11.4 XTS (XEX with Ciphertext Stealing)

XTS is a clever variant of XEX that encrypts multiblock messages that need not be a multiple of the cryptographic blocksize while still keeping the ciphertext the same size as the plaintext. XTS accomplishes this goal (of being length-preserving despite a message not being a multiple of the cryptographic blocksize) through a very clever but somewhat complicated trick known as ciphertext stealing.

Ciphertext stealing pads the final block m_n with as many of the bits of the previous block of ciphertext (c_{n-1}) as necessary to make block m_n be full sized (in our example, $128 - 93 = 35$ bits need to be stolen from c_{n-1}). The padded block m_n is then encrypted. But the ciphertext is now longer than the plaintext, since the last block of ciphertext is now a full-sized block. To solve that problem, we swap ciphertext blocks c_n and c_{n-1} and truncate the final ciphertext block (which was the encryption of block m_{n-1}) to be the size of the original block m_n. In our example, where the original block mn was 93 bits long, the final ciphertext block will be 93 bits. Now the ciphertext is the same size as the message, but how do we decrypt either of the last two blocks? To obtain block m_n, decrypt c_{n-1} (using implicit IVs and Bmods associated with block m_n). The result will be plaintext m_n with appended padding, but you need to know how much of the result is message and how much is padding. The size of plaintext block m_n will be the size of the final ciphertext block (in our example, 93 bits). The padding (the remaining 128-93=35 bits) is stolen ciphertext from c_{n-1}.

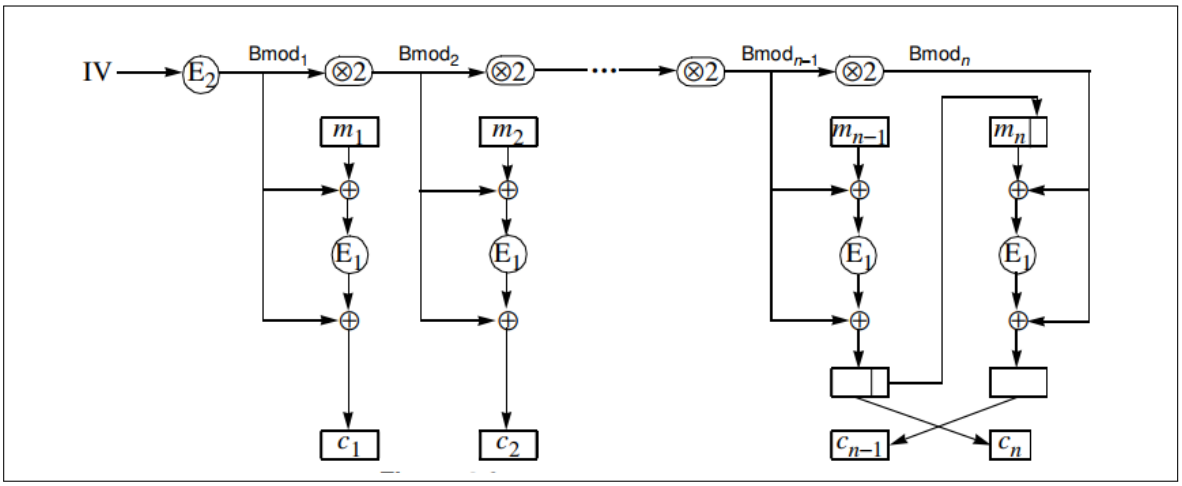

Figure 9: XTS Mode Decryption

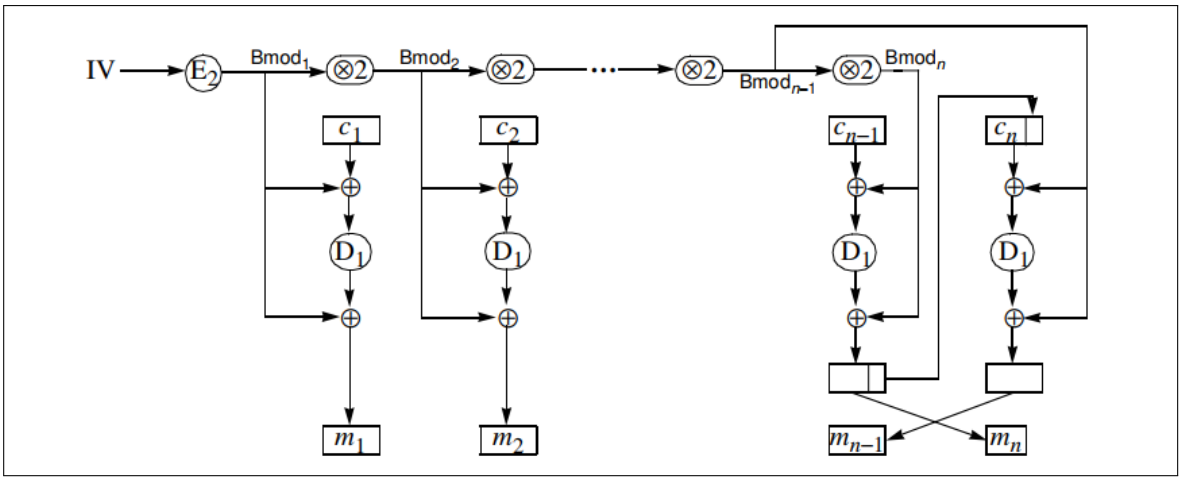

Figure 10: XTS Mode Decryption

2.11.5 Cipher Feedback (CFB)

CFB (short for cipher feedback) is an AES block cipher mode like CBC, makes a block cipher into an asynchronous stream cipher (i.e., supports some re-synchronizing after error, if input to encryptor is given through a shift-register) in the sense that for the encryption of a block, B_i, the cipher of the previous block, C_{i-1} is required. CFB also makes use of an initialization vector like CBC. The main difference is that in CFB, the ciphertext block of the previous block is encrypted first and then XOR-ed with the block in focus.

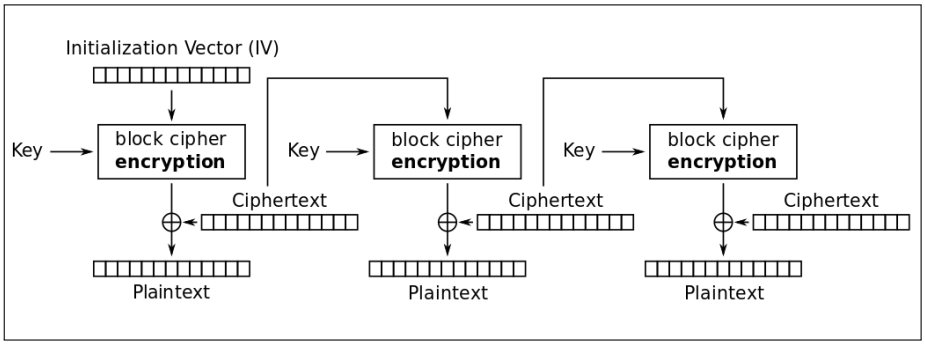

Figure 11: Cipher Feedback (CFB)

2.11.6 Output Feedback (OFB)

The output feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property

allows many error-correcting codes to function normally even when applied before encryption.

Properties of OFB:

- Synchronous stream cipher
- Errors in ciphertext do not propagate
- Pre-processing is possible
- Conceals plaintext patterns
- No parallel implementation known
- Active attacks by manipulating plaintext are possible

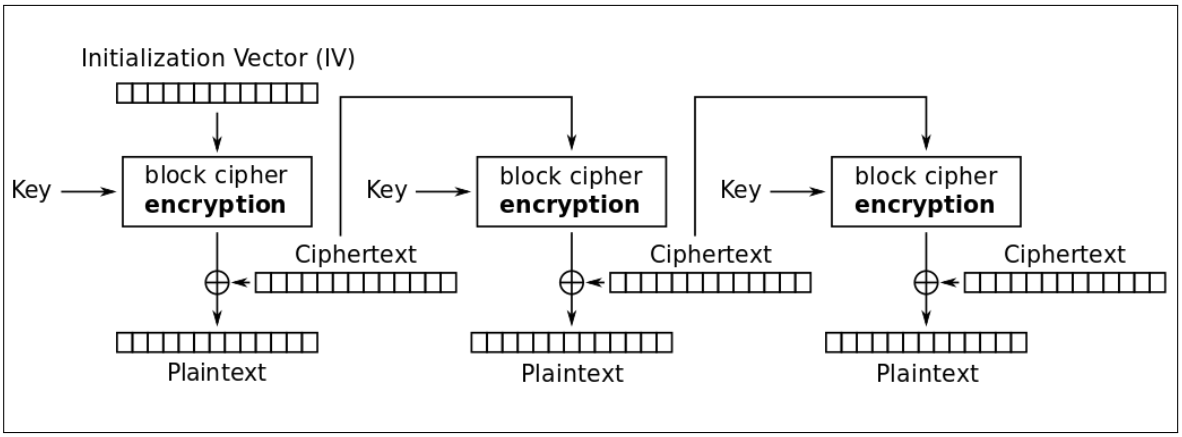

Figure 12: Feedback (OFB)

2.11.7 COUNTER MODE (CTR)

also known as Integer Counter Mode (ICM) and Segmented Integer Counter (SIC) mode

- turns a block cipher into a stream cipher: it generates the next keystream block by encrypting successive values of a ”counter”
- counter can be any function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular.

- the usage of a simple deterministic input function raised controversial discussions
- has similar characteristics to OFB, but also allows a random access property during decryption
- well suited to operation on a multi-processor machine where blocks can be encrypted in parallel

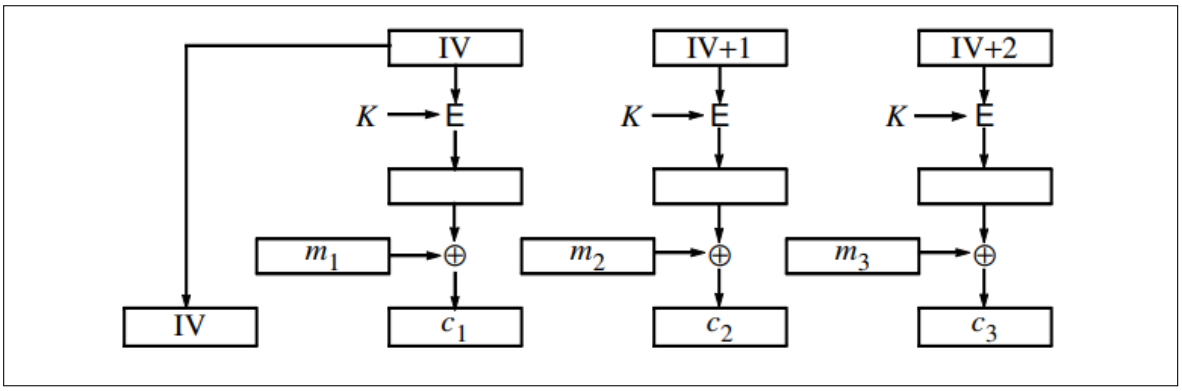

Figure 13: Counter Mode (CTR)

2.12 Ensuring Privacy and Integrity Together

2.12.1 CCM (Counter with CBC-MAC)

2.12.2 GCM (Galois/Counter Mode)

3 Data Integrity

A secret key system can be used to generate a cryptographic integrity check known as a MAC (Message Authentication Code). MACs are used to protect the data from modification in transit.
In this chapter, we'll focus on MACs based on secret key encryption functions and MACs based on hash functions.

3.1 Definition

- *Authentication Algorithm* (A): A is the algorithm used to generate an authentication tag for a given message using a secret authentication key (k). It takes the message (m) as input and produces the authentication tag $A_k(m)$.
- *Verification Algorithm* (V): V is the algorithm used to verify the authenticity and integrity of a message. It takes the received message (m) and its corresponding authentication tag $(A_k(m))$ as input and outputs either *accept* or *reject* based on whether the authentication tag is valid or not.
- *Authentication Key* (k): The authentication key (k) is a secret key shared between Alice and Bob. It is used by the authentication algorithm (A) to generate the authentication tag and by the verification algorithm (V) to validate the tag.
- *Message Space (usually binary strings)*: The message space refers to the set of possible messages that can be authenticated using CBC-MAC. Typically, these messages are represented as binary strings.
- *Message Format*: Every message exchanged between Alice and Bob is a pair $(m, A_k(m))$, where m is the actual message and $A_k(m)$ is its corresponding authentication tag generated by applying the authentication algorithm (A) with the authentication key (k).

3.2 MAC based on CBC Mode Encryption

CBC-MAC computes a MAC of a message using key K. It encrypts the message in CBC mode, using key K, and uses the last block (called the **residue**, see (Figure

14)) as the MAC for the message. If an attacker modifies any portion of a message, the residue will no longer be the correct value (except with probability 1 in 2^{128}, assuming 128-bit blocks).

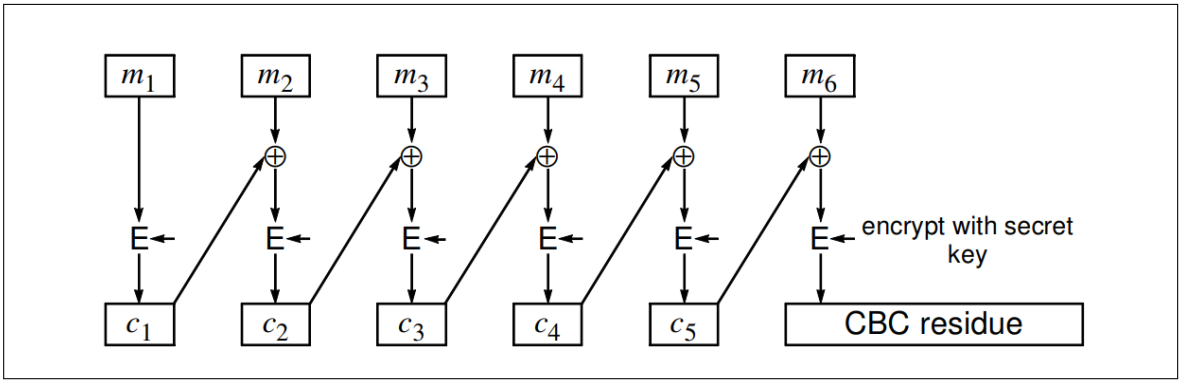

Figure 14: Cipher Block Chaining Residue

3.3 Vulnerabilities of CBC-MAC

3.3.1 Initialization Vector or IV

When we are working on CBC-MAC or Cipher Block Chaining Message Authentication Code, we generally use the Initialization Vector or IV as zero.
When we use the IV or Initialization Vector as zero, a problem arises. Let's say there are two known messages named 'msg1' and 'msg2'. These two messages will generate two signatures called 'sig1' and 'sig2' independently. Finally,

- E (msg1 XOR 0) = sig1
- E (msg2 XOR 0) = sig2

Now, a message which consists of msg1 and msg2 will generate two signatures. Let's name the concatenated message as msg3 and the signals generated as sig31 and sig32.

- E (msg1 XOR 0) = sig31 = sig1
- E (msg2 XOR sig1) = sig32

We can calculate this without even knowing the key of the encryption.

3.3.2 Fixed and Variable-length message

If the block cipher used is secure (meaning that it is a pseudorandom permutation), then CBC-MAC is secure for fixed-length messages. However, by itself, it is not secure for variable-length messages. Thus, any single key must only be used for messages of a fixed and known length. This is because an attacker who knows the correct authentication tag (i.e. CBC-MAC) pairs for two messages (m,t) and (m',t') can generate a third message m" whose CBC-MAC will also be t'. This is simply done by XORing the first block of m' with t and then concatenating m with this modified m'; i.e., by making $m" = m \parallel (m'_1 \oplus t) \parallel m'_2 \parallel \ldots \parallel m'_x$
There are three main ways of modifying CBC-MAC so that it is secure for variable length messages:

1. Input-length key separation
2. Length-prepending
3. Encrypt last block.
4. Use HMACs or CMACs.

3.4 MAC based on cryptographic hash

A hash function inputs an arbitrary-sized bitstring and outputs a fixed-size bitstring, ideally so that all output values are equally likely. A cryptographic hash (also known as a message digest) has some extra security properties:
Preimage Resistance: It should be computationally infeasible to find a message that has a given pre-specified hash.
Collision Resistance: It should be computationally infeasible to find two messages that have the same hash.
Second Preimage Resistance: It should be computationally infeasible to find a second message that has the same hash as a given message.

Strong collision resistance: A hash function is said to have strong collision resistance if it is computationally infeasible to find **any two distinct inputs** that produce the

same hash output. In other words, given a hash value h, it is difficult to find any two different messages m1 and m2 such that hash(m1) = hash(m2).
Weak collision resistance (also known as second preimage resistance): A hash function is said to have weak collision resistance if it is computationally infeasible **to find a second input message that produces the same hash output as a given fixed input message.** In simpler terms, it is difficult to find a different message m2 such that hash(m1) = hash(m2) for a known message m1.

The strong collision resistance property implies the weak collision resistance property (strong $\rightarrow$ weak). For the proof we show not Weak $\rightarrow$ not Strong If a hash function is strongly collision resistant, it automatically implies that it is weakly collision resistant. If it were possible to find a second preimage for a fixed input message, it would effectively mean finding a collision between the fixed input message and the second preimage, contradicting the strong collision resistance assumption. Therefore, a hash function that exhibits strong collision resistance will inherently exhibit weak collision resistance as well.
As an example, the mod s function (% s), where s is a large random prime number, is not suitable for cryptographic purposes due to the following reasons:
a) Predictability: The mod s function is deterministic and has a predictable output. Given an input x, the output of x % s will always be the remainder when x is divided by s. This predictability makes it vulnerable to attacks, as an attacker can calculate the output for various inputs and try to identify patterns or predict future outputs.
b) Lack of diffusion: Cryptographic hash functions need to exhibit the property of diffusion, which means that a small change in the input should result in a significantly different output. However, the mod s function does not provide diffusion. If two inputs x1 and x2 are very close to each other (e.g., x2 = x1 + 1), their outputs x1 % s and x2 % s will also be very close to each other, thus lacking the necessary diffusion property.
Limited output space: The output of the mod s function is constrained to the range from 0 to s-1, where s is the prime number. This limited output space makes it

vulnerable to brute-force attacks, where an attacker can exhaustively try different inputs until finding a collision or a preimage.

Lack of resistance to mathematical attacks: The mod s function is vulnerable to various mathematical attacks, such as modular arithmetic properties or number-theoretic attacks. These attacks exploit the specific properties of modular arithmetic and the structure of prime numbers, making the function unsuitable for cryptographic purposes.

3.5 The Birthday Paradox

If there are 23 or more people in a room, the odds are better than 50% that two of them will have the same birthday. Analyzing this parlor trick can give us some insight into cryptography.

Let's do this in a slightly more general way. Let's assume n inputs (which would be humans in the birthday example), k possible outputs, and an unpredictable mapping from input to output. With n inputs, there are $\frac{n(n-1)}{2}$ pairs of inputs. For each pair, there's a probability of $\frac{1}{k}$ of both inputs producing the same output value. Therefore, you'll need about $\frac{k}{2}$ pairs in order for the probability to be about $\frac{1}{2}$ that you'll find a matching pair. That means that if n is greater than $\frac{k}{2}$, there's a good chance of finding a matching pair.

3.6 MACs based on Hash Functions

3.7 SHA-1

3.8 Authenticated Encryption (AE)

Authenticated Encryption (AE) is an encryption scheme which simultaneously assures the data confidentiality (also known as privacy: encrypted message is impossible to understand without the knowledge of a secret key) and authenticity (in other words, it is unforgeable: the encrypted message includes an authentication tag that the sender can calculate only if she possesses the secret key)
Many (but not all) AE schemes allow the message to contain ”associated data” (AD) which is not made confidential, but its integrity is protected (i.e., it is readable, but tampering with it will be detected). A typical example is the header of a network packet that contains its destination address. To properly route the packet, all intermediate nodes in the message path need to know the destination, but for security reasons they cannot possess the secret key. Schemes that allow associated data provide authenticated encryption with associated data, or AEAD.

3.8.1 Encrypt-then-MAC (EtM)

The plaintext is first encrypted, then a MAC is produced based on the resulting ciphertext. The ciphertext and its MAC are sent together. Used in, e.g., IPsec. This is the only method which can reach the highest definition of security in AE, but this can only be achieved when the MAC used is ”strongly unforgeable”. Note that key separation is mandatory (distinct keys must be used for encryption and for the keyed hash), otherwise it is potentially insecure depending on the specific encryption method and hash function used.

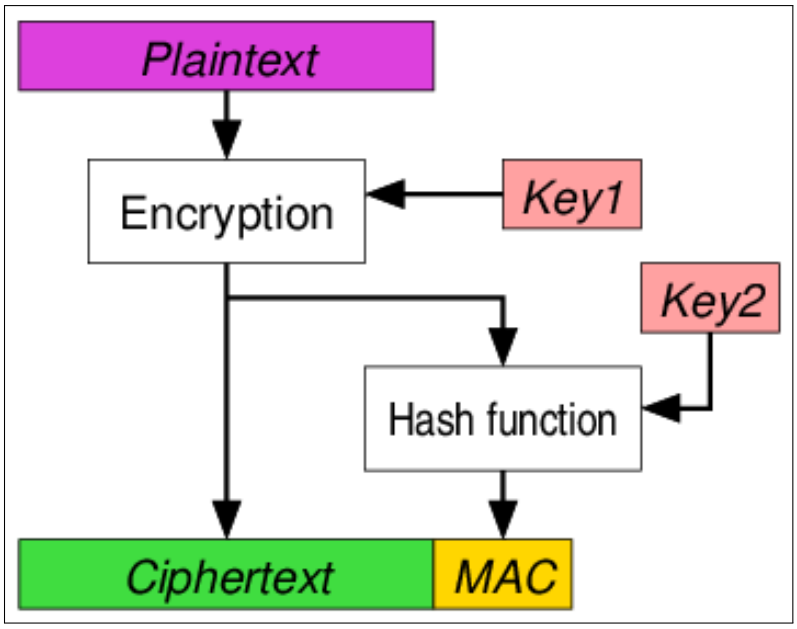

Figure 15: Authenticated Encryption EtM

3.8.2 Encrypt-and-MAC (E&M)

A MAC is produced based on the plaintext, and the plaintext is encrypted without the MAC. The plaintext's MAC and the ciphertext are sent together. Used in, e.g., SSH. Even though the E&M approach has not been proved to be strongly unforgeable in itself, it is possible to apply some minor modifications to SSH to make it strongly unforgeable despite the approach.

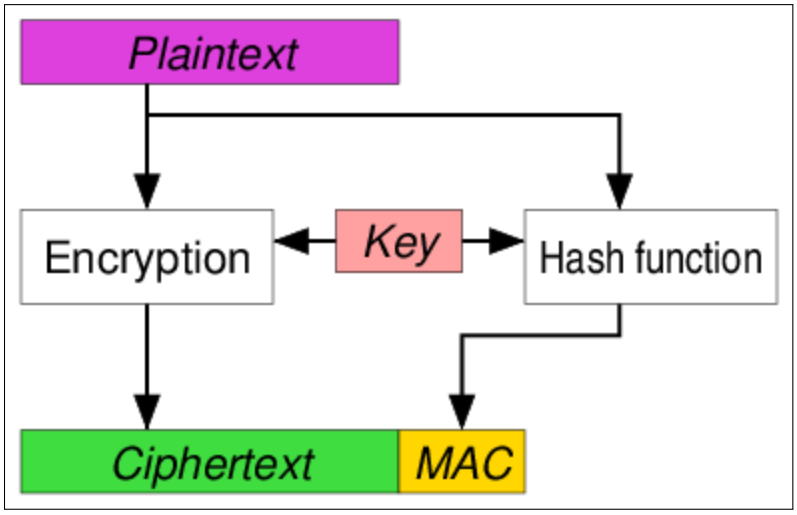

Figure 16: Authenticated Encryption EaM

3.8.3 MAC-then-Encrypt (MtE)

A MAC is produced based on the plaintext, then the plaintext and MAC are together encrypted to produce a ciphertext based on both. The ciphertext (containing an encrypted MAC) is sent. AEAD is used in SSL/TLS. Even though the MtE approach has not been proven to be strongly unforgeable in itself,[15] the SSL/TLS implementation has been proven to be strongly unforgeable by Krawczyk who showed that SSL/TLS was, in fact, secure because of the encoding used alongside the MtE mechanism. Despite the theoretical security, deeper analysis of SSL/TLS modeled the protection as MAC-then-pad-then-encrypt, i.e. the plaintext is first padded to the block size of the encryption function. Padding errors often result in the detectable errors on the recipient's side, which in turn lead to padding oracle attacks, such as Lucky Thirteen.

AEAD is a cryptographic construction that combines both encryption and authentication in a single operation.

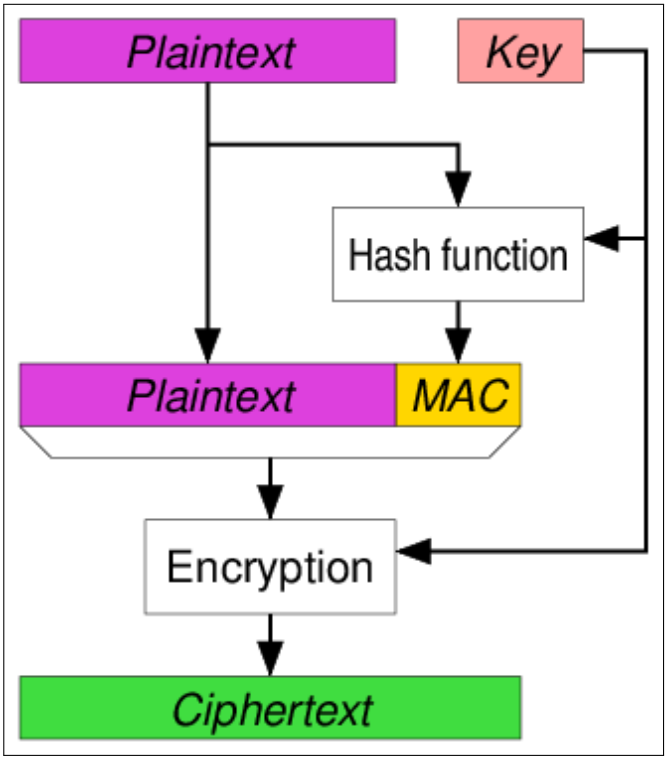

Figure 17: Authenticated Encryption MtE

4 Public Key Cryptography

In this chapter we'll have asymmetric techniques that secret and public keys are used instead of a single shared key. In this chapter we'll have:

- RSA, which does encryption and digital signatures
- ElGamal and DSA and ECDSA (elliptic curve DSA), which do digital signatures
- Diffie-Hellman and ECDH (elliptic curve Diffie-Hellman), which establish a shared secret and can also do encryption

The thing that all public key algorithms have in common is the concept that a principal has a pair of related quantities, one private and one public

4.1 RSA

RSA is named after its inventors, Rivest, Shamir, and Adleman. The real premise behind RSA's security is the assumption that factoring a big number is hard.

4.1.1 Example 1

First, we pick two big prime numbers p and q.

1. $p = 2, q = 7$

   Then we will calculate the multiplication of q and p.

2. $p \times q = 14$

   Now, we calculate ϕ function: $\phi = (p-1)(q-1)$

3. $\phi = 1 \times 6 = 6$

   Now, we need to choose a value e having the following conditions:

   - $1 < e < \phi(n)$
   - e should be prime with respect to n and $\phi(n)$

4. $e = 5$

   Finally, we choose a number under the following condition: $d \times e(mod\phi(n)) = 1$

5. $d = 11$

   Finally, we have the following numbers:n = 14 ,e = 5 ,d = 11

   Based on RSA, we use e and n as the public-key and e and d as the private-key.

Now, assume we want to send the number 2 to our friend. Having our fiend's public-key which has been sent to use before (5,14) we first calculate
$2^5 mod 14 = 4$
Now, our encrypted message is 4. We send 4 to our friend.
Our friend, with his private-key (11, 14) decrypts the message:
$4^{11} mod 14 = 2$

4.1.2 Example 2

4.1.3 Properties

- key length is variable

  The longer, the higher security (4096 bits is common)

- Block size also variable

  but $|plaintext| \leq |N|$ (in practice, for avoiding weak cases, $|plaintext| < |N|$)

  $|plaintext| = |N|$

- Slower than DES

  not used in practice for encrypting long messages

  mostly used to encrypt a symmetric key, then used for encrypting the message

4.2 Diffie-Hellman

Diffie-Hellman allows two individuals (Alice and Bob) to agree on a shared key even though they can only exchange messages in public.

he security of Diffie-Hellman depends on the difficulty of solving the discrete log problem, which can be informally stated as, "If you know g, p, and $g^x modp$, what is x?" In other words, what exponent did you have to raise g to, $modp$, to get g^x?

So, there are integers p and g, where p is a large prime (say 2048 bits) and g is a number less than p with some restrictions that aren't too important for a basic understanding of the algorithm.

The values p and g are known beforehand and can be publicly known, or Alice and Bob can negotiate with each other across the crowded room.

Once Alice and Bob agree on a p and g, each chooses a large, say, 512-bit number at random and keeps it secret. Let's call Alice's private Diffie-Hellman key a and Bob's private Diffie-Hellman key b. Each raises g to their private Diffie-Hellman number, $modp$, resulting in their public Diffie-Hellman value. So Alice computes $g^a modp$ and Bob computes $g^b modp$, and each informs the other. Finally, each raises the other side's public value to their own private value.

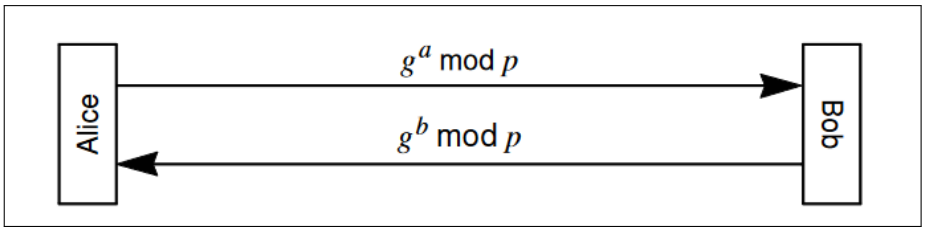

Figure 18: Diffie-Hellman Exchange

Alice raises $g^b modp$ to her private number a. Bob raises $g^a modp$ to his private number b. So Alice computes $(g^b modp)^a = g^{ba} modp$. Bob computes $(g^a modp)^b = g^{ab} modp$. And, of course, $g^{ba} modp = g^{ab} modp$.

Without knowing either Alice's private number a or Bob's private number b, nobody else can calculate $g^{ab} modp$ in a reasonable amount of time even though they can see $g^a modp$ and $g^b modp$.

4.2.1 MITM (Meddler-in-the-Middle) Attack

Diffie-Hellman alone does not prevent MITM attacks, if we assume that Alice and Bob do not have credentials (such as public keys) for each other. As a result, this form of Diffie-Hellman is only secure against passive attacks where the intruder just watches the encrypted messages.

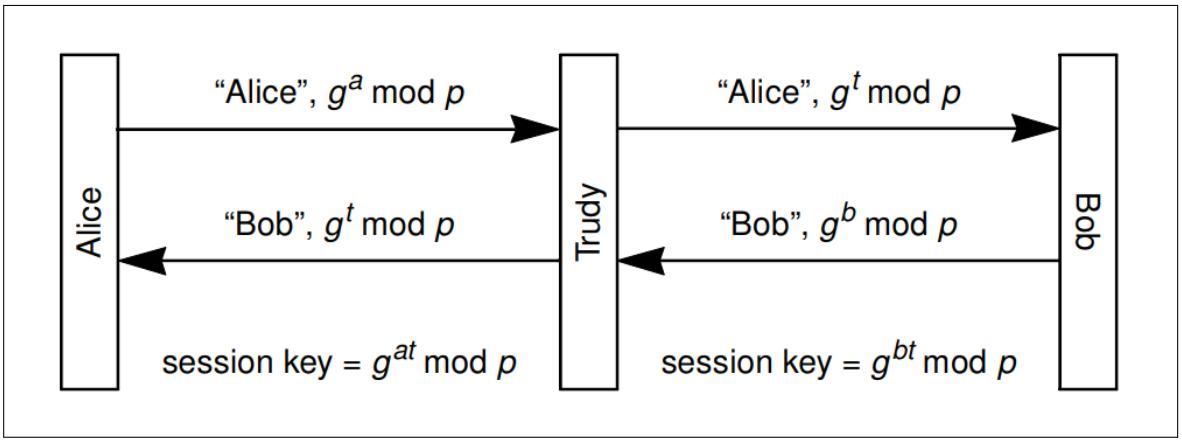

Figure 19: Diffie-Hellman with Trudy in the Middle

4.2.2 Defenses Against MITM Attack

- Having a personal permanent Diffie-Hellman value and do a handshake first.
- Use any authenticated Diffie-Hellman exchange techniques
- Use channel bindings

4.2.3 Safe Primes and the Small-Subgroup Attack

4.2.4 DF Properties

- DH is at most as strong as DL in Z_p^*.
- Formal equivalence unknown, though some partial results known.
- Despite 25 years of effort, still considered secure to date.
- Secret integers a and b are discarded at the end of the session, hence Diffie-Hellman key exchange achieves perfect forward secrecy, because no long-term private keying material exists to be disclosed
- Computation time is $O(\log^3 p)$

4.2.5 ElGamal Signatures

4.3 Digital Signatures

4.4 How Secure Are RSA and Diffie-Hellman?

4.5 Elliptic Curve Cryptography (ECC)

5 Cryptographically Secure Pseudo-Random Number Generators

6 Authentication

6.1 Passwords

6.1.1 Lamport's Hash

6.1.2 Strong Password Protocols

6.2 Biometric

6.3 Authentication by symmetric key

6.3.1 One way authentication using nonce (challenge-response)

ONE-WAY AUTHENTICATION OF ALICE

arg1

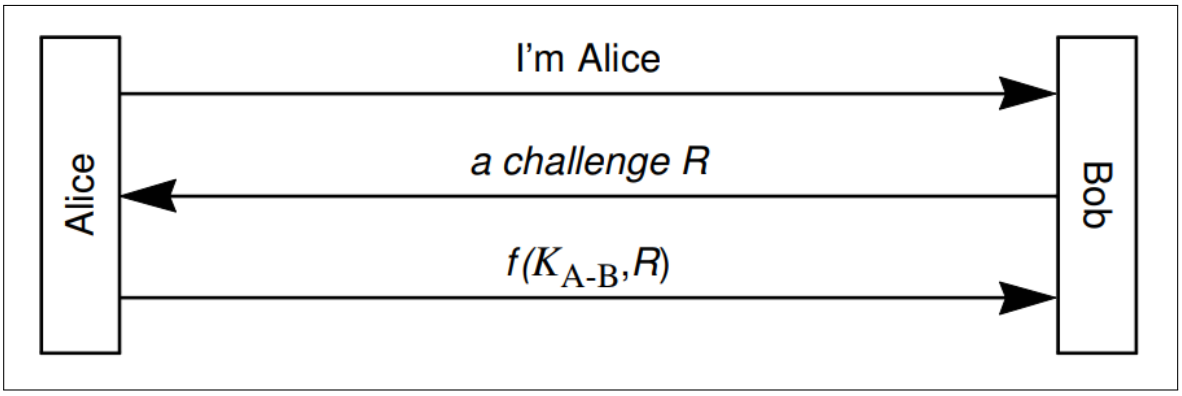

Figure 20: Bob Authenticates Alice Based on a Shared Secret $K_{(A-B)}$

6.3.2 One way authentication using timestamps

6.4 Authentication with trusted server

7 Secret Sharing

8 Access Control

9 Secure Protocols

10 Firewalls

Firewalls are security devices or software that are used to separate a local network from the Internet. They control and monitor network traffic by limiting inbound and outbound traffic, allowing only authorized traffic to pass through. Firewalls also hide the internal network from the external world and regulate access to services.

- In most cases, a firewall acts as a bridge between an internal network and the public internet. It examines the headers and sometimes the payloads of network packets to determine whether to allow or block them. This type of firewall is known as a **packet-filtering router** or packet filter. It can be *stateless* (only considers individual packets) or *stateful* (maintains information about the state of connections).
- Another type of firewall is a **proxy gateway**. All incoming and outgoing traffic is directed through the firewall, and outgoing traffic appears to originate from the firewall itself. Proxy gateways can operate at the *application level* or *circuit level*. Application-level proxies use separate proxies for each application, such as SMTP for email, HTTP for web browsing, and FTP for file transfers. Filtering rules in application-level proxies are application-specific. Circuit-level proxies, on the other hand, are application-independent and work transparently.
- In addition to network-level firewalls, there are **personal firewalls** that are installed on end-user machines. These firewalls have application-specific rules and can control outbound connections from specific applications. For example, a personal firewall may block outbound telnet connections from an email client.

While firewalls provide an important layer of security, it's important to note their limitations. Firewalls cannot protect against attacks that manage to bypass the firewall or originate from within the protected network unless a personal firewall is used. They also cannot prevent or block all possible viruses and worms, as the effectiveness of virus and worm protection depends on the specific characteristics of the operating system and the firewall software itself.

10.1 Packet Filtering

Packet filtering is a method used by firewalls to make decisions on whether to allow or reject individual packets based on specific criteria. Each packet is examined independently, and the decision is made solely based on the information available in the packet itself. Packet filtering is *stateless*, meaning it does not consider the context of the packet, such as the TCP connection or the application it belongs to.
The information used for packet filtering includes the source and destination IP addresses, ports, protocol identifier (**TCP**, **UDP**, **ICMP**, etc.), **TCP flags** (**SYN**, **ACK**, **RST**, **PSH**, **FIN**), and **ICMP message type**. Filtering rules are created based on *pattern-matching* against these packet attributes.
Default rules are defined to determine whether to **accept** or **reject** packets that do not match any specific filtering rule.
As an example, let's consider an FTP packet filter. The following filtering rules allow a user to FTP from any IP address to the FTP server at 172.168.10.12:

- `access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 21`
- `access-list 100 permit tcp any gt 1023 host 172.168.10.12 eq 20`
- `access-list 101 permit tcp host 172.168.10.12 eq 21 any gt 1023`
- `access-list 101 permit tcp host 172.168.10.12 eq 20 any gt 1023`

These rules allow packets from any client to the FTP control and data ports on the FTP server and also allow the FTP server to send packets back to any IP address with TCP ports greater than 1023. The rules are applied to inbound (`access-list 100`) and outbound (`access-list 101`) traffic on the Ethernet 0 interface.
Packet filtering has certain weaknesses. It does not prevent **application-specific attacks**, such as buffer overflows in URL decoding routines. Packet filters lack **user**

authentication mechanisms, except for address-based authentication which can be spoofed. They also lack upper-level functionality and are vulnerable to **TCP/IP attacks** like spoofing.

To mitigate these weaknesses, firewalls can maintain a list of allowed addresses for each interface, ensuring that packets with internal addresses do not originate from outside the network. However, security breaches can still occur due to misconfiguration.

Fragmentation attacks are a specific type of attack that exploits packet filtering. In a fragmentation attack, multiple packets are designed in such a way that each packet individually passes through the firewall. However, when the packets are assembled at the receiving end (where it is possible to check the TCP header), they form a packet that should have been dropped by the firewall. This allows the attacker to bypass the firewall's filtering rules.

10.2 Stateless Packet Filtering and Session Filtering

Stateless packet filtering has a limitation when it comes to handling **TCP connections** that use **port numbers**. In TCP connections, ports with numbers less than 1024 are typically permanently assigned to servers, while clients use ports numbered from 1024 to 16383. Clients must have these ports available to receive responses from servers.

The limitation arises when a firewall receives an incoming request to a client's port numbered, let's say, 1234. The firewall cannot determine whether this incoming traffic is a legitimate response from a previously established connection or if it is malicious traffic. Without keeping *state information* for each connection, the firewall cannot make an accurate decision.

To overcome this limitation, **session filtering** is used. Session filtering takes into account the context of a connection when making filtering decisions. When a new connection is established, it is checked against the security policy. If it is an existing connection, the firewall looks it up in its session table and updates the table if necessary. For example, incoming traffic to a high-numbered port would only be allowed

if there is an established connection to that port.
However, session filtering is more challenging for stateless protocols like **UDP** and **ICMP**. Stateless protocols do not have the concept of connections, making it difficult to track and filter them accurately. In session filtering, a typical approach is to deny everything that is not explicitly allowed. Care must be taken when filtering out service traffic such as ICMP, as it is an essential part of network communication.
It's important to note that even with session filtering, there are still limitations. Firewalls can be bypassed with techniques like **IP tunneling**, where packets are encapsulated within other protocols to evade filtering mechanisms.

10.3 IP Tunneling

IP tunneling is a technique used to encapsulate one network protocol within another network protocol, allowing the traffic of the encapsulated protocol to traverse a network that may not support it directly. It involves wrapping the original protocol packets within the payload of the outer protocol packets.
The concept of IP tunneling is based on the idea of creating a virtual "tunnel" or pathway through a network by encapsulating the original packets within new packets. The encapsulated packets can then be transmitted over a network that may not natively support the encapsulated protocol.

10.4 iptables

iptables is a powerful command-line utility in Linux used for setting up, maintaining, and inspecting the tables of IPv4 packet filter rules in the Linux kernel. It allows administrators to define rules that determine how packets are handled and filtered within the network stack.
iptables operates with different tables, each containing built-in chains and the possibility of user-defined chains. The commonly used tables include:

- **Filter**: This is the default table and is responsible for filtering packets. It contains built-in chains such as INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the system), and OUTPUT (for

locally-generated packets).

- **NAT**: The NAT (Network Address Translation) table is used for address translation, allowing packets to be modified before routing. It includes chains like PREROUTING (for altering incoming packets), OUTPUT (for altering locally-generated packets), and POSTROUTING (for altering outgoing packets).
- **Mangle**: The mangle table is used for specialized packet alteration, such as modifying TCP header values or packet quality of service. It contains chains like PREROUTING, OUTPUT, INPUT, FORWARD, and POSTROUTING.
- **Raw**: The raw table is primarily used for configuring exemptions from connection tracking. It allows packets to bypass connection tracking mechanisms.

Each chain within a table is a list of rules that can match specific criteria for a packet. The rules define the conditions for a packet and specify an associated target, which determines the action to be taken if the packet matches the rule.
Common targets in iptables include:

- **Accept**: The packet is allowed to pass through.
- **Drop**: The packet is silently discarded.
- **Queue**: The packet is passed to userspace for further processing by a queue handler.
- **Return**: Stops traversing the current chain and resumes at the next rule in the previous (calling) chain.
- **User-defined chain**: A rule can specify the name of a user-defined chain, which allows for more complex rule organization and processing.

By defining rules in the appropriate chains within the tables, administrators can control and filter network traffic based on various criteria, such as source/destination IP addresses, ports, protocols, and packet states.
iptables provides granular control over network traffic, allowing administrators to enhance network security, implement firewall rules, and perform network address

translation. It is a versatile tool widely used for network administration and security in Linux-based systems.

10.5 Extended Modules in iptables

iptables, as a packet filtering firewall tool, can utilize extended packet matching modules to provide additional flexibility and functionality in rule definition. These modules can be used implicitly by specifying the protocol (-p) or explicitly with the -m option followed by the name of the matching module.
Once a matching module is specified, additional command-line options become available, depending on the specific module being used. One example of an extended module is the ”-m state” module, which allows matching based on the state of the connection.
The ”-m state” module provides the ability to match packets based on their state, using the ”–state” option followed by a state value. The state values can include:

- **INVALID**: Matches packets that are not associated with a known connection or have invalid states.
- **ESTABLISHED**: Matches packets that are associated with an established connection.
- **NEW**: Matches packets that are part of a new connection.
- **RELATED**: Matches packets that are related to an established connection.

Using these extended modules and their associated options, iptables rules can be constructed to match specific packet attributes and apply corresponding actions. The basic structure of an iptables rule includes the following parameters:

- `-A` (`--append`) *chain*: Specifies the chain to which the rule will be appended.
- *rule-specification*: Defines the criteria for matching packets.
- `-j` (`--jump`) *target*: Specifies the action to be taken if a packet matches the rule.

The parameters in the rule specification can include options such as `-p` (`--protocol`) to specify the protocol, `-s` (`--source`) to specify the source address, `-i` (`--in-interfac` to specify the network interface, and others.
Here are a few examples of iptables rules:

- Block all incoming traffic on the FORWARD chain for the eth0 interface: `iptables -A FORWARD -i eth0 -j DROP`
- Accept packets from outside if they refer to a TCP connection initiated from within the network: `iptables -A FORWARD -i eth0 -m state --state ESTABLISHED -j ACCEPT`

In the examples above, the rules are applied to the FORWARD chain, specifying the eth0 network interface as the point of origin for the packets. The `-j` option specifies the action to be taken on the matching packets, such as dropping, accepting, or rejecting them.
By constructing iptables rules using the available options and extended modules, network administrators can define granular packet filtering and control the flow of network traffic based on specific criteria.

10.6 Writing a rule in iptables

Step 1: Determine the Purpose of the Rule

- What network traffic do you want to allow or block?
- Is it for incoming or outgoing traffic?
- Which network interface is involved?
- What criteria should be used to match the packets?

Step 2: Choose the Chain

- `INPUT`: For filtering incoming packets destined for the local machine.
- `OUTPUT`: For filtering outgoing packets generated by the local machine.
- `FORWARD`: For filtering packets being routed through the machine (if it's acting as a router).

Step 3: Write the Rule Syntax

- `iptables -A [CHAIN] [OPTIONS] -j [ACTION]`
- `-A` appends the rule to the specified chain.
- `[CHAIN]` is the chain name where the rule will be added.
- `[OPTIONS]` are the criteria to match the packets. This can include source/destination IP addresses, ports, protocols, etc.
- `-j [ACTION]` specifies the action to be taken if a packet matches the rule, such as ACCEPT, DROP, REJECT, etc.

Step 4: Specify Criteria

- Based on the purpose of the rule, specify the criteria to match the packets.
- `-s [SOURCE]` specifies the source IP address or IP range.
- `-d [DESTINATION]` specifies the destination IP address or IP range.
- `-p [PROTOCOL]` specifies the protocol (e.g., tcp, udp, icmp).

- `--sport [SOURCE PORT]` specifies the source port.
- `--dport [DESTINATION PORT]` specifies the destination port.
- `-i [INPUT INTERFACE] specifies input interface`
- `-o [OUTPUT INTERFACE] specifies output interface`
- `--icmp-type [ICMP TYPE] ICMP message type`
- `-m state --state [STATE] Connection state`

Step 5: Choose the Action

- Decide on the action to be taken if a packet matches the rule.
- `ACCEPT`: Allow the packet to pass through.
- `DROP`: Silently discard the packet.
- `REJECT`: Discard the packet and send an error response back to the sender.

Step 6: Write the Rule

- Combine all the elements together to write the complete rule.
- `iptables -A INPUT -s 192.168.1.100 -p tcp --dport 22 -j ACCEPT`
- This rule allows incoming TCP traffic on port 22 (SSH) from the source IP address 192.168.1.100.

Step 7: Apply the Rule

- Once you have written the rule, you can apply it using the iptables command.
- However, make sure to consider the order of the rules, as they are evaluated from top to bottom.
- You may need to adjust the rule placement or use specific positions (-I, -R) to insert or replace rules in specific positions.

10.7 Examples

10.7.1 Example 1

This rule allows the firewall to accept TCP packets for routing when they enter on the *eth0* interface, originate from any IP address, and are destined for the IP address 192.168.1.58 reachable via the *eth1* interface. The source port should be in the range of 1024 to 65535, and the destination port should be port 80 (www/http).

- `iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p tcp --sport 1024:65535 --dport 80 -j ACCEPT`

- The rule is using the `iptables` command to append (-A) a rule to the FORWARD chain, which is responsible for packets being routed through the system.
- The explanation states that the rule allows the firewall to accept TCP packets for routing when they enter on the *eth0* interface (`-i eth0`), originate from any IP address (`-s 0/0`), and are destined for the IP address 192.168.1.58 reachable via the *eth1* interface (`-d 192.168.1.58 -o eth1`). This means that the rule is intended for packets that are passing through the firewall from one interface to another.
- The criteria for the source port is specified as being in the range of 1024 to 65535 (`--sport 1024:65535`), indicating that the source port can be any port above 1023 (since ports below 1024 are typically reserved for well-known services).
- The criteria for the destination port is specified as port 80 (`--dport 80`), which corresponds to the standard port used for HTTP (web) traffic.
- The action to be taken for the packets that match this rule is to accept them (`-j ACCEPT`), allowing them to proceed through the firewall for routing.
- Overall, this rule ensures that TCP packets entering on the *eth0* interface, coming from any source IP address, and destined for the IP address 192.168.1.58 on the

eth1 interface, with the source port in the specified range and destination port as port 80, will be accepted for routing purposes.

10.7.2 Example 2

These rules allow the firewall to send ICMP echo-requests (pings) and accept the corresponding ICMP echo-replies.

- `iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT`
- `iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT`

The given example focuses on allowing the firewall to send ICMP echo-requests (pings) and accept the corresponding ICMP echo-replies.
`iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT`
This rule allows outgoing ICMP echo-request packets from the firewall. Let's break down the components:

- `-A OUTPUT` appends the rule to the OUTPUT chain, which handles outgoing packets.
- `-p icmp` specifies the protocol as ICMP.
- `--icmp-type echo-request` specifies the ICMP type as echo-request, which corresponds to a ping request.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

In other words, the first rule allows the firewall to send ICMP echo-request packets.
`iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT`
This rule allows incoming ICMP echo-reply packets to the firewall. Here's how it is broken down:

- `-A INPUT` appends the rule to the INPUT chain, which handles incoming packets.
- `-p icmp` specifies the protocol as ICMP.
- `--icmp-type echo-reply` specifies the ICMP type as echo-reply, which corresponds to a ping reply.

- `-j ACCEPT` indicates that the action for matching packets is to accept them.

This rule enables the firewall to accept ICMP echo-reply packets.
Together, these rules ensure that the firewall can send ICMP echo-requests (pings) and accept the corresponding ICMP echo-replies as responses to those requests. The rules allow bidirectional communication for ICMP ping requests and responses, facilitating network diagnostics and connectivity testing.

10.7.3 Example 3

This rule limits the acceptance of ICMP echo-requests (pings) to at most 1 ping per second on the *eth0* interface.

- `iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -i eth0 -j ACCEPT`

limiting the acceptance of TCP segments with the SYN bit set to no more than five per second

- `iptables -A INPUT -p tcp --syn -m limit --limit 5/s -i eth0 -j ACCEPT`

This example demonstrates limiting the acceptance of certain types of packets based on rate limits.
`iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -i eth0 -j ACCEPT`
This rule limits the acceptance of ICMP echo-request packets (ping requests) to a rate of 1 packet per second. Let's break down the components:

- `-A INPUT` appends the rule to the INPUT chain, which handles incoming packets.
- `-p icmp` specifies the protocol as ICMP.
- `--icmp-type echo-request` specifies the ICMP type as echo-request, which corresponds to a ping request.
- `-m limit --limit 1/s` uses the limit module to set a rate limit of 1 packet per second.
- `-i eth0` specifies that the rule applies to packets received on the eth0 interface.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

In other words, this rule allows the firewall to accept ICMP echo-request packets but limits their acceptance to a rate of 1 packet per second. This helps prevent potential ping flood attacks or excessive ICMP traffic.

```
iptables -A INPUT -p tcp --syn -m limit --limit 5/s -i eth0
-j ACCEPT
```

This rule limits the acceptance of TCP packets with the SYN flag set (indicating the start of a new connection) to a rate of 5 packets per second. Here's how it is broken down:

- `-A INPUT` appends the rule to the INPUT chain, which handles incoming packets.
- `-p tcp` specifies the protocol as TCP.
- `--syn` matches packets with the SYN flag set, indicating the start of a new TCP connection.
- `-m limit --limit 5/s` applies a rate limit of 5 packets per second using the limit module.
- `-i eth0` specifies that the rule applies to packets received on the eth0 interface.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

This rule allows the firewall to accept TCP packets with the SYN flag set but limits their acceptance to a rate of 5 packets per second. It helps mitigate potential SYN flood attacks and controls the rate of incoming TCP connection requests.

10.7.4 Example 4

These rules allow the firewall to accept TCP packets for routing when they enter on the *eth0* interface, originate from any IP address, and are destined for the IP address 192.168.1.58 reachable via the *eth1* interface. The source port should be in the range of 1024 to 65535, and the destination ports should be port 80 (www/http) and port 443 (https).
The return packets from 192.168.1.58 are also allowed. Instead of stating the source and destination ports, you can simply allow packets related to established connections using the `-m` state and `--state ESTABLISHED` options.

- `iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p tcp --sport 1024:65535 -m multiport --dports 80,443 -j ACCEPT`
- `iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i eth1 -p tcp -m state --state ESTABLISHED -j ACCEPT`

The example rules can be written in LaTeX as follows:
Rule 1: Allowing TCP Packets for Routing

- `iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p tcp --sport 1024:65535 -m multiport --dports 80,443 -j ACCEPT`

This rule allows the firewall to accept TCP packets for routing. Let's break down the components:

- `-A FORWARD` appends the rule to the FORWARD chain.
- `-s 0/0` matches packets originating from any source IP address.
- `-i eth0` specifies that the rule applies to packets entering on the eth0 interface.
- `-d 192.168.1.58` matches packets destined for the IP address 192.168.1.58.

- `-o eth1` specifies that the rule applies to packets leaving via the eth1 interface.
- `-p tcp` specifies the protocol as TCP.
- `--sport 1024:65535` matches packets with a source port in the range of 1024 to 65535.
- `-m multiport --dports 80,443` matches packets with a destination port of either 80 or 443.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

This rule allows the firewall to accept TCP packets that originate from any source IP address, enter via the eth0 interface, and are destined for the IP address 192.168.1.58 on the eth1 interface. The source port should be in the range of 1024 to 65535, and the destination ports should be 80 (HTTP) and 443 (HTTPS).

Rule 2: Allowing ESTABLISHED TCP Packets

- `iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i eth1 -p tcp -m state --state ESTABLISHED -j ACCEPT`

This rule allows the firewall to accept TCP packets in the ESTABLISHED state. Here's the breakdown:

- `-A FORWARD` appends the rule to the FORWARD chain.
- `-d 0/0` matches packets destined for any IP address.
- `-o eth0` specifies that the rule applies to packets leaving via the eth0 interface.
- `-s 192.168.1.58` matches packets originating from the IP address 192.168.1.58.
- `-i eth1` specifies that the rule applies to packets entering on the eth1 interface.
- `-p tcp` specifies the protocol as TCP.
- `-m state --state ESTABLISHED` matches packets in the ESTABLISHED state, which are part of existing TCP connections.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

This rule allows the firewall to accept TCP packets in the ESTABLISHED state, which are responses or related to previously initiated connections from the IP address 192.168.1.58.

10.7.5 Example 5

These rules allow DNS (UDP port 53) access from and to the firewall.

- `iptables -A OUTPUT -p udp -o eth0 --dport 53 --sport 1024:65 -j ACCEPT`
- `iptables -A INPUT -p udp -i eth0 --sport 53 --dport 1024:655 -j ACCEPT`

Rule 1: Allowing Outgoing DNS Traffic

The rule `iptables -A OUTPUT -p udp -o eth0 --dport 53 --sport 1024:65535 -j ACCEPT` is added to the OUTPUT chain using the `-A` option, which appends the rule to the chain.

- `-p udp` specifies that the rule applies to UDP traffic.
- `-o eth0` indicates that the rule applies to outgoing traffic on the `eth0` network interface.
- `--dport 53` specifies that the destination port should be port 53, which is commonly used for DNS (Domain Name System) queries.
- `--sport 1024:65535` sets the source port range from 1024 to 65535, allowing dynamic or ephemeral ports used for the response packets.
- `-j ACCEPT` indicates that the matched packets should be accepted.

In summary, this rule allows outgoing UDP packets on the `eth0` interface with a destination port of 53 (DNS) and a source port range of 1024 to 65535. It permits DNS queries originating from the system.

Rule 2: Allowing Incoming DNS Traffic

The rule `iptables -A INPUT -p udp -i eth0 --sport 53 --dport 1024:65535 -j ACCEPT` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-p udp` specifies that the rule applies to UDP traffic.
- `-i eth0` indicates that the rule applies to incoming traffic on the `eth0` network interface.
- `--sport 53` specifies that the source port should be port 53, which is commonly used for DNS response packets.
- `--dport 1024:65535` sets the destination port range from 1024 to 65535, allowing dynamic or ephemeral ports used for the original DNS queries.
- `-j ACCEPT` indicates that the matched packets should be accepted.

In summary, this rule allows incoming UDP packets on the `eth0` interface with a source port of 53 (DNS responses) and a destination port range of 1024 to 65535. It permits the DNS response packets to be accepted by the system.

10.7.6 Example 6

These rules allow HTTP (port 80) and SSH (port 22) access to the firewall.

- `iptables -A OUTPUT -o eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT`
- `iptables -A INPUT -p tcp -i eth0 --dport 22 --sport 1024:65535 -m state --state NEW -j ACCEPT`
- `iptables -A INPUT -p tcp -i eth0 --dport 80 --sport 1024:65535 -m state --state NEW -j ACCEPT`

Rule 1: Allowing Established and Related Outgoing Connections

The rule `iptables -A OUTPUT -o eth0 -m state --state ESTABLISHED,R -j ACCEPT` is added to the OUTPUT chain using the `-A` option, which appends the rule to the chain.

- `-o eth0` specifies the outgoing network interface as eth0.
- `-m state --state ESTABLISHED,RELATED` matches packets that are part of an established or related connection.
- `-j ACCEPT` indicates that the matched packets should be accepted.

This rule allows outgoing packets on the eth0 interface that are part of an established or related connection, such as response packets to outgoing requests.

Rule 2: Allowing Incoming SSH Connections

The rule `iptables -A INPUT -p tcp -i eth0 --dport 22 --sport 1024:65535 -m state --state NEW -j ACCEPT` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-p tcp` specifies that the rule applies to TCP traffic.
- `-i eth0` indicates that the rule applies to incoming traffic on the eth0 network interface.

- `--dport 22` specifies that the destination port should be port 22, which is commonly used for SSH (Secure Shell) connections.
- `--sport 1024:65535` sets the source port range from 1024 to 65535, allowing dynamic or ephemeral ports used for the response packets.
- `-m state --state NEW` matches packets that are part of a new connection.
- `-j ACCEPT` indicates that the matched packets should be accepted.

This rule allows incoming TCP packets on the eth0 interface with a destination port of 22 (SSH) and a source port range of 1024 to 65535, which are typically used for client-side connections. It permits new SSH connections initiated from external sources.

Rule 3: Allowing Incoming HTTP Connections

The rule `iptables -A INPUT -p tcp -i eth0 --dport 80 --sport 1024:65535 -m state --state NEW -j ACCEPT` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-p tcp` specifies that the rule applies to TCP traffic.
- `-i eth0` indicates that the rule applies to incoming traffic on the eth0 network interface.
- `--dport 80` specifies that the destination port should be port 80, which is commonly used for HTTP connections.
- `--sport 1024:65535` sets the source port range from 1024 to 65535, allowing dynamic or ephemeral ports used for the response packets.
- `-m state --state NEW` matches packets that are part of a new connection.
- `-j ACCEPT` indicates that the matched packets should be accepted.

This rule allows incoming TCP packets on the eth0 interface with a destination port of 80 (HTTP) and a source port range of 1024 to 65535, which are typically used for client-side connections. It permits new HTTP connections initiated from external sources.

These rules are designed to allow specific incoming and outgoing connections on host H. The first rule ensures that established or related outgoing connections are accepted. The second and third rules allow incoming SSH (port 22) and HTTP (port 80) connections, respectively, permitting new connections from external sources.

10.7.7 Example 7

These rules allow the firewall to access the Internet, allowing outbound HTTP (port 80) and HTTPS (port 443) traffic.

- `iptables -A OUTPUT -j ACCEPT -m state --state NEW,ESTABLISHED,RELATED -o eth0 -p tcp -m multiport --dports 80,443 --sport 1024:65535`
- `iptables -A INPUT -j ACCEPT -m state --state ESTABLISHED,RELATED -i eth0 -p tcp`

Rule 1: Allowing Outgoing HTTP and HTTPS Connections

The rule `iptables -A OUTPUT -j ACCEPT -m state --state NEW,ESTABLI -o eth0 -p tcp -m multiport --dports 80,443 --sport 1024:65535` is added to the OUTPUT chain using the `-A` option, which appends the rule to the chain.

- `-j ACCEPT` indicates that the matched packets should be accepted.
- `-m state --state NEW,ESTABLISHED,RELATED` matches packets that are either new, established, or related connections.
- `-o eth0` specifies the outgoing network interface as eth0.
- `-p tcp` specifies that the rule applies to TCP traffic.
- `-m multiport --dports 80,443` matches packets with a destination port of either 80 (HTTP) or 443 (HTTPS).
- `--sport 1024:65535` sets the source port range from 1024 to 65535, allowing dynamic or ephemeral ports used for the response packets.

This rule allows outgoing TCP packets on the eth0 interface with a destination port of either 80 or 443. It permits new, established, or related connections for HTTP and HTTPS traffic. The source port range ensures that response packets can be received.

Rule 2: Allowing Incoming Established and Related TCP Connections

The rule `iptables -A INPUT -j ACCEPT -m state --state ESTABLISHED, -i eth0 -p tcp` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-j ACCEPT` indicates that the matched packets should be accepted.
- `-m state --state ESTABLISHED,RELATED` matches packets that are either part of an established or related connection.
- `-i eth0` specifies the incoming network interface as eth0.
- `-p tcp` specifies that the rule applies to TCP traffic.

This rule allows incoming TCP packets on the eth0 interface that are part of an established or related connection. It permits responses to outgoing connections and ensures that established connections can receive data.

These rules effectively allow access to the Internet from the firewall (host H) and permit outbound connections on ports 80 (HTTP) and 443 (HTTPS). The first rule allows outgoing packets for new, established, or related connections to these ports, and the second rule accepts incoming packets that are part of established or related connections on the eth0 interface.

10.7.8 Example 8

These rules allow the home network (192.168.1.0/24) to access the firewall.

- `iptables -A INPUT -j ACCEPT -p all -s 192.168.1.0/24 -i eth1`
- `iptables -A OUTPUT -j ACCEPT -p all -d 192.168.1.0/24 -o eth1`

Rule 1: Allowing Incoming Traffic from the LAN Network

The rule `iptables -A INPUT -j ACCEPT -p all -s 192.168.1.0/24 -i eth1` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-j ACCEPT` indicates that the matched packets should be accepted.
- `-p all` matches packets of any protocol.
- `-s 192.168.1.0/24` specifies the source IP address or network as 192.168.1.0/24, representing the LAN network.
- `-i eth1` specifies the incoming network interface as eth1.

This rule allows incoming packets from the LAN network (192.168.1.0/24) on the eth1 interface, regardless of the protocol. It permits communication from the LAN to the firewall (host H).

Rule 2: Allowing Outgoing Traffic to the LAN Network

The rule `iptables -A OUTPUT -j ACCEPT -p all -d 192.168.1.0/24 -o eth1` is added to the OUTPUT chain using the `-A` option, which appends the rule to the chain.

- `-j ACCEPT` indicates that the matched packets should be accepted.
- `-p all` matches packets of any protocol.
- `-d 192.168.1.0/24` specifies the destination IP address or network as 192.168.1.0/24 representing the LAN network.

- `-o eth1` specifies the outgoing network interface as eth1.

This rule allows outgoing packets to the LAN network (192.168.1.0/24) on the eth1 interface, regardless of the protocol. It permits communication from the firewall (host H) to the LAN network.

10.7.9 Example 9

These rules allow loopback traffic, which enables communication between processes on the firewall itself.

- `iptables -A INPUT -i lo -j ACCEPT`
- `iptables -A OUTPUT -o lo -j ACCEPT`

Rule 1: Allowing Incoming Traffic on the Loopback Interface
The rule `iptables -A INPUT -i lo -j ACCEPT` is added to the INPUT chain using the `-A` option, which appends the rule to the chain.

- `-i lo` specifies the incoming network interface as lo, which represents the loopback interface.
- `-j ACCEPT` indicates that the matched packets should be accepted.

This rule allows incoming packets on the loopback interface, which is typically used for local communication within the same host. It permits internal communication between processes running on the host.
Rule 2: Allowing Outgoing Traffic on the Loopback Interface
The rule `iptables -A OUTPUT -o lo -j ACCEPT` is added to the OUTPUT chain using the `-A` option, which appends the rule to the chain.

- `-o lo` specifies the outgoing network interface as lo, which represents the loopback interface.
- `-j ACCEPT` indicates that the matched packets should be accepted.

This rule allows outgoing packets on the loopback interface, enabling local communication from processes running on the host.
These rules ensure that the loopback interface is accessible and permit local communication between processes running on the host. The INPUT rule allows incoming packets on the loopback interface, and the OUTPUT rule allows outgoing packets on the loopback interface.

The loopback interface is essential for internal host operations, and these rules enable various services and processes to communicate with each other on the same host without going through the network stack.

11 Email Security

12 Web Technologies

13 Web Security

14 Web Tracking

15 Exams

15.1 9 January 2023

15.1.1 Ex 1. Hashing

1. What are pros and cons of keyed and unkeyed hashing?
2. Do cryptographic hash functions have less collision than non-cryptographic hashing functions? Elaborate.
3. Why are strongly collision-resistant functions also weakly collision-resistant?
4. Is the traditional binary representation of integers a cryptographic hash function? Explain.

Solution:

1. Pros and Cons of Keyed and Unkeyed Hashing:

   Keyed Hashing: Keyed hashing, also known as HMAC (Hash-based Message Authentication Code), provides an additional layer of security by incorporating a secret key into the hashing process. Pros of keyed hashing include:

   - Authentication: The use of a secret key ensures that only entities possessing the key can generate or verify the hash, providing authentication of the data.
   - Tamper Resistance: Modifying the data or the key will result in a different hash value, making it difficult for an attacker to tamper with the data without detection.

   Unkeyed Hashing: Unkeyed hashing, also referred to as non-keyed hashing, does not utilize a secret key in the hashing process. Pros of unkeyed hashing include:

   - Simplicity: Unkeyed hash functions are generally simpler to implement and use, as they do not require the management of secret keys.
   - Efficiency: Without the additional overhead of a secret key, unkeyed hash functions can be computationally more efficient.

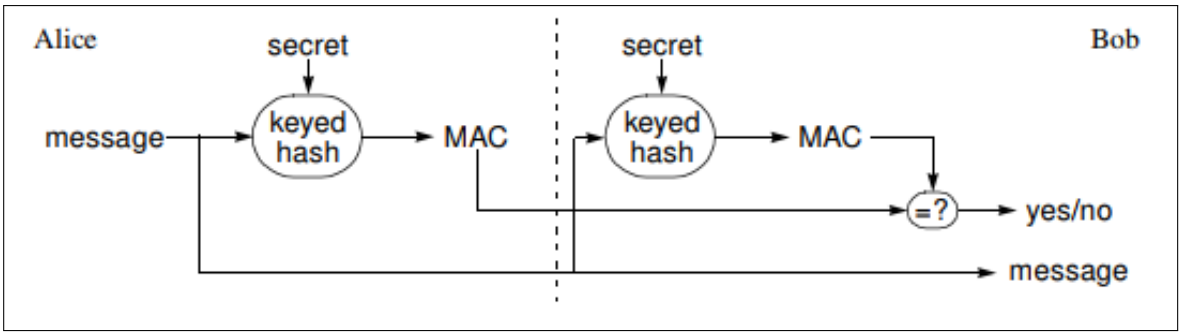

Figure 21: Keyed Hashed Map

2. Cryptographic Hash Functions and Collisions:

   Cryptographic hash functions are designed to have strong collision resistance, meaning it is computationally infeasible to find two different inputs that produce the same hash value (collision). Non-cryptographic hashing functions, on the other hand, may have a higher probability of collisions due to their primary focus on efficiency rather than collision resistance.

   While cryptographic hash functions aim to minimize collisions, it is important to note that collisions are still possible due to the pigeonhole principle, which states that if the number of possible inputs is larger than the number of possible hash outputs, collisions are inevitable. However, the strength of cryptographic hash functions lies in the difficulty of finding such collisions, making them highly useful for applications like data integrity verification, password storage, and digital signatures.

3. Strong Collision Resistance and Weak Collision Resistance:

   Strong collision resistance refers to the property of a hash function where it is computationally infeasible to find any two distinct inputs that produce the same hash value. Weak collision resistance, on the other hand, means that it is computationally difficult to find two specific inputs that result in the same hash value.

   The reason why strongly collision-resistant functions are also weakly collision-resistant is that if an attacker can find a specific pair of inputs that collide, they can demonstrate a general collision by applying additional modifications to those

inputs. In other words, if there exists a pair of inputs A and B such that hash(A) = hash(B), an attacker can create a new pair of inputs A' and B' such that hash(A') = hash(B') by manipulating A and B based on the collision found.

4. No, the traditional binary representation of integers is not a cryptographic hash function. The traditional binary representation of integers is a way of representing numbers in a base-2 numeral system, where each digit can be either 0 or 1. It is a fundamental concept in computer science and mathematics but does not possess the properties or characteristics of a cryptographic hash function.

   A cryptographic hash function, on the other hand, is a mathematical algorithm that takes an input (message or data) and produces a fixed-size output called a hash value or digest. Cryptographic hash functions are designed to have specific properties, including:

   (a) Deterministic: For the same input, the hash function always produces the same hash value.

   (b) Pre-image resistance: It should be computationally infeasible to determine the original input from the hash value.

   (c) Collision resistance: It should be highly unlikely for two different inputs to produce the same hash value.

   (d) Diffusion: A small change in the input should result in a significantly different hash value.

   (e) Non-reversible: It should be computationally infeasible to determine the original input from the hash value without a pre-existing mapping.

   The traditional binary representation of integers does not exhibit these properties.

15.1.2 Ex 2. Key exchange

Describe at least two methods used for letting two parties exchange a secret key.

Solution:

1. Diffie-Hellman Key Exchange: Diffie-Hellman allows two parties, Alice and Bob, to agree on a shared secret key over an insecure channel. They each generate their own public-private key pairs and exchange public keys. By performing mathematical operations with their own private keys and the other party's public key, they can compute a shared secret that can be used as a symmetric key for encryption and decryption.

2. Key Exchange using Public Key Infrastructure (PKI): In a PKI-based key exchange, Alice and Bob each have a pair of public and private keys. They obtain each other's public keys through a trusted certificate authority (CA) or a secure key distribution protocol. Alice can encrypt the secret key with Bob's public key and send it to him. Bob can then decrypt the encrypted key using his private key to obtain the shared secret.

3. Key Transport using Symmetric Encryption: In this method, a trusted third party, called a Key Distribution Center (KDC), is involved. Alice and Bob both share a secret key with the KDC. When they want to exchange a secret key, they request the KDC to securely send the key to each other. The KDC uses symmetric encryption to protect the key during transmission.

4. Pre-shared Key (PSK) Exchange: In scenarios where Alice and Bob already share a secret key, such as a previously established secure channel or a shared secret obtained through an out-of-band method, they can directly use this shared key as the secret key for their communication.

5. Secure Sockets Layer/Transport Layer Security (SSL/TLS) Handshake: SSL/TLS protocols use asymmetric encryption and digital certificates to establish a secure channel between two parties. During the handshake process, the client and server exchange public keys and establish a shared secret key for subsequent symmetric encryption.

15.1.3 Ex 3. Authentication

1. Why do servers that authenticate users with Lamport hashing prefer that the user always use the same device?
2. With reference to password-based authentication, describe a (any) protocol to make client and server talk so that the client supplies the password but prevents replay and reflection attacks.
3. Can digital signature (whatever it is) authenticate a user of a web application? Discuss.

Solution:

1. Servers that authenticate users with Lamport hashing prefer that the user always use the same device because Lamport hashing is a one-time password scheme that relies on the secrecy of the password and the server's knowledge of the user's password hash. When the user authenticates with the server, the server compares the received password hash with its stored hash. However, if the user switches devices, the stored password hash on the server will not match the hash generated by the new device.

   To address this issue, the server would need to store multiple password hashes corresponding to each device used by the user, which increases complexity and potential security risks. Therefore, to maintain simplicity and security, it is preferable for users to consistently use the same device for authentication with servers that employ Lamport hashing.

2. When it comes to password-based authentication, a protocol that can address replay and reflection attacks is the use of a challenge-response mechanism with the inclusion of a timestamp.

   Here's a high-level description of the protocol:

   - The client sends a request to the server to initiate the authentication process.
   - The server generates a random challenge and sends it to the client.

- The client combines the challenge with the password, computes a hash of the concatenation (e.g., using a cryptographic hash function), and sends the hash value back to the server.
- The server verifies the received hash by performing the same hash computation using the stored password and the challenge.
- To prevent replay attacks, the server includes a timestamp in the challenge. The client includes this timestamp in the response, and the server verifies that it matches the expected value within an acceptable time window.

This protocol prevents replay attacks because the client cannot simply replay a previously captured response. Additionally, it mitigates reflection attacks because the server includes a challenge that is unique for each authentication attempt.

3. Key Transport using Symmetric Encryption: In this method, a trusted third party, called a Key Distribution Center (KDC), is involved. Alice and Bob both share a secret key with the KDC. When they want to exchange a secret key, they request the KDC to securely send the key to each other. The KDC uses symmetric encryption to protect the key during transmission.

4. Pre-shared Key (PSK) Exchange: In scenarios where Alice and Bob already share a secret key, such as a previously established secure channel or a shared secret obtained through an out-of-band method, they can directly use this shared key as the secret key for their communication.

5. Secure Sockets Layer/Transport Layer Security (SSL/TLS) Handshake: SSL/TLS protocols use asymmetric encryption and digital certificates to establish a secure channel between two parties. During the handshake process, the client and server exchange public keys and establish a shared secret key for subsequent symmetric encryption.

15.1.4 Ex 4. Shamir secret sharing

1. In a Shamir scheme (2, 5) users got the points A=(1, 5), B=(2, 0), C=(3, 2), D=(4, 4) and E=(5, 6). Assume to work on GF(7). Reconstruct and return the secret starting from the points A and B.
2. Reconstruct again the secret starting from D and E, and verify that the secret is the same.

Solution: **Reconstruction starting from points A and B**

Given points:

$$A = (1, 5)$$

$$B = (2, 0)$$

To reconstruct the secret, we need to interpolate a polynomial of degree 1 (since we have 2 points).
The polynomial can be represented as:

$$P(x) = a + bx$$

Using the given points, we can set up two equations:

$$5 = a + b \cdot 1$$

$$0 = a + b \cdot 2$$

Solving these equations, we find:

$$a = 2$$

$$b = 3$$

Thus, the polynomial is:

$$P(x) = 2 + 3x$$

To find the secret, we evaluate the polynomial at $x = 0$:

$$P(0) = 2 + 3 \cdot 0 = 2$$

Therefore, the secret is 2.

Reconstruction starting from points D and E

Given points:

$$D = (4, 4)$$

$$E = (5, 6)$$

Using the same process, we set up the equations:

$$4 = a + b \cdot 4$$

$$6 = a + b \cdot 5$$

Solving these equations, we find:

$$a = 0$$

$$b = 6$$

Thus, the polynomial is:

$$P(x) = 0 + 6x$$

To find the secret, we evaluate the polynomial at $x = 0$:

$$P(0) = 0 + 6 \cdot 0 = 0$$

Therefore, the secret is 0.
By reconstructing the secret starting from points A and B and points D and E, we obtain the same secret value of 0.

15.1.5 Firewalls

1. When iptables is used as a personal firewall is there any case where the routing chain needs to be configured with rules?

2. Suppose iptables is running on host H which is protecting a LAN, from which it can be reached with the IP 192.168.1.254/24. Write appropriate rules so that the only network connection allowed to H is via ssh (two-way talk) 18 the LAN host

192.168.1.2/24 to allow firewall configuration. All other network connections to and from H are prohibited.

Solution:

1. When iptables is used as a personal firewall on a standalone system, the need to configure rules in the routing chain (i.e., FORWARD chain) depends on the specific network setup and requirements.

   In a typical personal firewall setup, where the system is not functioning as a router, the main focus is on incoming and outgoing traffic, which is handled by the INPUT and OUTPUT chains, respectively. The FORWARD chain, which deals with packets being routed through the system, is usually not a concern in this scenario.

   However, there might be cases where the personal firewall setup involves routing functionality, such as when the system is acting as a router or a gateway between multiple networks. In such cases, the FORWARD chain needs to be configured with appropriate rules to handle packets being forwarded through the system.

2. To allow only SSH connections from the LAN host 192.168.1.2 to the firewall host H (192.168.1.254) and prohibit all other network connections, you can use the following iptables rules:

   ```
   iptables -A INPUT -s 192.168.1.2/24 -p tcp --dport 22 -m
   state --state NEW,ESTABLISHED -j ACCEPT
   iptables -A OUTPUT -d 192.168.1.2/24 -p tcp --sport 22
   -m state --state ESTABLISHED -j ACCEPT

   iptables -A INPUT -j DROP
   iptables -A OUTPUT -j DROP
   ```

   More details:

   ```
   iptables -A INPUT -s 192.168.1.2/24 -p tcp --dport 22 -m
   ```

`state --state NEW,ESTABLISHED -j ACCEPT`

This rule allows incoming TCP packets from the LAN host 192.168.1.2 on port 22 (SSH). Let's break down the components:

- `-A INPUT` appends the rule to the INPUT chain, which handles incoming packets.
- `-s 192.168.1.2/24` specifies the source IP range as 192.168.1.2 to allow SSH connections from the LAN host.
- `-p tcp --dport 22` specifies the protocol as TCP and destination port as 22 (SSH).
- `-m state --state NEW,ESTABLISHED` matches packets in the NEW and ESTABLISHED states to allow the initial SSH connection and subsequent communication.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

```
iptables -A OUTPUT -d 192.168.1.2/24 -p tcp --sport 22
-m state --state ESTABLISHED -j ACCEPT
```

This rule allows outgoing TCP packets to the LAN host 192.168.1.2 on port 22 (SSH). Here's how it is broken down:

- `-A OUTPUT` appends the rule to the OUTPUT chain, which handles outgoing packets.
- `-d 192.168.1.2/24` specifies the destination IP range as 192.168.1.2 to allow SSH connections to the LAN host.
- `-p tcp --sport 22` specifies the protocol as TCP and source port as 22 (SSH).
- `-m state --state ESTABLISHED` matches packets in the ESTABLISHED state to allow the response packets of the SSH connection.
- `-j ACCEPT` indicates that the action for matching packets is to accept them.

```
iptables -A INPUT -j DROP
```

This rule drops all other incoming packets that do not match the previous rules, effectively prohibiting any additional network connections to the firewall host.

```
iptables -A OUTPUT -j DROP
```

This rule drops all outgoing packets that do not match the previous rules, effectively prohibiting any other network connections from the firewall host.

15.2 3 February 2023

15.2.1 Ex 1. Authenticity versus authentication

Does one contain the other? Or does the other contain one? Any differences? Carefully explain.

Solution:

- Authenticity: Authenticity refers to the quality or state of being genuine, original, or trustworthy. It relates to the assurance that something is what it claims to be and has not been tampered with or altered. Authenticity focuses on the integrity and trustworthiness of an entity, such as a message, data, or object. It ensures that the entity has not been modified, forged, or manipulated in any unauthorized way. Establishing authenticity involves mechanisms that provide evidence of the origin and integrity of the entity, such as digital signatures, checksums, or hashing.
- Authentication: Authentication, on the other hand, is the process of verifying the identity or legitimacy of an entity, such as a user, system, or device. It involves confirming that the claimed identity of the entity is valid and reliable. Authentication is about establishing trust in the identity of the entity and ensuring that it is authorized to access certain resources or perform specific actions. Common authentication methods include passwords, biometrics, digital certificates, or multi-factor authentication.

Authenticity can contribute to the overall trustworthiness of an authentication process by verifying the integrity of the data or message being authenticated. However, authentication itself does not guarantee authenticity. Authenticity is a broader concept that encompasses the trustworthiness of the entire entity, while authentication is a subset that specifically verifies identity.

15.2.2 Ex 2. Symmetric encryption/decryption

1. Describe the architecture of OFB, both for encryption and for decryption.

2. Does OFB need to use a randomly generated IV at any encryption? Why?

3. What is a KDF and why is it useful?

4. What is the effect of the following command line string?
   ```
   echo "SilverSurfer" -n | openssl enc -aes-128-cbc -p -out
   out.enc
   ```

5. If in CBC the attacker flips the third bit of the first ciphertext block, and no integrity mechanism is in place, what is the total effect of that in Bob's reconstructed plaintext message? Discuss.

6. If in the CBC it is Alice who inverts the third bit of the first plaintext block, and no integrity mechanism is provided, what is the total effect of this intervention in Bob's reconstructed plaintext message? Discuss.

15.2.3 Ex 3. Digital certificates

1. Compare CRLs to OCSP

2. What is OCSP stapling and why is it done?

3. Why don't we need the https protocol but can use plain http in retrieving a certificate?

Solution:

15.2.4 Ex 4. Digital signatures

1. Alice and Bob digitally sign the same document M, which is a contract. If Alice is the first signer is there a difference whether Bob signs M or M with Alice's signature? Discuss.

2. f the verification of a digital signature fails (without diagnostics, so we don't know the reason for the failure), what is it correct to infer? Discuss.

3. The following sentence contains a poorly posed (and therefore incorrect) question, written solely to confuse ideas.
   If Alice wants to digitally sign a document, but she doesn't know the public key,

how should she do it?

Discuss why such a question should not be asked.

Solution:

15.2.5 Ex 5. Firewalls

1. Compare blacklisting with whitelisting and discuss the impact of those policies on iptables.
2. Suppose iptables is running on host H, which is protecting a LAN. Write appropriate rules so that
   (a) H does not allow any incoming or outgoing network connections (can only be configured using the local console)
   (b) H allows any connection from the LAN to the Internet, but does not allow opening connections from the Internet to the LAN.

Solution:

1. **Blacklisting:**
   - Focuses on identifying and blocking known threats.
   - Blocks traffic based on a list of known malicious entities or actions.
   - Rules are created in iptables to drop or reject traffic that matches the blacklist.
   - Requires continuous monitoring and updating of the blacklist.
   - Reactive approach, may not be effective against unknown threats or zero-day vulnerabilities.
   - Can result in a large number of rules and complex rule management.

   Whitelisting:
   - Focuses on explicitly allowing trusted entities or actions.
   - Allows traffic that matches a list of approved entities or actions.

- Rules are created in iptables to explicitly allow traffic from trusted sources.
- Proactive approach, minimizes the attack surface and reduces unauthorized access.
- Requires careful planning and ongoing management to avoid blocking legitimate traffic.
- Can result in a smaller number of rules, simplifying rule management.

Impact on iptables:

- Blacklisting involves creating rules to drop or reject traffic, potentially leading to a larger number of rules and performance impact.
- Whitelisting involves creating rules to explicitly allow traffic, requiring careful planning and ongoing management.
- Both policies can be implemented in iptables, and the choice depends on security requirements and level of control needed.

2. **Rule 1: Set Default Policies to DROP**

- `iptables -P INPUT DROP`: Sets the default policy for the INPUT chain to DROP, meaning that any incoming packets that do not match a specific rule will be dropped.
- `iptables -P OUTPUT DROP`: Sets the default policy for the OUTPUT chain to DROP, so any outgoing packets that do not match a specific rule will be dropped.
- `iptables -P FORWARD DROP`: Sets the default policy for the FORWARD chain to DROP, ensuring that any packets being forwarded through the firewall will be dropped unless there is a specific rule allowing them.

Rule 2: Allow Local Console Access

- `iptables -A INPUT -i lo -j ACCEPT`: Allows incoming packets on the loopback interface (lo), which is used for local communication. This rule ensures that local console access is permitted.

- `iptables -A OUTPUT -o lo -j ACCEPT`: Allows outgoing packets on the loopback interface (lo), ensuring that local console access can communicate with other local processes.

3. **Rule 1: Set Default Policies**

   - `iptables -P INPUT DROP`: Sets the default policy for the INPUT chain to DROP, meaning that any incoming packets that do not match a specific rule will be dropped.
   - `iptables -P FORWARD DROP`: Sets the default policy for the FORWARD chain to DROP, ensuring that any packets being forwarded through the firewall will be dropped unless there is a specific rule allowing them.
   - `iptables -P OUTPUT ACCEPT`: Sets the default policy for the OUTPUT chain to ACCEPT, allowing outgoing packets by default.

   Rule 2: Allow Loopback Interface

   - `iptables -A INPUT -i lo -j ACCEPT`: Allows incoming packets on the loopback interface (lo), which is used for local communication.

   Rule 3: Allow Established and Related Connections on eth0

   - `iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RE -j ACCEPT`: Allows incoming packets on the eth0 interface that are part of established or related connections. This rule ensures that responses to outgoing connections or related traffic are accepted.

   Rule 4: Allow Established and Related Connections on Forwarding

   - `iptables -A FORWARD -i eth0 -o eth1 -m state --state ESTABLISHED,RELATED -j ACCEPT`: Allows forwarding of packets from the eth0 interface to the eth1 interface if they are part of established or related connections. This rule ensures that packets belonging to established connections can be forwarded.

By implementing these rules, the firewall will drop incoming packets by default, except for those on the loopback interface and established/related connections on the eth0 interface. The forwarding of packets from eth0 to eth1 will also be allowed if they are part of established or related connections.

15.3 6 April 2023

15.3.1 Ex 1. Digital signatures and time-stamping

1. Describe the basic characteristics of the DSS approach to digital signing. What is the advantage of using two pairs of keys for each signature?

2. Alice, Bob and Charlie have made a written agreement and now need to digitally sign it, and to attach a secure time-stamp to each signature. Describe what type of infrastructure they need and a sequence of steps for accomplishing their task.

Solution:

15.3.2 Ex 2. Cryptographic hashing functions

1. Describe the requirements to be met by a cryptographic hashing function.

2. Describe the Merkle-Damgard construction for hashing a message longer than just one block.

3. Discuss the security of (keyed) hashing k|m, m|k, k|m|k, where m is a message. k is a secret key and | a symbol denoting concatenation.

Solution:

15.3.3 Ex 3. Rock-paper-scissors game

Alice and Bob play a Rockpaperscissors match. In a single match the two parties simultaneously form one of the shapes and the winner is established by the simple chain of circular rules rock beats scissors, scissor beats paper and paper beats rocks. The two players use the following protocol:
(Alice and Bob choose their shapes a and b, where h is a known cryptographic hash function.)
A$\rightarrow$B: $h(a)$
B$\rightarrow$A: b
A$\rightarrow$B: a
(Bob checks $h(a)$; then both Alice and Bob know the winner of the game)

1. Discuss possible weaknesses of the protocol, with respect to possible fraudulent behaviors from Alice and(or) Bob, both ready to cheat in order to win the game.
2. Fix the weaknesses (small changes), without introducing third parties or public-key cryptography.

Solution:

15.3.4 Ex 4. Firewall

1. llustrate the most relevant characteristics of iptables, employed as a firewall.
2. What iptables rules would you set for a mail server accepting (bidirectional) conversations in EMSTP (port 465) and IMAP (port 993), having a network interface eth1 exposed to the Internet and another network interface eth2 exposed to the corporate network? Default policy is DENY.

Solution:

15.3.5 Ex 5. Miscellaneous

Provide short answers to the following questions.

1. What is the Optimal Asymmetric Encryption Padding (OAEP) and why does it provide ”all-or-nothing” security?
2. Determine the multiplicative inverse of 47 mod 64.
3. Given the two primes 23 and 11 find integer a > 1 such that $a^{11} = 1 \bmod 23$

Solution:

15.4 9 June 2023

15.4.1 Ex 1. Symmetric ciphers

1. Describe the scenario of symmetric ciphers and define the concepts of: synchronous/asynchronous stream ciphers, block ciphers and modes of operations.
   If definition is wrong subsequent questions cannot be correctly answered
2. Describe the RC4 cipher (both the key generation and the encryption process). What type of cipher is it?
3. Describe and compare OFB and CTR. Can you suggest possible design criteria for their adoption?

Solution:

1.

15.4.2 Ex 2. Man in the middle

1. Describe the attack Man-In-The-Middle, as well as a possible scenario where it could be run.
2. Alice suspects she is currently being the target of a Man-In-The-Middle attack, and she decides to hire you as a personal adviser. Is it still possible to carry out safe actions? Discuss.

Solution:

1.

15.4.3 Ex 3. Hashing

1. Define the properties that qualify a hashing function as cryptographic (the more formal, the better).
2. Describe the Merkle-Damgård construction. If the underlying hash function maps 256b blocks into 128b blocks, how many rounds are required for hashing a 140KB file?

3. Discuss and compare possible schemes for keying a hash function.

Solution:

15.4.4 Ex 4. RSA verification

1. Describe how to verify an RSA digital signature.
2. Explicitly describe the possible causes of a verification failure. For having non-repudiation is the verification needed?

Solution:

15.4.5 Ex 5. Authentication

1. Discuss the security of the following challenge-based scheme for mutual authentication, where Alice (A) and Bob (B) share a secret key K: (information below is transmitted as clear text)
   A $\rightarrow$ B: (A, NA, B) NA is a nonce chosen by A
   B $\rightarrow$ A: (B, NB, K(NA), A) NB is a nonce chosen by B
   A $\rightarrow$ B: (A, K(NB), B)
2. How would you improve the above schema?

Solution:

15.4.6 Ex 6. Miscellaneous

Provide short answers to the following questions.

1. Compute 5^{12241} mod 13
2. Describe as best as you can the meaning of the following command
   ```
   iptables -A INPUT -p tcp -s 0/0-d 195.55.55.78 --sport
   513:65535 --dport 22 -m state --state NEW, ESTABLISHED
   -j ACCEPT
   ```
3. Describe as best as you can the meaning of the following command
   ```
   ssh -L 44044:192.168.1.221:22 user@host.example.com
   ```

Solution:

15.5 13 January 2022 - A

15.5.1 Ex 1. Cryptographic hashing functions

1. State a minimal but sufficient set of conditions why a hashing function be cryptographic.
2. Can I use a cryptographic hashing function for a hash table? Motivate.
3. Can I use the hashing function of a hash table as a cryptographic hashing function? Motivate.

Solution:

15.5.2 Ex 2. Authentication

1. Username/password based authentication: propose a protocol for having this type of authentication in the LAN, resistant to all types of replay attacks. Can you avoid saving user passwords on the server?
2. How could an authentication process be based on a public key and what are the relevant weaknesses?
3. Locate the weakness in this bidirectional challenge-response authentication scheme and mitigate it:
   A $\rightarrow$ B: A, $Enc(a, X)$
   B $\rightarrow$ A: a, $Enc(b, X)$
   A $\rightarrow$ B: b
   where X is a shared secret key, Enc(y, X) denotes symmetric encryption of y by key X, a is a nonce chosen by A and b is a nonce chosen by B
4. Bob is a smart guy and knows how to invent 128-bits random passphrases (typeable on keyboard) that are robust against dictionary attacks. Alice tells him that in spite of such robustness a 128-bits random key is more secure. Bob replies that the security is the same, but the passphrase can be more easily remembered. What's your position? Explain

Solution:

15.5.3 Ex 3. Access control

1. Is the HRU (Harrison, Ruzzo, Ullman) model secure against offline dictionary attacks? Explain.
2. Give at least two cases where HRU is decidable. Explain

Solution:

15.5.4 Ex 4. Firewalls

Assume that the iptables software is running on host H, having a network interface eth0 (IP:192.168.0.2) connected to a LAN (IP: 192.168.0.0/24; the LAN is protected by H) and a network interface eth1 (IP: 151.100.4.3) connected to the Internet. Assume that the default policy for all built-in chains is DROP

1. Since the LAN contains only local hosts with private IP addresses, it is not possible to directly contact local hosts from the Internet (no IP remapping). However, if H accepts, local hosts can establish two-ways communications with remote hosts on the Internet, based on TCP. For this purpose, which rules will you have to define on the perimeter firewall?
2. Define suitable rules for mitigating a distributed DOS attack to H, based on ping-flooding. Avoid blocking/slowing down other protocols.

Solution:

15.5.5 Ex 5. Short answer (at most 4 lines)

1. Make a comparison between HMAC and digital signing for the purpose of non-repudiation.

Solution:

15.6 13 January 2022 - B

15.6.1 Ex 1. Data integrity

1. Describe what we mean by data integrity and discuss the use of keyed HMACs for guaranteeing the integrity of a file being transmitted over the network (no other guarantees requested).
2. Describe what we mean by data integrity and discuss the use of keyed HMACs for guaranteeing the integrity of a file being transmitted over the network (no other guarantees requested).

Solution:

15.6.2 Ex 2. Diffie-Hellman

1. Describe in detail how two parties can establish a secret key by using the Diffie-Hellman scheme and discuss the vulnerability of the approach.
2. Generalize Diffie-Hellman so that three parties can establish a shared secret key.
3. Describe a scheme for mutual authentication that is strong with respect to dictionary attack and that uses Diffie-Hellman for defining a session key. Do vulnerabilities discussed in 1 still hold?

Solution:

15.6.3 Ex 3. Leader selection

A leader should be selected by randomly choosing one of three parties A, B and C. The parties use the following protocol:

1. Discuss the security of the protocol with respect to possible fraudulent behaviors of A, B and/or C. In particular, is it possible for some of the parties to deterministically choose the leader, while the others are not aware of the fraud?
2. Fix the protocol.

Solution:

15.6.4 Ex 4. Shamir

1. Describe the Shamir scheme (k, n) for sharing a secret
2. Make a numerical example for the case (2, 4), for sharing the secret number 6. Show how the 4 fragments are computed.

Solution:

15.6.5 Ex 5. Access control

1. Illustrate the DAC model (from Harrison-Ruzzo-Ullman, or HRU), define the concept of safety of the protection system and discuss what practical problems arise within the model.
2. Why is the DAC model vulnerable to Trojans? What type of access control model can prevent them from illegally accessing private data? Discuss.

Solution:

15.6.6 Ex 6. Miscellaneous

Provide short answers (2 lines max per question) to the following questions.

1. RSA: if p = 13 and q = 17, what is the range for exponent e?
2. Can iptables block incoming datagrams that are IPSec-tunneled packets going to port 25?
3. What is port forwarding and what protocol implements it?

Solution:

15.7 17 June 2022

15.7.1 Ex 1.

1.

2.

3.

Solution:

15.7.2 Ex 2.

1.

2.

3.

Solution:

15.7.3 Ex 3.

1.

2.

3.

Solution:

15.7.4 Ex 4.

1.

2.

3.

Solution:

15.7.5 Ex 5.

1.

2.

3.

Solution:

15.8 14 July 2022

15.8.1 Ex 1.

1.

2.

3.

Solution:

15.8.2 Ex 2.

1.

2.

3.

Solution:

15.8.3 Ex 3.

1.

2.

3.

Solution:

15.8.4 Ex 4.

1.

2.

3.

Solution:

15.8.5 Ex 5.

1.

2.

3.

Solution:

15.9 8 September 2022

15.9.1 Ex 1.

1.

2.

3.

Solution:

15.9.2 Ex 2.

1.

2.

3.

Solution:

15.9.3 Ex 3.

1.

2.

3.

Solution:

15.9.4 Ex 4.

1.

2.

3.

Solution:

15.9.5 Ex 5.

1.

2.

3.

Solution:

15.9.6 Ex 6.

1.

2.

3.

Solution:

15.9.7 Ex 7.

1.

2.

3.

Solution:

15.9.8 Ex 8.

1. What is the difference between a personal firewall and a perimeter firewall?
2. ou are a home user with a strong Internet connection, however the modem/router - offering only wired connection - is not providing any firewall/NAT. Carefully check the figure. Default iptables policy is ALLOW and you are asked to block all connections to the pc running iptables (and vice versa) except those initiated inside the LAN. Write the corresponding iptables rules.

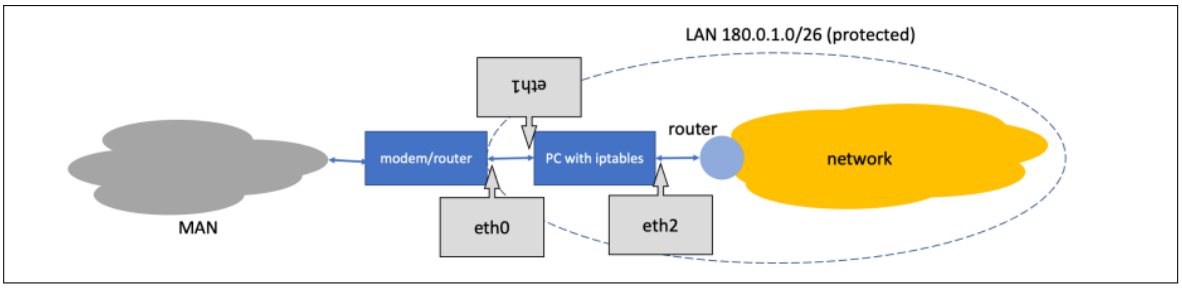

Figure 22: Figure

Solution:

1. A personal firewall is a type of firewall that is installed on an individual device, such as a computer or a smartphone. Its primary purpose is to protect the specific device from unauthorized access and malicious activities. It monitors and controls network traffic to and from the device, blocking or allowing connections based on predefined rules. Personal firewalls are typically configured by the device owner and are focused on protecting the individual device.

   On the other hand, a perimeter firewall, also known as a network firewall, is deployed at the network perimeter to protect an entire network or an organization's infrastructure. It acts as the first line of defense and filters incoming and outgoing traffic between the internal network and the external internet. Perimeter firewalls are responsible for enforcing security policies, blocking unauthorized access attempts, and mitigating various types of network-based threats. They are usually managed by network administrators and provide security for an entire network rather than individual devices.

2. **Setting the Default Policy to Drop for All Chains**

   To set the default policy to drop for all chains, the following commands can be used:

   - `iptables -P INPUT DROP`
   - `iptables -P OUTPUT DROP`
   - `iptables -P FORWARD DROP`

These rules set the default policy to `DROP` for the `INPUT`, `OUTPUT`, and `FORWARD` chains, ensuring that any packets that do not match specific rules will be dropped.

Allowing Traffic Initiated from the Internal LAN

To allow traffic initiated from the internal LAN, the following commands can be used:

- `iptables -A INPUT -i eth1 -s 192.0.1.0/26 -j ACCEPT`
- `iptables -A OUTPUT -o eth1 -d 192.0.1.0/26 -j ACCEPT`

These rules allow incoming and outgoing traffic on the `eth1` interface (connected to the LAN) with the source or destination IP addresses within the LAN network (192.0.1.0/26).

Allowing Loopback (Localhost) Traffic

To allow loopback (localhost) traffic, the following commands can be used:

- `iptables -A INPUT -i lo -j ACCEPT`
- `iptables -A OUTPUT -o lo -j ACCEPT`

These rules allow incoming and outgoing traffic on the loopback interface (localhost), enabling local communication within the PC itself.

These rules effectively block all incoming and outgoing connections to the PC running iptables, except for the traffic initiated from within the LAN and the local loopback traffic. It provides a basic level of security by allowing only internal LAN communication and local host communication while blocking any external connections.

15.10 25 October 2022

15.10.1 Ex 1.

1.

2.

3.

Solution:

15.10.2 Ex 2.

1.

2.

3.

Solution:

15.10.3 Ex 3.

1.

2.

3.

Solution:

15.10.4 Ex 4. Symmetric encryption

1. What is a perfect cipher? Give the mathematical definition.
2. Under what conditions is OTP perfect?
3. Why is it dangerous reusing the same keystream in OTP?
4. Carefully describe the Meet-in-the-Middle attack.
5. What is ”key whitening”? Why and how is it made?

Solution:

1. A perfect cipher, also known as an unconditionally secure cipher, is a theoretical concept in cryptography. It refers to a cipher that provides **perfect secrecy**, meaning that the *ciphertext* does not reveal any information about the *plaintext*, regardless of the computational resources available to an adversary.

   Mathematical Definition

   The mathematical definition of a perfect cipher can be stated as follows:

   - Let P be the set of all possible plaintexts, C be the set of all possible ciphertexts, K be the set of all possible keys, and E_k be the encryption function with key $k \in K$.
   - A cipher (E, D) is considered a perfect cipher if and only if for every plaintext $p \in P$ and ciphertext $c \in C$, and for all distributions of p and c, the following conditions hold:
     (a) For every $p \in P$ and $k \in K$, the probability of encrypting p to obtain c using key k is equal to the probability of obtaining c from any other plaintext p' using the same key k.

     $$\Pr[E_k(p) = c] = \Pr[E_k(p') = c]$$

     (b) For every $c \in C$ and $k \in K$, the probability of obtaining c from any plaintext p using key k is equal to the probability of obtaining c from any other plaintext p' using the same key k.

     $$\Pr[D_k(c) = p] = \Pr[D_k(c) = p']$$

   In simpler terms, a perfect cipher ensures that the probability of obtaining a specific ciphertext from a given plaintext and key is the same as the probability of obtaining that ciphertext from any other plaintext using the same key. Similarly, the probability of obtaining a specific plaintext from a given ciphertext and key is the same as the probability of obtaining that plaintext from any other ciphertext using the same key.

2. • **Key Length**: The key used in OTP must be at least as long as the plaintext and should be truly random. Each key bit should be independent of other key bits and have an equal probability of being 0 or 1.
   • **Key Reuse**: Each key in OTP should be used only once and never repeated. Reusing a key or using a key more than once compromises the security of OTP and can lead to encryption vulnerabilities.
   • **Key Distribution**: The key must be securely distributed to both the sender and the recipient of the encrypted message. The key should be shared through a secure channel to prevent interception or tampering.
   • **Key Secrecy**: The key used in OTP must be kept completely secret from any unauthorized parties. If the key is compromised or known by an adversary, the security of OTP is compromised.
   • **Encryption Process**: The encryption process in OTP involves performing a bitwise exclusive OR (XOR) operation between the plaintext and the corresponding key bits. The resulting ciphertext is a combination of the key and plaintext, making it impossible for an adversary to extract any meaningful information without knowing the exact key.

3. • **Information Leakage**: Interception of multiple ciphertexts allows attackers to exploit the relationship between them. XORing corresponding ciphertexts can reveal patterns and potentially recover the original plaintexts.
   • **Statistical Analysis**: Reusing the keystream enables attackers to perform statistical analysis. Patterns in the plaintext can manifest as patterns in the ciphertext, weakening the encryption's security.
   • **Vulnerability to Known Plaintext Attacks**: If the attacker knows or can guess the plaintext of one message encrypted with a particular keystream, reusing that keystream for another message can lead to the recovery of the second plaintext.
   • **Loss of Perfect Secrecy**: Reusing the keystream violates perfect secrecy, which is achieved when each key is used only once. The introduction of

patterns and relationships makes it possible to gain information about the plaintext, compromising its security.

4. The **Meet-in-the-Middle** attack is a cryptographic attack that takes advantage of the limited key space of a double encryption scheme. It is a technique used to reduce the complexity of exhaustive key search attacks in certain scenarios.

   In a double encryption scheme, a plaintext is encrypted twice using two different encryption algorithms with different keys. The **Meet-in-the-Middle** attack exploits the fact that performing both encryptions in reverse order allows us to find the matching key pair more efficiently than exhaustive search.

   Here's a brief description of the **Meet-in-the-Middle** attack:

   (a) Assume we have a double encryption scheme where the plaintext is encrypted with *Key A* and then with *Key B*: ciphertext = E(Key B, E(Key A, plaintext)).

   (b) The attacker generates a table, known as the **Meet-in-the-Middle** table, by encrypting all possible values of the first encryption using different keys: intermediate = E(Key A, plaintext) for all possible Key A values.

   (c) The attacker also computes all possible values of the second encryption using different keys: candidate = E(Key B, ciphertext) for all possible Key B values.

   (d) By comparing the values in the **Meet-in-the-Middle** table with the computed candidates, the attacker looks for a match. If a match is found, it indicates that the combination of *Key A* and *Key B* is the correct key pair.

   (e) The attacker has now discovered the correct key pair without having to exhaustively search the entire key space.

   This attack demonstrates a clever method to reduce the computational effort required to find the correct key pair in a double encryption scheme, making it a valuable technique for cryptanalysis in certain scenarios.

5. Key whitening is a technique that enhances the security of cryptographic algorithms by combining the original key with a whitening key to increase complexity and randomness in the resulting key. It adds an extra layer of mixing or diffusion, making it harder for attackers to exploit patterns or vulnerabilities.

15.10.5 Ex 5. Firewalls

1. Can a packet filtering firewall detect malware? Discuss.
2. Provide the iptables rules for the following scenario:
   (a) Default policy is allow for all chains
   (b) Packets are delivered to the LAN through the ethernet adapter eth0
   (c) TCP segments are delivered to the LAN with a maximum speed of 3 per second (other segments will be dropped)

Solution:

1. A packet filtering firewall alone typically cannot detect malware in the traditional sense. Packet filtering firewalls primarily focus on filtering network traffic based on predetermined rules, such as IP addresses, ports, and protocols. Their main purpose is to control and monitor the flow of network traffic to enforce security policies.

   While packet filtering firewalls can help prevent unauthorized access to a network and mitigate certain types of network-based attacks, they do not have the ability to inspect the content of the packets or analyze their payload for malware detection. They primarily operate at the network layer (Layer 3) and transport layer (Layer 4) of the network stack, where they make decisions based on IP addresses, ports, and protocols.

2. you can use the following rules:
   (a) **Setting Default Policies to Allow All Chains**

   To set the default policy to allow for all chains, the following commands can be used:

- `iptables -P INPUT ACCEPT`
- `iptables -P FORWARD ACCEPT`
- `iptables -P OUTPUT ACCEPT`

These commands set the default policy to ACCEPT for the INPUT, FORWARD, and OUTPUT chains, allowing all traffic by default.

(b) **Configuring Rules for Inbound Traffic on the eth0 Interface**

To configure rules for inbound traffic on the eth0 interface, the following commands can be used:

- `iptables -A INPUT -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT`
- `iptables -A INPUT -i eth0 -p tcp --syn -m limit --limit 3/second -j ACCEPT`
- `iptables -A INPUT -i eth0 -j DROP`

These commands specify the following rules:

- The first rule allows incoming traffic on the eth0 interface that is related to an established connection.
- The second rule allows incoming TCP packets with the SYN flag set (indicating the start of a new connection) at a maximum rate of 3 per second.
- The third rule drops any remaining incoming packets on the eth0 interface.

These rules help control and filter inbound traffic on the eth0 interface by accepting established connections and limiting the rate of new TCP connections while dropping other packets.

(c) **Configuring Rules for Outbound Traffic on the eth0 Interface (if needed)**

If needed, rules can be configured for outbound traffic on the eth0 interface using the following commands:

- `iptables -A OUTPUT -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT`

- `iptables -A OUTPUT -o eth0 -j DROP`

These commands specify the following rules:

- The first rule allows outgoing traffic on the eth0 interface that is related to an established connection.
- The second rule drops any other outgoing packets on the eth0 interface.

These rules help control and filter outbound traffic on the eth0 interface by accepting established connections and dropping other packets if needed.