# Workbook

## Network & Information Security

## (CT-460)

## Final Year


**Name** _____

**Roll No** _____

**Batch** _____

**Year** _____

**Department** _____

**Department of Computer Science & Information Technology**

**NED University of Engineering & Technology**

# Workbook

## Network & Information Security

## (CT-460)

## Final Year

Prepared by

Mr. Mehboob Ahmed

Approved by

Chairman

Department of Computer Science & Information Technology

**Department of Computer Science & Information Technology**

**NED University of Engineering & Technology**

# Table of Contents

# LAB 1 CAESAR CIPHER AND ROT-13 QUESTIONS

1)explain the working of caesar cipher and rot-13?

2)what is the difference between rot13 and caesar.how can caesar cipher become rot-13?

3)using brute force, break the encyrpted text "pgf wpkxgtukva", what is the key?

4)why is rot-13 considered an inverse of itself? Encrypt the same text twice using the same key and see the result?

5)what is rot-26 and how will it work, give an example of encryption and decryption?

6)what does frequency analysis tell us about any encryption algorithm knowing that some letters are more common in the english language than others. how can this be used in cryptanalysis?(take the default text in cryptool and run analysis->tools->histogram)

7)what is a histogram and digram. what are three most common single letters in the English language and double letters (digram)?

# LAB 2: VIGENERE CIPHER QUESTIONS

1) Explain the working of the vigenere cipher and the term polyalphabetic substitution. Encrypt the text "NED University" with a key "karachi". what is the result.

2) Decrypt the text "LXFOPVEFRNHR" using the key "LEMONLEMONLE" and explain the process.

3) Using brute force decrypt the text "ZL ANZR VF XUNA NAQ V NZ ABG N GREEBEVFG" using vignere cipher where the key is a single letter.

3) How would you define the vigenere cipher with respect to caesar cipher where the key length is 1.

4) What is atbash substitution, how does it work? Is the atbash a weak or strong cipher and why?

5) A few English words 'Atbash' into other English words. For example, "hob"="sly", "hold"="slow", "holy"="slob", "horn"="slim", "zoo"="all", "irk"="rip", "low"="old", "glow"="told", and "grog"="tilt." Some other English words Atbash into their own reverses, e.g., "wizard" = "draziw. Prove this using cryptool and state the advantage or disadvantage of this?

6) If we go in to the atbash menu and select key entry: remaining characters are filed in ascending order, what is the effect of choosing a key 'LEMON' in one instance and in another instance the key 'ZXC'. Which key is stronger?

# NIS LAB 3: RAIL FENCE, ROUTE & PLAYFAIR CIPHER

## QUESTIONS

Q1) Encrypt the message **"THIS IS A SECRET MESSAGE "**, using rails 4 on paper and verify the results.(do not insert dummy characters, in text options select uppercase letter)

Q2) Explain the decryption process of rail fence. Solve using brute force **“TAR HFIEEYRH PACPLI”** where the number of rails is X on paper.(hint:the first word is THE)

Q3) Describe the working of both railfence and route cipher?

Q4) Encrypt **“This is a secret message”** using a 3x7 matrix using route cipher where there are 3 rows and a clockwise spiral pattern and see how it would be decrypted?

Q5) Decrypt the text where a 7x5 matrix and spiral inward technique is used **“XTEAN ITROB ATSYV NTEDX OEHOM EHSOE SPBUI”**?

Q6) Encrypt and Decrypt any sentence using Playfair cipher and show the steps involved?

# Railfence Cipher

railfence is a transposition cipher. also known as zigzag cipher.

If we consider that we have 3 "rails" and a message of

Cleartext='WE ARE DISCOVERED. FLEE AT ONCE'

Encryption process=

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Encrypted text=

WECRL TEERD SOEEF EAOCA IVDEN

# Route Cipher

A Route Cipher is very similar to a Rail Fence cipher with one exception. You still write the message vertically in columns, but instead of reading off the secret message horizontally, you read it off using a predetermined pattern. For example, lets use a spiral pattern for this one:

```
T S A C T S G
H I S R M S E
I S E E E A J
```

The last J is just a random letter to fill in the space. If we are using a clockwise spiral to read off the message, then it becomes:

TSACTSGEJAEEESIHISRMSE

# Playfair: Playing Fair Since 1854

PLAYFAIR is a polygraphic substitution cipher.

First, we need to create our password matrix:

1. Pick a password. Remove all not alphabetical characters and remove duplicate letters so that at most only one occurrence of each letter of the alphabet appears. For example "Question Authority" becomes "QUESTIONAHRY" (case doesn't matter).

2. Create a 5 by 5 matrix and enter each letter of the password in the spaces. There are multiple ways of placing the password, but the most common way to do it is enter the letters starting at the top, left-most position and move right across the first row, then second row, etc. Here is a matrix using the previous example:

   Q U E S T
   I O N A H
   R Y - - -
   - - - - -
   - - - - -

3. There a 25 total spots in this matrix. Fill the rest of the alphabet. Since there's 26 letters of the alphabet. Either replace all instances of J with I, or discard instances of the letter Q.

   Replace J with I

   Q U E S T
   I O N A H
   R Y B C D
   F G K L M
   P V W X Z

   Omit Q

   U E S T I
   O N A H R
   Y B C D F
   G J K L M
   P V W X Z

Now that we have the password matrix, we can encrypt text.

1. encrypt "Joel enjoys programming on Friday nights! Yah". Split the text we are encrypting into two-letter groups (called digraphs).

   "IO EL EN IO YS PR OG RA MX MI NG ON FR ID AY NI GH TS YA HX". There are three edge-cases you need to consider.

   1. If you run into a digraph that has two of the same letters, split the double letters up with an uncommon character. X is most commonly used. For example the digraph MM (from "programming") becomes MX.

2. The other edge case you need to consider is if there are an odd number of letter in the text phrase. When this happens, include X on the end of the phrase. So Mr. H a Ms. X.
3. Since the rule used is replace J with I rule to create my 5 by 5 matrix. As you can see, all the J's in my input text (from "Joel" and from "enjoys") are changed into I's.

2. So now we have a list of digraphs which roughly resemble our phrase which we plan on encrypting. Every digraph will be encrypted into another corresponding digraph. There are three cases we need to look at.
    1. Take the first digraph in my list (in my case it is IO) and find I and O on my grid. The two letters appear on the same row of the matrix.

        Q U E S T
        I O N A H
        R Y B C D
        F G K L M
        P V W X Z

    2. To get the corresponding encrypted digraph when the two letters are on the same row, shift one space to the right from each letter. So I would become O and O would become N. If the rightward shift takes you off the edge of the matrix, loop around to the other side. For example, the digraph IH would become OI. KM would become LF.
    3. The next case you need to look at is if the two letters of a digraph are in the same column. Let's take the third digraph in my example, which is EN.

        Q U E S T
        I O N A H
        R Y B C D
        F G K L M
        P V W X Z

    4. Very similar to the same row case, in this case you shift one space down. Again, if the shift takes you off the edge of the matrix, just loop back around to the top. EN would become NB. CL would become LX. HZ would become DT.
    5. The last case you need to look at is if the two letters are not in the same row or column. In this case, we say they form the opposite corners of a rectangle. To get the digraph, you just take the other two corners of the rectangle. Let's take the digraph RA.

        Q U E S T
        I O N A H
        R Y B C D

F G K L M
P V W X Z

6. Picture the R and A spaces in the matrix above forming a rectangle, where A is the top-right corner and R is the bottom-left. To get the encrypted digraph for this example, just take the letter in the corners to the right or left of R and A. So in this case, A becomes I and R becomes C. Notice that R does not become I. You take the corner that is to the right or left of the letter you are currently encrypting. Here are some more examples to make sure you get it. The digraph QK becomes EF. HL becomes AM.

3. Once you've done that to all of your digraphs.

   "Joel enjoys programming on Friday nights! Yah!"

   becomes "ONSKNBONCUQFYVCILZFHOKNAPFHROCAOMOQTCOAZ".

---

Decryption is very similar to the encryption process. Given you have the password, create the 5 by 5 password matrix just as you did above. You need to make sure to use the same omission rule (i.e. omit Q or replace J with I) as you used to encrypt the text.

- Separate the encrypted phrase into digraphs and use the inverse of the cases I outlined above to decrypt each encrypted digraph.
  - For digraphs which occur in one row, shift one space to the left (looping around if necessary).
  - For digraphs which occur in one column, shift one space up (looping if necessary).
  - For digraphs which form a rectangle, use the same exact process as in the encryption process (use the opposite corner in the same row).

Answer1 “4 rails”= TATG HS S EMAEII ERESSCS

ANSWER2 =“THE PLAYFAIR CIPHER”, key 5

Answer5= "abort the mission, you have been spotted"

| | | | | |
|---|---|---|---|---|
| A | B | O | R | T |
| T | H | E | M | I |
| S | S | I | O | N |
| Y | O | U | H | A |
| V | E | B | E | E |
| N | S | P | O | T |
| T | E | D | X | X |

## LAB 4 CODES AND CIPHERS QUESTIONS

## Codes and Ciphers :: Bifid Cipher

The Bifid Cipher uses a Polybius Square to encipher a message in a way that makes it fairly difficult to decipher without knowing the secret. This is because each letter in the ciphertext message is dependent upon two letters from the plaintext message. As a result, frequency analysis of letters becomes much more difficult.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | A | B | C | D | E |
| 2 | F | G | H | I | K |
| 3 | L | M | N | O | P |
| 4 | Q | R | S | T | U |
| 5 | V | W | X | Y | Z |

The first step is to use the Polybius Square to convert the letters into numbers. We will be writing the numbers vertically below the message.

```
secret message
411414 3144121
353254 2533125
```

The numbers are now read off horizontally and grouped into pairs.

```
41 14 14 31 44 12 13 53 25 42 53 31 25
```

The Polybius Square is used again to convert the numbers back into letters which gives us our ciphertext.

```
qddltbcxkrxlk
```

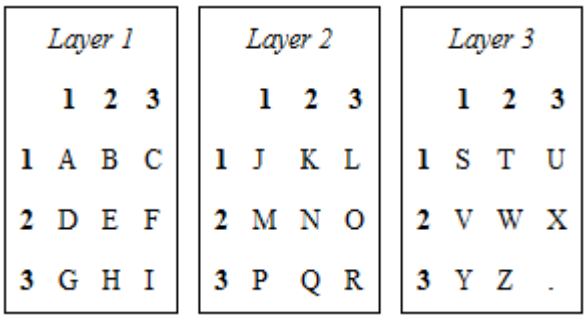
Since the first letter in the plaintext is encoded into the first and middle letters of the ciphertext, the recipient of the message must have the entire message before they can decode it. This means that if part of the ciphertext is discovered by a third party, it is unlikely that they will be able to crack it.

To decipher a Bifid encrypted message, you first convert each letter into its corresponding number via the Polybius Square. Now, divide the long string of numbers into two equal rows. The digit in the top row and the digit in the bottom row will together reference the decoded letter in the Polybius Square.

The Bifid Cipher can be taken into three dimensions to slightly increase the security of the message. This new cipher is called the Trifid Cipher.

# Trifid Cipher

The Trifid Cipher is the Bifid Cipher taken to one more dimension. Instead of using a 5x5 Polybius Square, you use a 3x3x3 cube. Otherwise everything else remains the same. As with the Bifid Cipher, the cube can be mixed to add an extra layer of protection.



| *Layer 1* | | | |
|---|---|---|---|
| | **1** | **2** | **3** |
| **1** | A | B | C |
| **2** | D | E | F |
| **3** | G | H | I |

| *Layer 2* | | | |
|---|---|---|---|
| | **1** | **2** | **3** |
| **1** | J | K | L |
| **2** | M | N | O |
| **3** | P | Q | R |

| *Layer 3* | | | |
|---|---|---|---|
| | **1** | **2** | **3** |
| **1** | S | T | U |
| **2** | V | W | X |
| **3** | Y | Z | . |

```
secret message
311213 2133111
123322 1211112
121321 2211132
```

The numbers are now read off horizontally and grouped into triplets.

```
311 213 213 311 112 332 212 111 121 213 212 211 132
```

The cube is used again to convert the numbers back into letters which gives us our ciphertext.

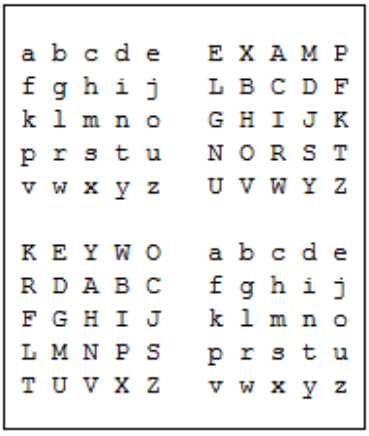
```
sppsdxmabpmjf
```

## Codes and Ciphers :: Four-square Cipher

The four-square cipher encrypts pairs of letters (digraphs) and is thus less susceptible to frequency analysis attacks.

The four-square cipher uses four 5 by 5 matrices arranged in a square. Each of the 5 by 5 matrices contains the letters of the alphabet (usually omitting "Q" or putting both "I" and "J" in the same location to reduce the alphabet to fit). In general, the upper-left and lower-right matrices are the "plaintext squares" and each contain a standard alphabet. The upper-right and lower-left squares are the "ciphertext squares" and contain a mixed alphabetic sequence.

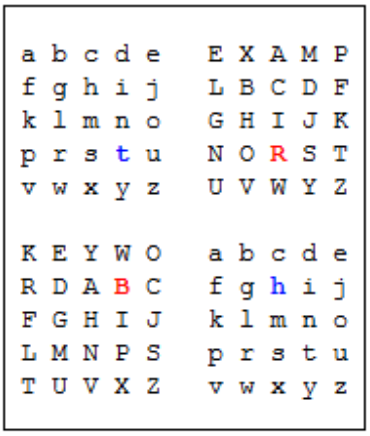
To generate the ciphertext squares, one would first fill in the spaces in the matrix with the letters of a keyword or phrase (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order. The four-square algorithm allows for two separate keys, one for each of the two ciphertext matrices. In the example to the right, "EXAMPLE" and "KEYWORD" have been used as keywords.




To encrypt a message you would first split the message into digraphs. "This is a secret message" would become:

```
TH IS IS AS EC RE TM ES SA GE
```

Once that was complete, you would take the first pair of letters and find the first letter in the upper left square and the second letter in the lower right square. In this example we are enciphering TH, so we locate T and H in the grid below (see blue characters). Now, we find the intersections of the rows and columns of the plain text letters. In this example, they have been highlighted in red (R and B). These new letters are the enciphered digraph (RB).




You continue enciphering digraphs in this way until you reach the end of the message. To continue our example, "This is a secret message" would be enciphered as:

Q1) DECRYPT the text 'saontldryhttk' using bifid cipher?

Q2) Explain the process and Encrypt any sentence using bifid cipher?

Q3) Explain the encryption process of tifid cipher and decrypt the following text

'sphwbt'? Why is the answer different to the example above?

Q4) What would be the enciphered text of "This is a secret message" using four square cipher.

Answer 1) tobeornottobe

Answer 3) decipheratrifidencryptedmessage

Answer 4)

```
RB CP CP AL AO TE RI AS NY FE
```

# LAB 5 HILL CIPHER QUESTIONS

Description

In classical cryptography, the **Hill cipher** is a polygraphic substitution cipher based on linear algebra.The Hill cipher is an example of a block cipher. A block cipher is a cipher in which groups of letters are enciphered together in equal length blocks. The Hill cipher was developed by Lester Hill and introduced in an article published in 1929[1].

Example

**Encipher**

In order to encrypt a message using the Hill cipher, the sender and receiver must first agree upon a key matrix *A* of size *n* x *n*. *A* must be invertible mod 26. The plaintext will then be enciphered in blocks of size *n*. In the following example *A* is a 2 x 2 matrix and the message will be enciphered in blocks of 2 characters.

Key Matrix: $A = \begin{bmatrix} 3 & 25 \\ 24 & 17 \end{bmatrix}$

Message: MISSISSIPPI

The first block MI corresponds to the matrix $\begin{bmatrix} 12 \\ 8 \end{bmatrix}$. The sender will then calculate:

$$A \begin{bmatrix} 12 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \end{bmatrix} \pmod{26}$$

The first two letters of the ciphertext correspond to 2,8 and are therefore CI. This step is repeated for the entire plaintext. If there are not enough letters to form blocks of 2, pad the message with some letter, say Z.

The message:

MI SS IS SI PP IK

will be enciphered as:

CI KK GE UW ER OY

Notice that the repeated digraphs IS, SS and repeated letters S and P in the plaintext are masked using the Hill cipher.

**Decipher**

To decipher a message, first calculate the inverse of the key A.

$$A^{-1} = \det(A)^{-1} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \pmod{26}$$

Then multiply the inverse of the key by each pair of ciphertext letters (mod 26) to recover the original text.

Key Matrix: $A = \begin{bmatrix} 3 & 17 \\ 8 & 25 \end{bmatrix}$

Encrypted Message: CIKKGEUWEROY

The receiver will calculate:

$$A^{-1} \begin{bmatrix} 2 \\ 8 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} \pmod{26}$$

to decrypt the message. The first two letters are 12, 8 which correspond to M and I. The receiver will repeat this step for every pair of letters in the ciphertext to recover the original message MISSISSIPPIK.

**LAB 6 DES QUESTIONS**

# DES

CrypTool offers two variations of the DES cipher, electronic codebook (ECB) and cipher-block chaining (CBC), these are both known as *block cipher modes of operation*. As discussed in the report, DES breaks down a plaintext message into 64 bit sized blocks, these are then converted to 64 bit blocks of ciphertext. Conventionally these ciphertext blocks have no relation to one another's encoding, this is the former of the two methods, ECB. There is however a weakness with this method, the DES encodes two identical plaintext blocks as two identical ciphertext blocks, in a message where there could be large repletion of a 64 bit block, say an image, this could reveal patterns in the ciphertext message and ultimately lead to it been deciphered. CBC is a method that overcomes this by combining each block with its preceding block with an exclusive-OR logic gate (like the Vernam cipher). A factor that can make this cipher at times impractical is that the whole message must be transmitted error free in order for it to be comprehensibly deciphered at the receiver side. With the ECB method an error in one block will only effect the deciphering of that one block and not the entire message.

## Using DES with CrypTool

Open a new file and type a plaintext message or alternatively open the file **examples/Startingexample-en.txt**. Next click from the menu **Crypt/Decrypt > Symmetric (modern) > DES (ECB)…** This presents a key selection window, this key must be 64 bits long, which equates to 16 hexadecimal figures. For simplicity use the default key of:

00 00 00 00 00 00 00 00

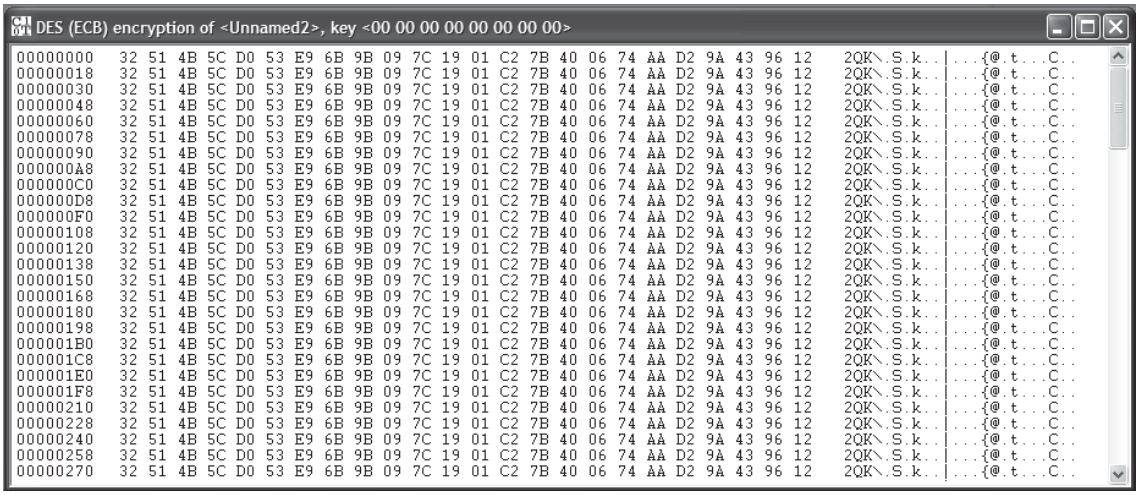
Select **Encrypt** and there should be presented a window showing the data encrypted in hexadecimal form and its corresponding ASCII representation. To decrypt the message again select **Crypt/Decrypt > Symmetric (modern) > DES (ECB)…** Use the same key and select **Decrypt**, and the original message will be displayed in hexadecimal representation. Selecting **View > Show as text** displays it in ASCII; you may also notice some of the formatting is lost in the process or some padding is added.

Encrypt the same message using the same process as above only selecting **Crypt/Decrypt > Symmetric (modern) > DES (CBC)…** instead. Comparing the two encrypted messages, you should notice they start with the same 8 bytes, but then they are not the same but both are illegible.

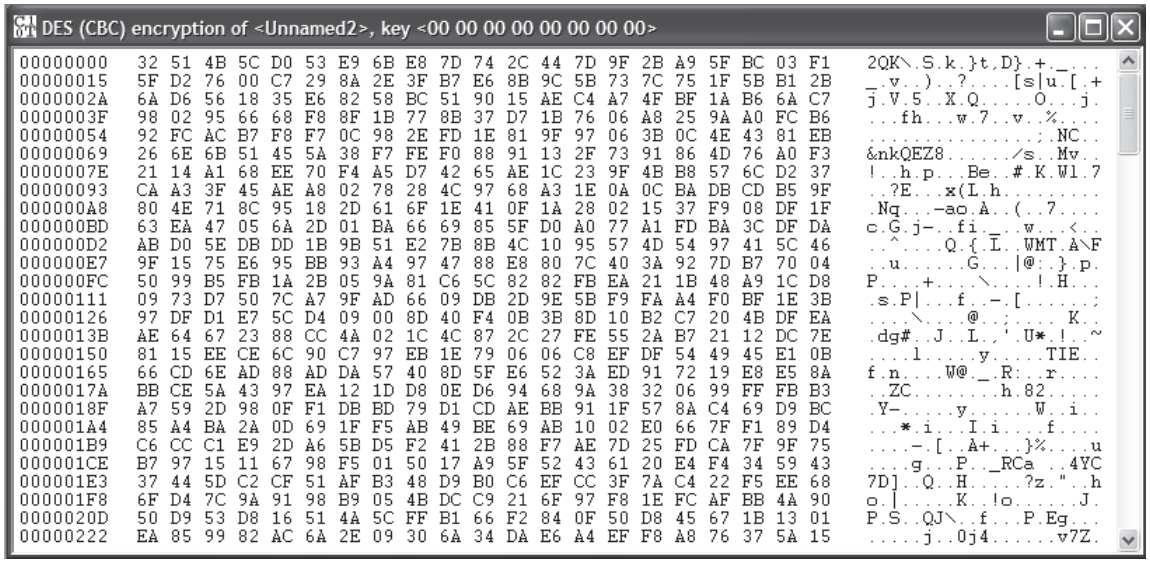
## Comparison of ECB and CBC

Now create a new plaintext file, choose a word and type it in, now copy and paste this word until there are about 100 repetitions. For the below example the word *Cryptography* was used:


Encrypt this first with the **ECB** variant of the **DES** cipher, there should be a notable pattern present (the word is 12 bytes long, 8 bytes form a block, all 3 blocks you see the same pattern):




A pattern like this could potentially help a cryptanalyst decipher the data. Next encrypt the same message using the **CBC** method.




As shown with the above example there should be no obvious pattern present making this a much stronger form of DES encryption.

**Q1.**What is ECB and CBC and their purpose. How do they differ?

**Q2.** For each of the following say whether ECB or CBC would be most appropriate and give a brief explanation as to why.

An online bank statement

An encrypted VoIP session

Viewing of a website using TCP/IP

**Q3.** The DES cipher can be demonstrated with the aid of the ANIMAL Animator. Select Indiv. Procedures > Visualization of Algorithms > DES... and watch through the whole demonstration. This should help to reinforce understanding of this cipher?

**Q4.** Why are the following keys considered to be weak keys of DES. Think about applying these keys to cryptool preferably trying to encrypt text with these keys twice.

K1= 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
K2= F E F E F E F E F E F E F E F E
K3= 1 F 1 F 1 F 1 F 0 E 0 E 0 E 0 E
K4= E 0 E 0 E 0 E 0 F 1 F 1 F 1 F 1

Q5. Why are the following keys considered to be semi-weak keys. The keys are shown in pairs and have some correlation to each other. Think about applying these keys to cryptool preferably trying to encrypt text with these keys twice.

The semi-weak keys of the DES are [DP84]:

$$
\begin{array}{lcl}
K_{1,1} & = & 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \\
K_{1,2} & = & \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,1 \\
K_{2,1} & = & 1\,\mathrm{F} \;\; \mathrm{E}\,0 \;\; 1\,\mathrm{F} \;\; \mathrm{E}\,0 \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,1 \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,1 \\
K_{2,2} & = & \mathrm{E}\,0 \;\; 1\,\mathrm{F} \;\; \mathrm{E}\,0 \;\; 1\,\mathrm{F} \;\; \mathrm{F}\,1 \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,1 \;\; 0\,\mathrm{E} \\
K_{3,1} & = & 0\,1 \;\; \mathrm{E}\,0 \;\; 0\,1 \;\; \mathrm{E}\,0 \;\; 0\,1 \;\; \mathrm{F}\,1 \;\; 0\,1 \;\; \mathrm{F}\,1 \\
K_{3,2} & = & \mathrm{E}\,0 \;\; 0\,1 \;\; \mathrm{E}\,0 \;\; 0\,1 \;\; \mathrm{F}\,1 \;\; 0\,1 \;\; \mathrm{F}\,1 \;\; 0\,1 \\
K_{4,1} & = & 1\,\mathrm{F} \;\; \mathrm{F}\,\mathrm{E} \;\; 1\,\mathrm{F} \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,\mathrm{E} \\
K_{4,2} & = & \mathrm{F}\,\mathrm{E} \;\; 1\,\mathrm{F} \;\; \mathrm{F}\,\mathrm{E} \;\; 1\,\mathrm{F} \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,\mathrm{E} \;\; \mathrm{F}\,\mathrm{E} \;\; 0\,\mathrm{E} \\
K_{5,1} & = & 0\,1 \;\; 1\,\mathrm{F} \;\; 0\,1 \;\; 1\,\mathrm{F} \;\; 0\,1 \;\; 0\,\mathrm{E} \;\; 0\,1 \;\; 0\,\mathrm{E} \\
K_{5,2} & = & 1\,\mathrm{F} \;\; 0\,1 \;\; 1\,\mathrm{F} \;\; 0\,1 \;\; 0\,\mathrm{E} \;\; 0\,1 \;\; 0\,\mathrm{E} \;\; 0\,1 \\
K_{6,1} & = & \mathrm{E}\,0 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{E}\,0 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{F}\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{F}\,1 \;\; \mathrm{F}\,\mathrm{E} \\
K_{6,2} & = & \mathrm{F}\,\mathrm{E} \;\; \mathrm{E}\,0 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{E}\,0 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{F}\,1 \;\; \mathrm{F}\,\mathrm{E} \;\; \mathrm{F}\,1
\end{array}
$$

# LAB 7 TRIPLE DES QUESTIONS

**Q1.** Perform encryption using Triple DES (CBC) using key 1234ABCD and all zero on the plaintext provided. This will help you in performing the rest of the questions?

**Q2.** Using the .hex file named 'tripledesbruteforce' that is provided; decrypt the file using brute force, where the key partially begins with 5432 **** 000000000000000000000000. The asterisk represents alphabets in a specific order. To perform brute force, go to Analysis -> Symmetric Encryption (modern). What is the key?

**Q3.** If we use the key 5432 ABCD ********** how much time is required to decrypt the text using Triple DES(CBC)? Write your answers without the exponents.

**Q4.** Is there a difference in the time required to decrypt using brute force with DES(CBC) and Triple DES(CBC) and why is that so? Use the text "NED UNIVERSITY CSIT". Then encrypt it with the key "5432 DCBA 00000000". Now perform brute force using they key 5432 ABCD ********.

Similarly take the same text and encrypt with triple des (CBC) using the key "5432 DCBA 000000000000000000000000" and then perform brute force with the key 5432 DCBA **********?

**Q5.** Do weak keys have the same effect on Triple DES(ECB) as they do on simple Triple DES(CBC). Use the text "NED UNIVERSITY CSIT. SPRING FESTIVAL 2015" as an example. Why do you think that is so?

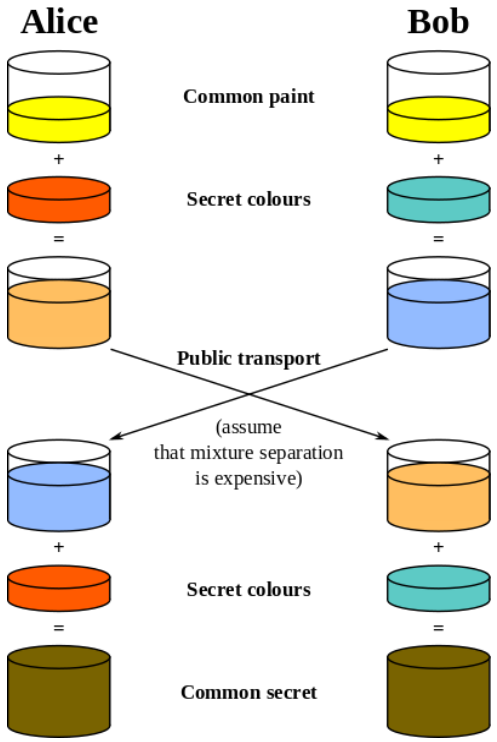
**Q6.**Do the weak keys have the same effect on Triple DES(ECB) and DES (ECB). Use the text "NED UNIVERSITY CSIT. SPRING FESTIVAL 2015" as an example. What is the result?

K1= 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
K2= F E F E F E F E F E F E F E F E
K3= 1 F 1 F 1 F 1 F 0 E 0 E 0 E 0 E
K4= E 0 E 0 E 0 E 0 F 1 F 1 F 1 F 1

# LAB 8 DIFFIE–HELLMAN KEY EXCHANGE QUESTIONS

Diffie–Hellman key exchange is a specific method of exchanging cryptographic keys. It is one of the earliest practical examples of key exchange implemented within the field of cryptography. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

Diffie–Hellman establishes a shared secret that can be used for secret communications while exchanging data over a public network. The following diagram illustrates the general idea of the key exchange by using colors instead of a very large number. The crucial part of the process is that Alice and Bob exchange their secret colors in a mix only. Finally this generates an identical key that is mathematically difficult (impossible for modern supercomputers to do in a reasonable amount of time) to reverse for another party that might have been listening in on them. Alice and Bob now use this common secret to encrypt and decrypt their sent and received data. Note that the starting color (yellow) is arbitrary, but is agreed on in advance by Alice and Bob. The starting color is assumed to be known to any eavesdropping opponent. It may even be public.



**Q1.** Perform Diffie–Hellman key exchange with a partner. Where both partners are Alice and Bob and calculate your values and compare them.

- Alice and Bob agree to use a prime number $p = 23$ and base $g = 5$.
- Alice chooses a secret integer $\boldsymbol{a} = \mathbf{6}$, then sends Bob $A = g^{\boldsymbol{a}} \bmod p$. Don't tell Bob the values of a.
  - $A = 5^{\mathbf{6}} \bmod 23$
  - $A = ?$
- Bob chooses a secret integer $\boldsymbol{b} = \mathbf{15}$, then sends Alice $B = g^{\boldsymbol{b}} \bmod p$. Don't tell Alice the value of b.
  - $B = 5^{\mathbf{15}} \bmod 23$
  - $B = ?$
- Alice computes $\boldsymbol{s} = B^{\boldsymbol{a}} \bmod p$
  - $\boldsymbol{s} = 19^{\mathbf{6}} \bmod 23$
  - $\boldsymbol{s} = \mathbf{?}$
- Bob computes $\boldsymbol{s} = A^{\boldsymbol{b}} \bmod p$
  - $\boldsymbol{s} = 8^{\mathbf{15}} \bmod 23$
  - $\boldsymbol{s} = \mathbf{?}$
- Alice and Bob now share a secret the number **s**=?

**Q2.** Perform Diffie–Hellman key exchange and verify the result with Cryptool? Also try the exchange and use cryptool to generate the prime numbers, base etc.

**Q3.** If a third partner is used to find the key, while he or she knows **p, g A, B**. Do you think it would be possible for that person to find the key value. What values need to be known?

# LAB 9 RSA ALGORITHM QUESTIONS

RSA was released in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman. Its design was based on that of the Diffie-Hellman concept and has since become the most widely used public key system to date. The cipher comprises of two keys, a public key and a private one. As the names suggest the public key can be known by any one, whereas the private key is to remain secret. When a message is sent from the client, it encrypts it using the servers public key, once encrypted this message may only be decrypted with the private key. The mathematical concept behind the cipher can be broken down into a series of steps which are explained below:

1. Choose two large prime numbers, $p$ and $q$. To aid in understanding of this cipher, an example is included (For mathematical simplicity, small prime numbers for $p$ and $q$ are used in the example, in practice for efficient security these numbers should be large enough to generate a product in excess of 300 decimal digits long)

   **$p$ as 17**

   **$q$ as 11**
2. Calculate $n$ by multiplying $p$ by $q$:

   **$n = pq$ = 17 * 11 = 187**

   $n$ represents the modulus to be used for both the public and private key operations and consequently determines the maximum size of the message's block size.
3. Calculate the *totient* for $n$, $\phi(n)$ (Like the other mathematical concepts mentioned in this section, a full explanation of the *totient* can be found in the earlier chapter of *Mathematical Principles*). An expression for this is:

   $\phi(n) = (p-1)(q-1)$

   $\phi(n)$ **= (17-1)*(11-1) = 160**
4. The integer, $e$, is to be chosen so as: **1 < $e$ <** $\phi(n)$

   $e$ should also be coprime to $\phi(n)$, so $e$ and $\phi(n)$ should share no factors other

than 1 (and -1). For our example we will use the value 7

***e=7***

*e* represents the public key exponent and may be released to the recipient of a transmission.

5. Next, the private key component, *d*, should be calculated. This number is selected so as it fits the following formula:

$$de = 1 + k\phi(n)$$

where *k* is a positive integer. That is to say the public key, *e*, and private key, *d*, have a congruence relationship

$$de \equiv 1 \pmod{\phi(n)}$$

Additionally, the value for *d* must be less than $\phi(n)$. There should only be one value that satisfies the above equation, and in the case of our example that is 23.

***de* = 23 * 7 = 161**

**161 mod 160 = 1**

Therefore:

***d* = 23**

6. *e* is then combined with *n* to make the public key {7, 187} and *d* with *n* to make the private key {23, 187}.
7. To encrypt the plaintext, *PT*, as ciphertext, *CT*, use the following formula:

**$CT=PT^e \bmod n$**

This ciphertext may be transmitted to the recipient.

In the context of our example we take the plaintext value 88:

**PT = 88**

**$CT=88^7 \bmod 187 = 11$**

8. To decipher the ciphertext, apply the following formula:

**$PT = CT^d \bmod n$**

**$PT = 11^{23} \bmod 187 = 88$**

Factorising n

The number n, shown above, is the product of two large prime numbers, p and q, consequently it is possible to derive both p and q from the value n, as there will only be one pair of prime numbers that multiply to give n, accordingly n is known as a semiprime (also called biprime). This may be considered an obvious security risk, however, factorising the value of n, should n be sufficiently large, would take a tremendous degree of computing power. To illustrate this there is an ongoing RSA challenge, where the RSA laboratories have offered sums of money to any one able to find the two prime numbers that multiply to give their published semiprime n. After 16 years of running the challenge, in May 2005, the largest semiprime to be broken has been the RSA-200, a 200 long digit, that took a cluster of 80 2.2 Ghz computers continually running for 3 months to calculate. This is estimated to be the equivalent of a single 2.2 Ghz machine running for 55 continual years. (source http://www.rsa.com/rsalabs/node.asp?id=2879). As it suggested n should be no less than 300 digits, RSA can still be considered secure.

Q1) Here is an example of RSA encryption and decryption. The parameters used here are small. Verify your results with cryptool?

1. Choose two distinct prime numbers, such as

   $p = 61$ and $q = 53$

2. Compute $n = pq$ giving

   n=?

3. Compute the totient of the product as $\varphi(n) = (p - 1)(q - 1)$ giving

   $\varphi(3233) = ?$

4. Choose any number $1 < e < 3120$ that is coprime to 3120. Choosing a prime number for $e$ leaves us only to check that $e$ is not a divisor of 3120.

   Let $e = 17$

5. Compute $d$, the modular multiplicative inverse of $e$ (mod $\varphi(n)$) yielding

   d=?

The **public key** is ($n$ = 3233, $e$ = 17). For a padded plaintext message $m$, the encryption function is

$$c(m) = m^{17} \bmod 3233$$

The **private key** is ($n$ = 3233, $d$ = 2753). For an encrypted ciphertext $c$, the decryption function is

$$m(c) = c^{2753} \mod 3233 = ?$$

For instance, in order to encrypt $m$ = 65, we calculate

$$c = 65^{17} \mod 3233 = ?$$

To decrypt $c$ = 2790, we calculate

$$m = 2790^{2753} \mod 3233 = ?$$

Q2) In RSA, practical difficulty of factoring the product of two large prime numbers is known as the factoring problem. This is what RSA is based on. The prime factors must be kept secret. If the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

If we know N = 639784868795271438588314150 41 (95 bit, 29 decimal digits) and then try this number N = 351573870816322547022741576341143304183 (129 bit, 39 digit). Find the factors using cryptool by going in to Indiv. Procedures -> RSA Cryptosystem -> Factorization of a number. Then enter the number and find the factors. What does this tell you about the difficulty level of finding the factors in both cases? What are the factors in both cases? What algorithm was used last to factorize in both cases? What can be done once the factors are known?

Q3) Using the Key file "[AHMED][MEHBOOB][RSA-1024][1396335918][1]" and the encrypted file provided "Cry-RSA-Unnamed1" and pin code "123". What is the encrypted text? What is the key (the initial 10 digits will do)? The key can be found by going to Digital Signature->PKI->Display and select your key.

# LAB 10 & 11ACCESS CONTROL LISTS QUESTIONS

---

Cisco provides basic traffic filtering capabilities with access control lists (also referred to as *access lists*). Access lists can be configured for all routed network protocols (IP, AppleTalk, and so on) to filter the packets of those protocols as the packets pass through a router.

You can configure access lists at your router to control access to a network: access lists can prevent certain traffic from entering or exiting a network.

## In This Chapter

This chapter describes access lists as part of a security solution. This chapter includes tips, cautions, considerations, recommendations, and general guidelines for how to use access lists.

This chapter has these sections:

- About Access Control Lists
- Overview of Access List Configuration
- Finding Complete Configuration and Command Information for Access Lists

## About Access Control Lists

This section briefly describes what access lists do; why and when you should configure access lists; and basic versus advanced access lists.

This section has the following sections:

- What Access Lists Do
- Why You Should Configure Access Lists
- When to Configure Access Lists
- Basic Versus Advanced Access Lists

### What Access Lists Do

Access lists filter network traffic by controlling whether routed packets are forwarded or blocked at the router's interfaces. Your router examines each packet to determine whether to forward or drop the packet, on the basis of the criteria you specified within the access lists.

Access list criteria could be the source address of the traffic, the destination address of the traffic, the upper-layer protocol, or other information. Note that sophisticated users can sometimes successfully evade or fool basic access lists because no authentication is required.

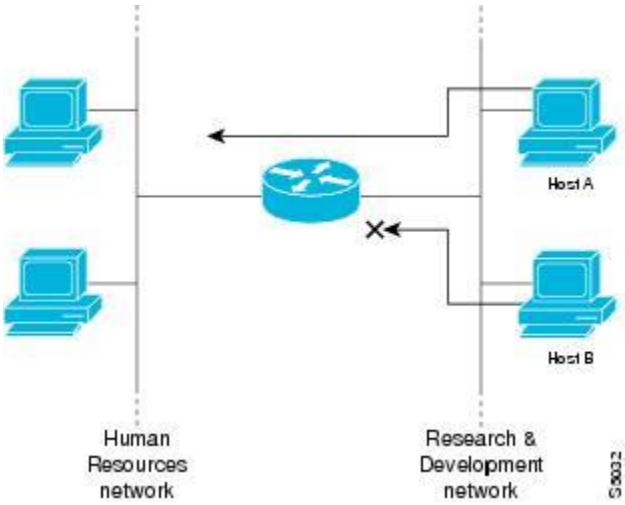
## Why You Should Configure Access Lists

There are many reasons to configure access lists; for example, you can use access lists to restrict contents of routing updates or to provide traffic flow control. One of the most important reasons to configure access lists is to provide security for your network, which is the focus of this chapter.

You should use access lists to provide a basic level of security for accessing your network. If you do not configure access lists on your router, all packets passing through the router could be allowed onto all parts of your network.

Access lists can allow one host to access a part of your network and prevent another host from accessing the same area. In Figure 14, host A is allowed to access the Human Resources network, and host B is prevented from accessing the Human Resources network.

Figure 14 Using Traffic Filters to Prevent Traffic from Being Routed to a Network




You can also use access lists to decide which types of traffic are forwarded or blocked at the router interfaces. For example, you can permit e-mail traffic to be routed, but at the same time block all Telnet traffic.

## When to Configure Access Lists

Access lists should be used in "firewall" routers, which are often positioned between your internal network and an external network such as the Internet. You can also use access lists on a router positioned between two parts of your network, to control traffic entering or exiting a specific part of your internal network.

To provide the security benefits of access lists, you should at a minimum configure access lists on border routers—routers situated at the edges of your networks. This provides a basic buffer from the outside network, or from a less controlled area of your own network into a more sensitive area of your network.

On these routers, you should configure access lists for each network protocol configured on the router interfaces. You can configure access lists so that inbound traffic or outbound traffic or both are filtered on an interface.

Access lists must be defined on a per-protocol basis. In other words, you should define access lists for every protocol enabled on an interface if you want to control traffic flow for that protocol.

---

**Note** Some protocols refer to access lists as *filters*.

---

### Basic Versus Advanced Access Lists

This chapter describes how to use standard and static extended access lists, which are the basic types of access lists. Some type of basic access list should be used with each routed protocol that you have configured for router interfaces.

Besides the basic types of access lists described in this chapter, there are also more advanced access lists available, which provide additional security features and give you greater control over packet transmission. These advanced access lists and features are described in the other chapters within the part "Traffic Filtering and Firewalls."

## Overview of Access List Configuration

Each protocol has its own set of specific tasks and rules that are required in order for you to provide traffic filtering. In general, most protocols require at least two basic steps to be accomplished. The first step is to create an access list definition, and the second step is to apply the access list to an interface.

The following sections describe these two steps:

- Creating Access Lists
- Applying Access Lists to Interfaces

Note that some protocols refer to access lists as *filters* and refer to the act of applying the access lists to interfaces as *filtering*.

## Creating Access Lists

Create access lists for each protocol you wish to filter, per router interface. For some protocols, you create one access list to filter inbound traffic, and one access list to filter outbound traffic.

To create an access list, you specify the protocol to filter, you assign a unique name or number to the access list, and you define packet filtering criteria. A single access list can have multiple filtering criteria statements.

Cisco recommends that you create your access lists on a TFTP server and then download the access lists to your router. This approach can considerably simplify maintenance of your access lists. For details, see the "Creating and Editing Access List Statements on a TFTP Server" section later in this chapter.

The protocols for which you can configure access lists are identified in Table 16.

This section has the following sections:

- Assigning a Unique Name or Number to Each Access List
- Defining Criteria for Forwarding or Blocking Packets
- Creating and Editing Access List Statements on a TFTP Server

## Assigning a Unique Name or Number to Each Access List

When configuring access lists on a router, you must identify each access list uniquely within a protocol by assigning either a name or a number to the protocol's access list.

---

**Note** Access lists of some protocols must be identified by a name, and access lists of other protocols must be identified by a number. Some protocols can be identified by either a name or a number. When a number is used to identify an access list, the number must be within the specific range of numbers that is valid for the protocol.

---

You can specify access lists by names for the following protocols:

- Apollo Domain

- IP
- IPX
- ISO CLNS
- NetBIOS IPX
- Source-route bridging NetBIOS

You can specify access lists by numbers for the protocols listed in Table 16. Table 16 also lists the range of access list numbers that is valid for each protocol.

Table 16 Protocols with Access Lists Specified by Numbers

| Protocol | Range |
|---|---|
| IP | 1-99, 1300-1999 |
| Extended IP | 100-199, 2000-2699 |
| Ethernet type code | 200-299 |
| Ethernet address | 700-799 |
| Transparent bridging (protocol type) | 200-299 |
| Transparent bridging (vendor code) | 700-799 |
| Extended transparent bridging | 1100-1199 |
| DECnet and extended DECnet | 300-399 |
| XNS | 400-499 |
| Extended XNS | 500-599 |
| AppleTalk | 600-699 |
| Source-route bridging (protocol type) | 200-299 |
| Source-route bridging (vendor code) | 700-799 |
| IPX | 800-899 |
| Extended IPX | 900-999 |
| IPX SAP | 1000-1099 |
| Standard VINES | 1-100 |
| Extended VINES | 101-200 |
| Simple VINES | 201-300 |

## Defining Criteria for Forwarding or Blocking Packets

When creating an access list, you define criteria that are applied to each packet that is processed by the router; the router decides whether to forward or block each packet on the basis of whether or not the packet matches the criteria.

Typical criteria you define in access lists are packet source addresses, packet destination addresses, and upper-layer protocol of the packet. However, each protocol has its own specific set of criteria that can be defined.

For a single access list, you can define multiple criteria in multiple, separate access list statements. Each of these statements should reference the same identifying name or number, to tie the statements to the same access list. You can have as many criteria statements as you want, limited only by the available memory. Of course, the more statements you have, the more difficult it will be to comprehend and manage your access lists.

## The Implied "Deny All Traffic" Criteria Statement

At the end of every access list is an implied "deny all traffic" criteria statement. Therefore, if a packet does not match any of your criteria statements, the packet will be blocked.

---

**Note** For most protocols, if you define an inbound access list for traffic filtering, you should include explicit access list criteria statements to permit routing updates. If you do not, you might effectively lose communication from the interface when routing updates are blocked by the implicit "deny all traffic" statement at the end of the access list.

---

## The Order in Which You Enter Criteria Statements

Note that each additional criteria statement that you enter is appended to the *end* of the access list statements. Also note that you cannot delete individual statements after they have been created. You can only delete an entire access list.

*The order of access list statements is important!* When the router is deciding whether to forward or block a packet, the Cisco IOS software tests the packet against each criteria statement in the order in which the statements were created. After a match is found, no more criteria statements are checked.

If you create a criteria statement that explicitly permits all traffic, no statements added later will ever be checked. If you need additional statements, you must delete the access list and retype it with the new entries.

## Creating and Editing Access List Statements on a TFTP Server

Because the order of access list criteria statements is important, and because you cannot reorder or delete criteria statements on your router, Cisco recommends that you create all access list statements on a TFTP server, and then download the entire access list to your router.

To use a TFTP server, create the access list statements using any text editor, and save the access list in ASCII format to a TFTP server that is accessible by your router. Then, from your router, use the **copy tftp:***file_id* **system:running-config** command to copy the access list to your router. Finally, perform the **copy system:running-config nvram:startup-config** command to save the access list to your router's NVRAM.

Then, if you ever want to make changes to an access list, you can make them to the text file on the TFTP server, and copy the edited file to your router as before.

---

**Note** The first command of an edited access list file should delete the previous access list (for example, type a **no access-list** command at the beginning of the file). If you do not first delete the previous version of the access list, when you copy the edited file to your router you will merely be appending additional criteria statements to the end of the existing access list.

---

### Applying Access Lists to Interfaces

For some protocols, you can apply up to two access lists to an interface: one inbound access list and one outbound access list. With other protocols, you apply only one access list which checks both inbound and outbound packets.

If the access list is inbound, when the router receives a packet, the Cisco IOS software checks the access list's criteria statements for a match. If the packet is permitted, the software continues to process the packet. If the packet is denied, the software discards the packet.

If the access list is outbound, after receiving and routing a packet to the outbound interface, the software checks the access list's criteria statements for a match. If the packet is permitted, the software transmits the packet. If the packet is denied, the software discards the packet.

# Configuring IP Access Lists

# Introduction

This document describes how IP access control lists (ACLs) can filter network traffic. It also contains brief descriptions of the IP ACL types, feature availability, and an example of use in a network.

Access the Software Advisor (registered customers only) tool in order to determine the support of some of the more advanced Cisco IOS® IP ACL features.

RFC 1700 contains assigned numbers of well-known ports. RFC 1918 contains address allocation for private Internets, IP addresses which should not normally be seen on the Internet.

**Note:** ACLs might also be used for purposes other than to filter IP traffic, for example, defining traffic to Network Address Translate (NAT) or encrypt, or filtering non-IP protocols such as AppleTalk or IPX. A discussion of these functions is outside the scope of this document.

# Prerequisites

### Requirements

There are no specific prerequisites for this document. The concepts discussed are present in Cisco IOS® Software Releases 8.3 or later. This is noted under each access list feature.

### Components Used

This document discusses various types of ACLs. Some of these are present since Cisco IOS Software Releases 8.3 and others were introduced in later software releases. This is noted in the discussion of each type.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

### Conventions

Refer to Cisco Technical Tips Conventions for more information on document conventions.

# ACL Concepts

This section describes ACL concepts.

### Masks

Masks are used with IP addresses in IP ACLs to specify what should be permitted and denied. Masks in order to configure IP addresses on interfaces start with 255 and have the large values on the left side, for example, IP address 209.165.202.129 with a 255.255.255.224 mask. Masks for IP ACLs are the reverse, for example, mask 0.0.0.255. This is sometimes called an inverse mask or a wildcard mask. When the value of the mask is broken down into binary (0s and 1s), the results determine which address bits are to be considered in processing the traffic. A 0 indicates that the address bits must be considered (exact match); a 1 in the mask is a "don't care". This table further explains the concept.

| Mask Example | |
|---|---|
| network address (traffic that is to be processed) | *10.1.1.0* |
| mask | *0.0.0.255* |
| network address (binary) | *00001010.00000001.00000001.00000000* |
| mask (binary) | *00000000.00000000.00000000.11111111* |

Based on the binary mask, you can see that the first three sets (octets) must match the given binary network address exactly (00001010.00000001.00000001). The last set of numbers are "don't cares" (.11111111). Therefore, all traffic that begins with 10.1.1. matches since the last octet is "don't care". Therefore, with this mask, network addresses 10.1.1.1 through 10.1.1.255 (10.1.1.x) are processed.

Subtract the normal mask from 255.255.255.255 in order to determine the ACL inverse mask. In this example, the inverse mask is determined for network address 172.16.1.0 with a normal mask of 255.255.255.0.

- 255.255.255.255 - 255.255.255.0 (normal mask) = 0.0.0.255 (inverse mask)

Note these ACL equivalents.

- The source/source-wildcard of 0.0.0.0/255.255.255.255 means "any".
- The source/wildcard of 10.1.1.2/0.0.0.0 is the same as "host 10.1.1.2".

### ACL Summarization

**Note:** Subnet masks can also be represented as a fixed length notation. For example, 192.168.10.0/24 represents 192.168.10.0 255.255.255.0.

This list describes how to summarize a range of networks into a single network for ACL optimization. Consider these networks.

```
192.168.32.0/24
192.168.33.0/24
192.168.34.0/24
```

```
192.168.35.0/24
192.168.36.0/24
192.168.37.0/24
192.168.38.0/24
192.168.39.0/24
```

The first two octets and the last octet are the same for each network. This table is an explanation of how to summarize these into a single network.

The third octet for the previous networks can be written as seen in this table, according to the octet bit position and address value for each bit.

| **Decimal** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 32 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 33 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 34 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 35 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 36 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 37 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 38 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 39 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| | M | M | M | M | M | D | D | D |

Since the first five bits match, the previous eight networks can be summarized into one network (192.168.32.0/21 or 192.168.32.0 255.255.248.0). All eight possible combinations of the three low-order bits are relevant for the network ranges in question. This command defines an ACL that permits this network. If you subtract 255.255.248.0 (normal mask) from 255.255.255.255, it yields 0.0.7.255.

```
access-list acl_permit permit ip 192.168.32.0 0.0.7.255
```

Consider this set of networks for further explanation.

```
192.168.146.0/24
192.168.147.0/24
192.168.148.0/24
```

192.168.149.0/24

The first two octets and the last octet are the same for each network. This table is an explanation of how to summarize these.

The third octet for the previous networks can be written as seen in this table, according to the octet bit position and address value for each bit.

| **Decimal** | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 146 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 147 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 148 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 149 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| | M | M | M | M | M | ? | ? | ? |

Unlike the previous example, you cannot summarize these networks into a single network. If they are summarized to a single network, they become 192.168.144.0/21 because there are five bits similar in the third octet. This summarized network 192.168.144.0/21 covers a range of networks from 192.168.144.0 to 192.168.151.0. Among these, 192.168.144.0, 192.168.145.0, 192.168.150.0, and 192.168.151.0 networks are not in the given list of four networks. In order to cover the specific networks in question, you need a minimum of two summarized networks. The given four networks can be summarized into these two networks:

- For networks 192.168.146.x and 192.168.147.x, all bits match except for the last one, which is a "don't care." This can be written as 192.168.146.0/23 (or 192.168.146.0 255.255.254.0).
- For networks 192.168.148.x and 192.168.149.x, all bits match except for the last one, which is a "don't care." This can be written as 192.168.148.0/23 (or 192.168.148.0 255.255.254.0).

This output defines a summarized ACL for the above networks.

```
!--- This command is used to allow access access for devices with IP
!--- addresses in the range from 192.168.146.0 to 192.168.147.254.

access-list 10 permit 192.168.146.0 0.0.1.255


!--- This command is used to allow access access for devices with IP
!--- addresses in the range from 192.168.148.0 to 192.168.149.254
```

```
access-list 10 permit 192.168.148.0 0.0.1.255
```

### Process ACLs

Traffic that comes into the router is compared to ACL entries based on the order that the entries occur in the router. New statements are added to the end of the list. The router continues to look until it has a match. If no matches are found when the router reaches the end of the list, the traffic is denied. For this reason, you should have the frequently hit entries at the top of the list. There is an implied deny for traffic that is not permitted. A single-entry ACL with only one deny entry has the effect of denying all traffic. You must have at least one permit statement in an ACL or all traffic is blocked. These two ACLs (101 and 102) have the same effect.

```
!--- This command is used to permit IP traffic from 10.1.1.0
!--- network to 172.16.1.0 network. All packets with a source
!--- address not in this range will be rejected.

access-list 101 permit ip 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255

!--- This command is used to permit IP traffic from 10.1.1.0
!--- network to 172.16.1.0 network. All packets with a source
!--- address not in this range will be rejected.

access-list 102 permit ip 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255
access-list 102 deny ip any any
```

In this example, the last entry is sufficient. You do not need the first three entries because TCP includes Telnet, and IP includes TCP, User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP).

```
!--- This command is used to permit Telnet traffic
!--- from machine 10.1.1.2 to machine 172.16.1.1.

access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1 eq telnet

!--- This command is used to permit tcp traffic from
!--- 10.1.1.2 host machine to 172.16.1.1 host machine.

access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1

!--- This command is used to permit udp traffic from
!--- 10.1.1.2 host machine to 172.16.1.1 host machine.

access-list 101 permit udp host 10.1.1.2 host 172.16.1.1

!--- This command is used to permit ip traffic from
!--- 10.1.1.0 network to 172.16.1.10 network.

access-list 101 permit ip 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255
```

### Define Ports and Message Types

In addition to defining ACL source and destination, it is possible to define ports, ICMP message types, and other parameters. A good source of information for well-known ports is RFC 1700 . ICMP message types are explained in RFC 792 .

The router can display descriptive text on some of the well-known ports. Use a **?** for help.

```
access-list 102 permit tcp host 10.1.1.1 host 172.16.1.1 eq ?
  bgp           Border Gateway Protocol (179)
  chargen       Character generator (19)
  cmd           Remote commands (rcmd, 514)
```

During configuration, the router also converts numeric values to more user-friendly values. This is an example where you type the ICMP message type number and it causes the router to convert the number to a name.

```
access-list 102 permit icmp host 10.1.1.1 host 172.16.1.1 14
```

becomes

```
access-list 102 permit icmp host 10.1.1.1 host 172.16.1.1 timestamp-reply
```

### Apply ACLs

You can define ACLs without applying them. But, the ACLs have no effect until they are applied to the interface of the router. It is a good practice to apply the ACL on the interface closest to the source of the traffic. As shown in this example, when you try to block traffic from source to destination, you can apply an inbound ACL to E0 on router A instead of an outbound list to E1 on router C. An access-list has a **deny ip any any** implicitly at the end of any access-list. If traffic is related to a DHCP request and if it is not explicity permitted, the traffic is dropped because when you look at DHCP request in IP, the source address is s=0.0.0.0 (Ethernet1/0), d=255.255.255.255, len 604, rcvd 2 UDP src=68, dst=67. Note that the source IP address is 0.0.0.0 and destination address is 255.255.255.255. Source port is 68 and destination 67. Hence, you should permit this kind of traffic in your access-list else the traffic is dropped due to implicit deny at the end of the statement.

**Note:** For UDP traffic to pass through, UDP traffic must also be permited explicitly by the ACL.

### Define In, Out, Inbound, Outbound, Source, and Destination

The router uses the terms in, out, source, and destination as references. Traffic on the router can be compared to traffic on the highway. If you were a law enforcement officer in Pennsylvania and wanted to stop a truck going from Maryland to New York, the source of the truck is Maryland and the destination of the truck is New York. The roadblock could be applied at the Pennsylvania–New York border (out) or the Maryland–Pennsylvania border (in).

When you refer to a router, these terms have these meanings.

- **Out**—Traffic that has already been through the router and leaves the interface. The source is where it has been, on the other side of the router, and the destination is where it goes.
- **In**—Traffic that arrives on the interface and then goes through the router. The source is where it has been and the destination is where it goes, on the other side of the router.
- **Inbound** —If the access list is inbound, when the router receives a packet, the Cisco IOS software checks the criteria statements of the access list for a match. If the packet is permitted, the software continues to process the packet. If the packet is denied, the software discards the packet.
- **Outbound**—If the access list is outbound, after the software receives and routes a packet to the outbound interface, the software checks the criteria statements of the access list for a match. If the packet is permitted, the software transmits the packet. If the packet is denied, the software discards the packet.

The in ACL has a source on a segment of the interface to which it is applied and a destination off of any other interface. The out ACL has a source on a segment of any interface other than the interface to which it is applied and a destination off of the interface to which it is applied.

### Edit ACLs

When you edit an ACL, it requires special attention. For example, if you intend to delete a specific line from a numbered ACL that exists as shown here, the entire ACL is deleted.

```
!--- The access-list 101 denies icmp from any to any network
!--- but permits IP traffic from any to any network.

  router#configure terminal
  Enter configuration commands, one per line.  End with CNTL/Z.
  router(config)#access-list 101 deny icmp any any
  router(config)#access-list 101 permit ip any any
  router(config)#^Z

  router#show access-list
  Extended IP access list 101
      deny icmp any any
      permit ip any any
  router#
  *Mar  9 00:43:12.784: %SYS-5-CONFIG_I: Configured from console by console
```

```
router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
router(config)#no access-list 101 deny icmp any any
router(config)#^Z

router#show access-list
router#
*Mar  9 00:43:29.832: %SYS-5-CONFIG_I: Configured from console by console
```

Copy the configuration of the router to a TFTP server or a text editor such as Notepad in order to edit numbered ACLs. Then make any changes and copy the configuration back to the router.

You can also do this.

```
router#configure terminal
  Enter configuration commands, one per line.
  router(config)#ip access-list extended test

!--- Permits IP traffic from 2.2.2.2 host machine to 3.3.3.3 host machine.

  router(config-ext-nacl)#permit ip host 2.2.2.2 host 3.3.3.3

!--- Permits www traffic from 1.1.1.1 host machine to 5.5.5.5 host machine.

  router(config-ext-nacl)#permit tcp host 1.1.1.1 host 5.5.5.5 eq www

!--- Permits icmp traffic from any to any network.

  router(config-ext-nacl)#permit icmp any any

!--- Permits dns traffic from 6.6.6.6 host machine to 10.10.10.0 network.

  router(config-ext-nacl)#permit udp host 6.6.6.6 10.10.10.0 0.0.0.255 eq
domain
  router(config-ext-nacl)#^Z
  1d00h: %SYS-5-CONFIG_I: Configured from console by consoles-l

  router#show access-list
  Extended IP access list test
      permit ip host 2.2.2.2 host 3.3.3.3
      permit tcp host 1.1.1.1 host 5.5.5.5 eq www
      permit icmp any any
      permit udp host 6.6.6.6 10.10.10.0 0.0.0.255 eq domain
```

Any deletions are removed from the ACL and any additions are made to the end of the ACL.

```
router#configure terminal
   Enter configuration commands, one per line.  End with CNTL/Z.
   router(config)#ip access-list extended test

!--- ACL entry deleted.

   router(config-ext-nacl)#no permit icmp any any
```

```
!--- ACL entry added.

   router(config-ext-nacl)#permit gre host 4.4.4.4 host 8.8.8.8
   router(config-ext-nacl)#^Z
   1d00h: %SYS-5-CONFIG_I: Configured from console by consoles-l

   router#show access-list
   Extended IP access list test
       permit ip host 2.2.2.2 host 3.3.3.3
       permit tcp host 1.1.1.1 host 5.5.5.5 eq www
       permit udp host 6.6.6.6 10.10.10.0 0.0.0.255 eq domain
       permit gre host 4.4.4.4 host 8.8.8.8
```

You can also add ACL lines to numbered standard or numbered extended ACLs by sequence number in Cisco IOS. This is a sample of the configuration:

Configure the extended ACL in this way:

```
Router(config)#access-list 101 permit tcp any any
Router(config)#access-list 101 permit udp any any
Router(config)#access-list 101 permit icmp any any
Router(config)#exit
Router#
```

Issue the **show access-list** command in order to view the ACL entries. The sequence numbers such as 10, 20, and 30 also appear here.

```
Router#show access-list
Extended IP access list 101
    10 permit tcp any any
    20 permit udp any any
    30 permit icmp any any
```

Add the entry for the access list 101 with the sequence number 5.

**Example 1:**

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip access-list extended 101
Router(config-ext-nacl)#5 deny tcp any any eq telnet
Router(config-ext-nacl)#exit
Router(config)#exit
Router#
```

In the **show access-list** command output, the sequence number 5 ACL is added as the first entry to the access-list 101.

```
Router#show access-list
Extended IP access list 101
    5 deny tcp any any eq telnet
    10 permit tcp any any
```

```
    20 permit udp any any
    30 permit icmp any any
Router#
```

**Example 2:**

```
internetrouter#show access-lists
Extended IP access list 101
    10 permit tcp any any
    15 permit tcp any host 172.162.2.9
    20 permit udp host 172.16.1.21 any
    30 permit udp host 172.16.1.22 any

internetrouter#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
internetrouter(config)#ip access-list extended 101
internetrouter(config-ext-nacl)#18 per tcp any host 172.162.2.11
internetrouter(config-ext-nacl)#^Z

internetrouter#show access-lists
Extended IP access list 101
    10 permit tcp any any
    15 permit tcp any host 172.162.2.9
    18 permit tcp any host 172.162.2.11
    20 permit udp host 172.16.1.21 any
    30 permit udp host 172.16.1.22 any
internetrouter#
```

Similarly, you can configure the standard access list in this way:

```
internetrouter(config)#access-list 2 permit 172.16.1.2
internetrouter(config)#access-list 2 permit 172.16.1.10
internetrouter(config)#access-list 2 permit 172.16.1.11

internetrouter#show access-lists
Standard IP access list 2
    30 permit 172.16.1.11
    20 permit 172.16.1.10
    10 permit 172.16.1.2

internetrouter(config)#ip access-list standard 2
internetrouter(config-std-nacl)#25 per 172.16.1.7
internetrouter(config-std-nacl)#15 per 172.16.1.16

internetrouter#show access-lists
Standard IP access list 2
    15 permit 172.16.1.16
    30 permit 172.16.1.11
    20 permit 172.16.1.10
    25 permit 172.16.1.7
    10 permit 172.16.1.2
```

The major difference in a standard access list is that the Cisco IOS adds an entry by descending order of the IP address, not on a sequence number.

This example shows the different entries, for example, how to permit an IP address (192.168.100.0) or the networks (10.10.10.0).

```
internetrouter#show access-lists
Standard IP access list 19
    10 permit 192.168.100.0
    15 permit 10.10.10.0, wildcard bits 0.0.0.255
    19 permit 201.101.110.0, wildcard bits 0.0.0.255
    25 deny any
```

Add the entry in access list 2 in order to permit the IP Address 172.22.1.1:

```
internetrouter(config)#ip access-list standard 2
internetrouter(config-std-nacl)#18 permit 172.22.1.1
```

This entry is added in the top of the list in order to give priority to the specific IP address rather than network.

```
internetrouter#show access-lists
Standard IP access list 19
    10 permit 192.168.100.0
    18 permit 172.22.1.1
    15 permit 10.10.10.0, wildcard bits 0.0.0.255
    19 permit 201.101.110.0, wildcard bits 0.0.0.255
    25 deny   any
```

**Note:** The previous ACLs are not supported in Security Appliance such as the ASA/PIX Firewall.

**Guidelines to change access-lists when they are applied to crypto maps**

- If you add to an existing access-list configuration, there is no need to remove the crypto map. If you add to them directly without the removal of the crypto map, then that is supported and acceptable.
- If you need to modify or delete access-list entry from an existing access-lists, then you must remove the crypto map from the interface. After you remove crypto map, make all changes to the access-list and re-add the crypto map. If you make changes such as the deletion of the access-list without the removal of the crypto map, this is not supported and can result in unpredictable behavior.

## Troubleshoot

### *How do I remove an ACL from an interface?*

Go into configuration mode and enter **no** in front of the **access-group** command, as shown in this example, in order to remove an ACL from an interface.

```
interface <interface>
no ip access-group <acl-number> in|out
```

### *What do I do when too much traffic is denied?*

If too much traffic is denied, study the logic of your list or try to define and apply an additional broader list. The **show ip access-lists** command provides a packet count that shows which ACL entry is hit.

The **log** keyword at the end of the individual ACL entries shows the ACL number and whether the packet was permitted or denied, in addition to port-specific information.

**Note:** The **log-input** keyword exists in Cisco IOS Software Release 11.2 and later, and in certain Cisco IOS Software Release 11.1 based software created specifically for the service provider market. Older software does not support this keyword. Use of this keyword includes the input interface and source MAC address where applicable.

### *How do I debug at the packet level that uses a Cisco router?*

This procedure explains the debug process. Before you begin, be certain that there are no currently applied ACLs, that there is an ACL, and that fast switching is not disabled.

**Note:** Use extreme caution when you debug a system with heavy traffic. Use an ACL in order to debug specific traffic. But, be sure of the process and the traffic flow.

1. Use the **access-list** command in order to capture the desired data.

   In this example, the data capture is set for the destination address of 10.2.6.6 or the source address of 10.2.6.6.

   ```
   access-list 101 permit ip any host 10.2.6.6
   access-list 101 permit ip host 10.2.6.6 any
   ```

2. Disable fast switching on the interfaces involved. You only see the first packet if fast switching is not disabled.

```
3. config interface
4. no ip route-cache
```

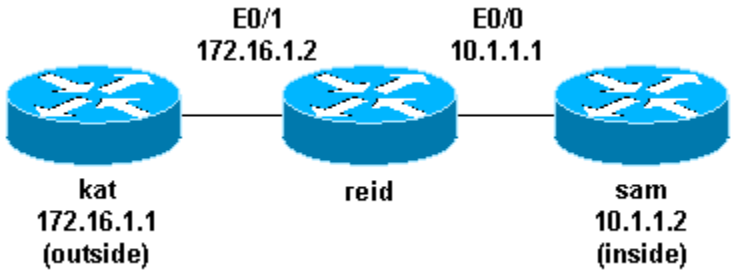
5. Use the **terminal monitor** command in enable mode in order to display **debug** command output and system error messages for the current terminal and session.
6. Use the **debug ip packet 101** or **debug ip packet 101 detail** command in order to begin the debug process.
7. Execute the **no debug all** command in enable mode and the **interface configuration** command in order to stop the debug process.
8. Restart caching.

```
9. config interface
10.  ip route-cache
```

# Types of IP ACLs

This section of the document describes ACL types.

## Network Diagram




## Standard ACLs

Standard ACLs are the oldest type of ACL. They date back to as early as Cisco IOS Software Release 8.3. Standard ACLs control traffic by the comparison of the source address of the IP packets to the addresses configured in the ACL.

This is the command syntax format of a standard ACL.

```
access-list access-list-number {permit|deny}
{host|source source-wildcard|any}
```

In all software releases, the *access-list-number* can be anything from 1 to 99. In Cisco IOS Software Release 12.0.1, standard ACLs begin to use additional numbers (1300 to 1999). These additional numbers are referred to as expanded IP ACLs. Cisco IOS Software Release 11.2 added the ability to use list *name* in standard ACLs.

A *source/source-wildcard* setting of 0.0.0.0/255.255.255.255 can be specified as **any**. The wildcard can be omitted if it is all zeros. Therefore, host 10.1.1.2 0.0.0.0 is the same as host 10.1.1.2.

After the ACL is defined, it must be applied to the interface (inbound or outbound). In early software releases, out was the default when a keyword out or in was not specified. The direction must be specified in later software releases.

```
interface <interface>
ip access-group number {in|out}
```

This is an example of the use of a standard ACL in order to block all traffic except that from source 10.1.1.x.

```
interface Ethernet0/0
ip address 10.1.1.1 255.255.255.0
ip access-group 1 in
access-list 1 permit 10.1.1.0 0.0.0.255
```

## Extended ACLs

Extended ACLs were introduced in Cisco IOS Software Release 8.3. Extended ACLs control traffic by the comparison of the source and destination addresses of the IP packets to the addresses configured in the ACL.

This is the command syntax format of extended ACLs. Lines are wrapped here for spacing considerations.

### *IP*

```
access-list access-list-number
     [dynamic dynamic-name [timeout minutes]]
     {deny|permit} protocol source source-wildcard
     destination destination-wildcard [precedence precedence]
     [tos tos] [log|log-input] [time-range time-range-name]
```

### *ICMP*

```
access-list access-list-number
     [dynamic dynamic-name [timeout minutes]]
     {deny|permit} icmp source source-wildcard
     destination destination-wildcard
     [icmp-type [icmp-code] |icmp-message]
     [precedence precedence] [tos tos] [log|log-input]
     [time-range time-range-name]
```

### *TCP*

```
access-list access-list-number
     [dynamic dynamic-name [timeout minutes]]
     {deny|permit} tcp source source-wildcard [operator [port]]
     destination destination-wildcard [operator [port]]
     [established] [precedence precedence] [tos tos]
     [log|log-input] [time-range time-range-name]
```

### *UDP*

```
access-list access-list-number
     [dynamic dynamic-name [timeout minutes]]
     {deny|permit} udp source source-wildcard [operator [port]]
     destination destination-wildcard [operator [port]]
     [precedence precedence] [tos tos] [log|log-input]
     [time-range time-range-name]
```

In all software releases, the *access-list-number* can be 100 to 199. In Cisco IOS Software Release 12.0.1, extended ACLs begin to use additional numbers (2000 to 2699). These additional numbers are referred to as expanded IP ACLs. Cisco IOS Software Release 11.2 added the ability to use list *name* in extended ACLs.

The value of 0.0.0.0/255.255.255.255 can be specified as **any**. After the ACL is defined, it must be applied to the interface (inbound or outbound). In early software releases, out was the default when a keyword out or in was not specified. The direction must be specified in later software releases.

```
interface <interface>
ip access-group {number|name} {in|out}
```

This extended ACL is used to permit traffic on the 10.1.1.x network (inside) and to receive ping responses from the outside while it prevents unsolicited pings from people outside, permitting all other traffic.

```
interface Ethernet0/1
ip address 172.16.1.2 255.255.255.0
ip access-group 101 in
access-list 101 deny icmp any 10.1.1.0 0.0.0.255 echo
access-list 101 permit ip any 10.1.1.0 0.0.0.255
```

**Note:** Some applications such as network management require pings for a keepalive function. If this is the case, you might wish to limit blocking inbound pings or be more granular in permitted/denied IPs.

### Lock and Key (Dynamic ACLs)

Lock and key, also known as dynamic ACLs, was introduced in Cisco IOS Software Release 11.1. This feature is dependent on Telnet, authentication (local or remote), and extended ACLs.

Lock and key configuration starts with the application of an extended ACL to block traffic through the router. Users that want to traverse the router are blocked by the extended ACL until they Telnet to the router and are authenticated. The Telnet connection then drops and a single-entry dynamic ACL is added to the extended ACL that exists. This permits traffic for a particular time period; idle and absolute timeouts are possible.

This is the command syntax format for lock and key configuration with local authentication.

```
username user-name password password
interface <interface>
ip access-group {number|name} {in|out}
```

The single-entry ACL in this command is dynamically added to the ACL that exists after authentication.

```
access-list access-list-number dynamic name {permit|deny} [protocol]
{source source-wildcard|any} {destination destination-wildcard|any}
[precedence precedence][tos tos][established] [log|log-input]
[operator destination-port|destination port]

line vty line_range


login local
```

This is a basic example of lock and key.

```
username test password 0 test
```

```
!--- Ten (minutes) is the idle timeout.

username test autocommand access-enable host timeout 10


interface Ethernet0/0
  ip address 10.1.1.1 255.255.255.0
  ip access-group 101 in

access-list 101 permit tcp any host 10.1.1.1 eq telnet

!--- 15 (minutes) is the absolute timeout.

access-list 101 dynamic testlist timeout 15 permit ip 10.1.1.0 0.0.0.255
172.16.1.0 0.0.0.255


line vty 0 4
login local
```

After the user at 10.1.1.2 makes a Telnet connection to 10.1.1.1, the dynamic ACL is applied. The connection is then dropped, and the user can go to the 172.16.1.x network.

### IP Named ACLs

IP named ACLs were introduced in Cisco IOS Software Release 11.2. This allows standard and extended ACLs to be given names instead of numbers.

This is the command syntax format for IP named ACLs.

```
ip access-list {extended|standard} name
```

This is a TCP example:

```
{permit|deny} tcp source source-wildcard [operator [port]]
destination destination-wildcard [operator [port]] [established]
[precedence precedence] [tos tos] [log] [time-range time-range-name]
```

This is an example of the use of a named ACL in order to block all traffic except the Telnet connection from host 10.1.1.2 to host 172.16.1.1.

```
interface Ethernet0/0
ip address 10.1.1.1 255.255.255.0
ip access-group in_to_out in

ip access-list extended in_to_out
permit tcp host 10.1.1.2 host 172.16.1.1 eq telnet
```

## Reflexive ACLs

Reflexive ACLs were introduced in Cisco IOS Software Release 11.3. Reflexive ACLs allow IP packets to be filtered based on upper-layer session information. They are generally used to allow outbound traffic and to limit inbound traffic in response to sessions that originate inside the router.

Reflexive ACLs can be defined only with extended named IP ACLs. They cannot be defined with numbered or standard named IP ACLs, or with other protocol ACLs. Reflexive ACLs can be used in conjunction with other standard and static extended ACLs.

This is the syntax for various reflexive ACL commands.

```
interface
ip access-group {number|name} {in|out}

ip access-list extended name
permit protocol any any reflect name [timeoutseconds]
ip access-list extended name

evaluate name
```

This is an example of the permit of ICMP outbound and inbound traffic, while only permitting TCP traffic that has initiated from inside, other traffic is denied.

```
ip reflexive-list timeout 120

interface Ethernet0/1
 ip address 172.16.1.2 255.255.255.0
 ip access-group inboundfilters in
 ip access-group outboundfilters out

ip access-list extended inboundfilters
permit icmp 172.16.1.0 0.0.0.255 10.1.1.0 0.0.0.255
evaluate tcptraffic


!--- This ties the reflexive ACL part of the outboundfilters ACL,
!--- called tcptraffic, to the inboundfilters ACL.

ip access-list extended outboundfilters
permit icmp 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255
permit tcp 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255 reflect tcptraffic
```

## Time-Based ACLs Using Time Ranges

Time-based ACLs were introduced in Cisco IOS Software Release 12.0.1.T. While similar to extended ACLs in function, they allow for access control based on time. A time range is created that defines specific times of the day and week in order to implement time-based ACLs. The time range is identified by a name and then referenced by a function. Therefore, the time

restrictions are imposed on the function itself. The time range relies on the router system clock. The router clock can be used, but the feature works best with Network Time Protocol (NTP) synchronization.

These are time-based ACL commands.

```
!--- Defines a named time range.

time-range time-range-name

!--- Defines the periodic times.

periodic days-of-the-week hh:mm to [days-of-the-week] hh:mm


!--- Or, defines the absolute times.

absolute [start time date] [end time date]

!--- The time range used in the actual ACL.

ip access-list name|number <extended_definition>time-rangename_of_time-range
```

In this example, a Telnet connection is permitted from the inside to outside network on Monday, Wednesday, and Friday during business hours:

```
interface Ethernet0/0
ip address 10.1.1.1 255.255.255.0
ip access-group 101 in

access-list 101 permit tcp 10.1.1.0 0.0.0.255 172.16.1.0 0.0.0.255
eq telnet time-range EVERYOTHERDAY

time-range EVERYOTHERDAY
periodic Monday Wednesday Friday 8:00 to 17:00
```

### Commented IP ACL Entries

Commented IP ACL entries were introduced in Cisco IOS Software Release 12.0.2.T. Comments make ACLs easier to understand and can be used for standard or extended IP ACLs.

This is the commented name IP ACL command syntax.

```
ip access-list {standard|extended} access-list-name
remark remark
```

This is the commented numbered IP ACL command syntax.

```
access-list access-list-number remark remark
```

This is an example of commenting a numbered ACL.

```
interface Ethernet0/0
ip address 10.1.1.1 255.255.255.0
ip access-group 101 in

access-list 101 remark permit_telnet
access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1 eq telnet
```

### Context-Based Access Control

Context-based access control (CBAC) was introduced in Cisco IOS Software Release 12.0.5.T and requires the Cisco IOS Firewall feature set. CBAC inspects traffic that travels through the firewall in order to discover and manage state information for TCP and UDP sessions. This state information is used in order to create temporary openings in the access lists of the firewall. Configure **ip inspect** lists in the direction of the flow of traffic initiation in order to allow return traffic and additional data connections for permissible session, sessions that originated from within the protected internal network, in order to do this.

This is the syntax for CBAC.

```
ip inspect name inspection-name protocol [timeoutseconds]
```

This is an example of the use of CBAC in order to inspect outbound traffic. Extended ACL 111 normally block the return traffic other than ICMP without CBAC opening holes for the return traffic.

```
ip inspect name myfw ftp timeout 3600
ip inspect name myfw http timeout 3600
ip inspect name myfw tcp timeout 3600
ip inspect name myfw udp timeout 3600
ip inspect name myfw tftp timeout 3600
interface Ethernet0/1
       ip address 172.16.1.2 255.255.255.0
       ip access-group 111 in
       ip inspect myfw out
access-list 111 deny icmp any 10.1.1.0 0.0.0.255 echo
access-list 111 permit icmp any 10.1.1.0 0.0.0.255
```

### Authentication Proxy

Authentication proxy was introduced in Cisco IOS Software Release 12.0.5.T. This requires that you have the Cisco IOS Firewall feature set. Authentication proxy is used to authenticate inbound or outbound users, or both. Users who are normally blocked by an ACL can bring up a browser to go through the firewall and authenticate on a TACACS+ or RADIUS server. The server passes additional ACL entries down to the router in order to allow the users through after authentication.

Authentication proxy is similar to lock and key (dynamic ACLs). These are the differences:

- Lock and key is turned on by a Telnet connection to the router. Authentication proxy is turned on by HTTP through the router.
- Authentication proxy must use an external server.
- Authentication proxy can handle the addition of multiple dynamic lists. Lock and key can only add one.
- Authentication proxy has an absolute timeout but no idle timeout. Lock and key has both.

Refer to the Cisco Secure Integrated Software Configuration Cookbook for examples of authentication proxy.

### Turbo ACLs

Turbo ACLs were introduced in Cisco IOS Software Release 12.1.5.T and are found only on the 7200, 7500, and other high-end platforms. The turbo ACL feature is designed in order to process ACLs more efficiently in order to improve router performance.

Use the **access-list compiled** command for turbo ACLs. This is an example of a compiled ACL.

```
access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1 eq telnet
access-list 101 permit tcp host 10.1.1.2 host 172.16.1.1 eq ftp
access-list 101 permit udp host 10.1.1.2 host 172.16.1.1 eq syslog
access-list 101 permit udp host 10.1.1.2 host 172.16.1.1 eq tftp
access-list 101 permit udp host 10.1.1.2 host 172.16.1.1 eq ntp
```

After the standard or extended ACL is defined, use the **global configuration** command in order to compile.

```
!--- Tells the router to compile.

access-list compiled

Interface Ethernet0/1
ip address 172.16.1.2 255.255.255.0

!--- Applies to the interface.

ip access-group 101 in
```

The **show access-list compiled** command shows statistics about the ACL.

### Distributed Time-Based ACLs

Distributed time-based ACLs were introduced in Cisco IOS Software Release 12.2.2.T in order to implement time-based ACLs on VPN-enabled 7500 series routers. Before the introduction of the distributed time-based ACL feature, time-based ACLs were not supported on line cards for the Cisco 7500 series routers. If time-based ACLs were configured, they behaved as normal ACLs. If an interface on a line card was configured with time-based ACLs, the packets switched

into the interface were not distributed switched through the line card but forwarded to the route processor in order to process.

The syntax for distributed time-based ACLs is the same as for time-based ACLs with the addition of the commands in regards to the status of the Inter Processor Communication (IPC) messages between the route processor and line card.

```
debug time-range ipc
show time-range ipc
clear time-range ipc
```

### Receive ACLs

Receive ACLs are used in order to increase security on Cisco 12000 routers by the protection of the gigabit route processor (GRP) of the router from unnecessary and potentially nefarious traffic. Receive ACLs were added as a special waiver to the maintenance throttle for Cisco IOS Software Release 12.0.21S2 and integrated into 12.0(22)S. Refer to GSR: Receive Access Control Lists for further information.

### Infrastructure Protection ACLs

Infrastructure ACLs are used in order to minimize the risk and effectiveness of direct infrastructure attack by the explicit permission of only authorized traffic to the infrastructure equipment while permitting all other transit traffic. Refer to Protecting Your Core: Infrastructure Protection Access Control Lists for further information.

### Transit ACLs

Transit ACLs are used in order to increase network security since they explicitly permit only required traffic into your network or networks. Refer to Transit Access Control Lists: Filtering at Your Edge for further information.