

The background of the cover is a complex, abstract fractal pattern. It consists of numerous overlapping, semi-transparent, glowing blue and purple lines and shapes that create a sense of depth and movement. The colors range from deep indigo to bright cyan, with some areas appearing more saturated than others. The overall effect is reminiscent of a digital or network structure, fitting the technical nature of the book's title.

The Clavister

cOS Core VPN cookbook

Only the tastiest recipes

By Peter Nilsson
Clavister AB Sweden

Copyright © 2017 by Peter Nilsson

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.

Printed by CreateSpace

First Printing, 2017

ISBN: 978-91-982968-4-6

Clavister AB
Sjögatan 6J
SE-89160 Örnsköldsvik
Sweden

www.clavister.com

cookbook@clavister.com

Acknowledgements

A big thank you for the encouragement and support of the following people in helping to get this book finished: Neal Sutherland, Torbjörn Fahlen, Fredrick Backman, André Sjöström, Kimmo Klemola, Tobias Rhén, Simon Bylund-Felixon and especially Magnus Strödin for being patient with the constant barrage of questions about all things IPsec.

About the Author

Peter Nilsson is the team leader for technical support at Clavister AB and has over a decade of experience helping enterprise customers set up network security solutions using Clavister products. He lives in Örnsköldsvik, Sweden



Table of Contents

Chapter 1: Introduction	7
1.1. An introduction to VPN – What is it?	9
1.2. Planning VPN	10
Chapter 2: IPsec	15
2.1. IPsec Explained	16
2.2. IPsec Tunnel Properties	24
Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario	43
Recipe 2.4. Configuring IPsec roaming / road warrior IPsec	69
Recipe 2.5. Configuring IPsec roaming split tunneling with multiple networks.	86
Recipe 2.6. Accessing satellite offices through central HQ using VPN	93
Recipe 2.7. Using Certificates as authentication on a Lan2Lan tunnel	110
Recipe 2.8. Using ID list to separate IPsec tunnels that are behind dynamic IP	132
Recipe 2.9. Making public internet access go through IPsec tunnel towards HQ	142
Chapter 3: L2TPv3	159
3.1. An introduction to L2TPv3	159
Recipe 3.2. Bridging a layer 2 network using L2TPv3 & IPsec	162
Recipe 3.3. Bridging a layer 2 VLAN network using L2TPv3 & IPsec	181
Appendix A - Troubleshooting rules and route problems using Ping Simulations.	196
Appendix B – IPsec IKE version 1 tunnel setup flowchart.	214
Afterword	216
Alphabetical Index	217

Chapter 1: Introduction

Welcome to the second Clavister book. In this book our main focus will be VPN (Virtual Private Network) in various forms, we will mainly focus on IPsec and IPsec scenarios but we will also cover some scenarios involving L2TPv3 as well as troubleshooting VPNs using ping simulation.

If not yet familiar with Clavister, it is recommended that the first book "The Clavister cOS Core Cookbook" be read first in order to get a fundamental grasp of Clavister and some of our general principles.

Network diagram icons and screenshots

Throughout the book we will be using network schematics of various scenarios. The various icons used is shown and described in figure 1.0.1.

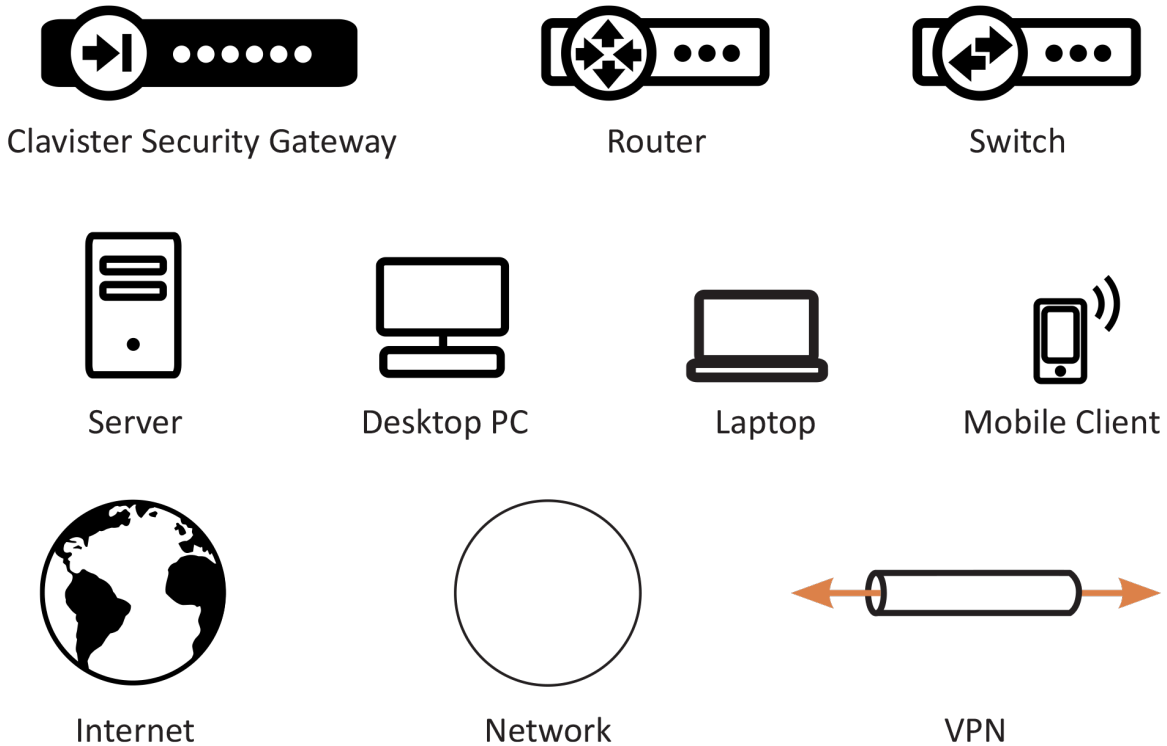


Figure 1.0.1 The various icons used throughout the book.

Screenshots

This book contains a large amount of screenshots primary taken from the Web User Interface (WebUI) of cOS Core. The screenshots are in most cases taken on a specific feature or function to highlight their use in the recipe context and may look different depending on what version of cOS Core that is used. Some pictures may be truncated slightly to make them fit better into the style and width of the book and also in an attempt to try limit the size of some screenshots to avoid them filling e.g. an entire page.

The book is based on the graphical style and feature set of cOS Core version 11.20.

1.1. An introduction to VPN – What is it?

The Internet is increasingly used as a means to connect computers since it offers efficient and inexpensive communication. The requirement therefore exists for data to traverse the Internet to its intended recipient without another party being able to read or alter the integrity of the message.

It is equally important that the recipient can verify that no one is falsifying data, in other words, pretending to be someone else. Virtual Private Networks (VPNs) meet this need, providing a highly cost effective means of establishing secure links between two co-operating computers so that data can be exchanged in a secure manner. VPN allows the setting up of a tunnel between two devices known as tunnel endpoints as shown in figure 1.1.1.



Figure 1.1.1 Connecting two devices together from different parts of the globe. Both sides have their own endpoint

All data flowing through a tunnel between the two endpoints is secured using encryption.

Encryption of VPN

Encryption of VPN traffic is done using the science of cryptography. Cryptography is an umbrella Expression, which covers 3 techniques and benefits:

1. **Confidentiality**

No one but the intended recipients is able to receive and understand the communication. Confidentiality is accomplished by encryption.

2. **Authentication and Integrity**

Proof for the recipient that the communication was actually sent by the expected sender, and that the data has not been modified in transit. This is accomplished by authentication, and is often implemented through the use of cryptographic keyed hashing.

3. **Non-repudiation**

Proof that the sender actually sent the data; the sender cannot later deny having sent it. Non-repudiation is usually a side-effect of authentication.

VPNs are normally only concerned with confidentiality and authentication. Non-repudiation is normally not handled at the network level but rather is usually done at a higher, transaction level.

1.2. Planning VPN

An attacker, targeting a VPN connection, will typically not attempt to crack the VPN encryption since this requires enormous effort. They will, instead, see VPN traffic as an indication that there is something worth targeting at the other end of the connection. Typically, mobile clients and branch offices are far more attractive targets than the main corporate network. Once an attacker gains access inside mobile clients or branch offices, getting to the corporate network then becomes easier. In designing a VPN there are many issues that need to be addressed which aren't always obvious. These include:

- Protecting mobile and home computers.
- Restricting access through the VPN to needed services only, since mobile computers are vulnerable.
- Creating DMZs for services that need to be shared with other companies through VPNs.
- Adapting VPN access policies for different groups of users.
- Creating key distribution policies.

Endpoint Security

A common misconception is that VPN-connections are equivalent to the internal network from a security standpoint and that they can be connected directly to it with no further precautions. It is important to remember that although the VPN-connection itself may be secure, the total level of security is only as high as the security of the tunnel endpoints.

It is becoming increasingly common for users on the move to connect directly to their company's network via VPN from their laptops or tablets. However, the client equipment itself is often not protected. In other words, an intruder can gain access to the protected network through an unprotected laptop and already-opened VPN connections.

Placement in a DMZ

A VPN connection should never be regarded as an integral part of a protected network. The VPN gateway should instead be located in a special DMZ or outside a gateway dedicated to this task. By doing this, the administrator can restrict which services can be accessed via the VPN and ensure that these services are well protected against intruders.

In cases where the router, firewall or gateway features an integrated VPN feature, it is usually possible to dictate the types of communication permitted and cOS Core has this feature.

Key Distribution

Key distribution schemes are best planned in advance. An example of a key distribution would be the Pre-shared key or Certificates needed for an IPsec tunnel.

Issues that need to be addressed include:

- How will keys be distributed? Email is not a good solution. Phone conversations might be secure enough.
- How many different keys should be used? One key per user? One per group of users? One per LAN-to-LAN connection? One key for all users and one key for all LAN-to-LAN connections? It is probably better using more keys than are necessary today, since it will be easier to adjust access per user (group) in the future.

- Should the keys be changed? If they are changed, how often? In cases where keys are shared by multiple users, consider using overlapping schemes, so that the old keys work for a short period of time when new keys have been issued.
- What happens when an employee in possession of a key leaves the company? If several users are using the same key, it should be changed.
- In cases where the key is not directly programmed into a network unit, such as a VPN gateway, how should the key be stored? On a USB stick? As a pass phrase to memorize? On a smart card? If it is a physical token, how should it be handled?

This means that it is very important to make sure that no unauthorized access can be made to a Laptop, for example, that has the required encryption keys or Certificates installed.

Good Anti-Virus software and strong passwords to access the laptop in question, are pretty much a requirement in such scenarios.

An example how a third party could gain access to a secure server is shown in figure 1.2.1.

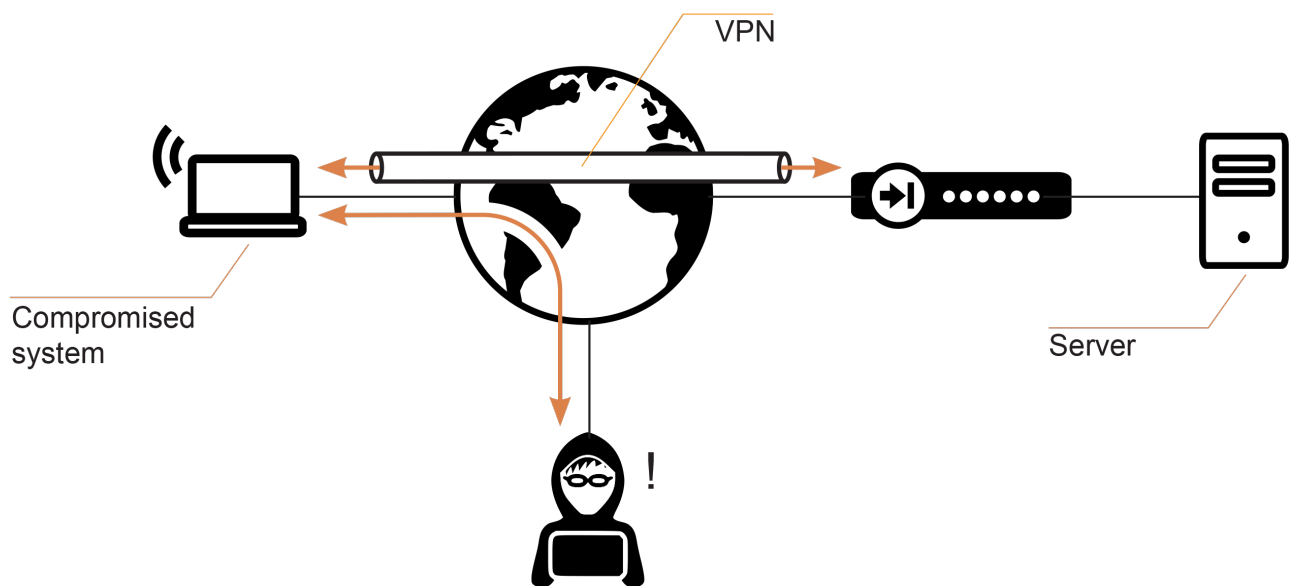


Figure 1.2.1 Very important to secure the system that establishes a VPN tunnel

If we take the scenario in figure 1.2.1 as an example we have a situation where we have an encrypted VPN connection between .e.g. a user's home PC and a Firewall located in the company the user works. The user uses the VPN tunnel in order to perform work from home.

The user has setup all the necessary VPN keys and encryption algorithms but if the home users PC get infected by a Trojan, an external third party could gain access to the company server by using the VPN tunnel already configured and established by the home user.

We will go through some of the common VPN tunnel scenarios as well as describe the various principles and how VPN works and how to apply it in various situations. But no matter how strong encryption and secure the VPN tunnel is, there may still be weak points in the network if a home users PC is lacking e.g. anti-virus software.

Chapter 2: IPsec

This section looks at the IPsec standards and describes in general terms the various components, techniques and algorithms that are used in IPsec based VPNs. To jump directly to the first example/recipe on how IPsec is configured and scenarios, please see *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*.

2.1. IPsec Explained

IPsec is, simply put, encryption at the network level. Let's say we have two PC's that want to communicate with each other over the internet. As an example, we want to send a text file from PC-A to PC-B. This text file contains sensitive information that we do not want to risk being intercepted by a third party or read by someone unauthorized. By transferring this file over the internet it must pass through a number of routers, switches etc. and at each step it risks interception by a third party. The solution is to encapsulate this text file inside an encrypted connection, shown as a tube in figure 2.1.1.

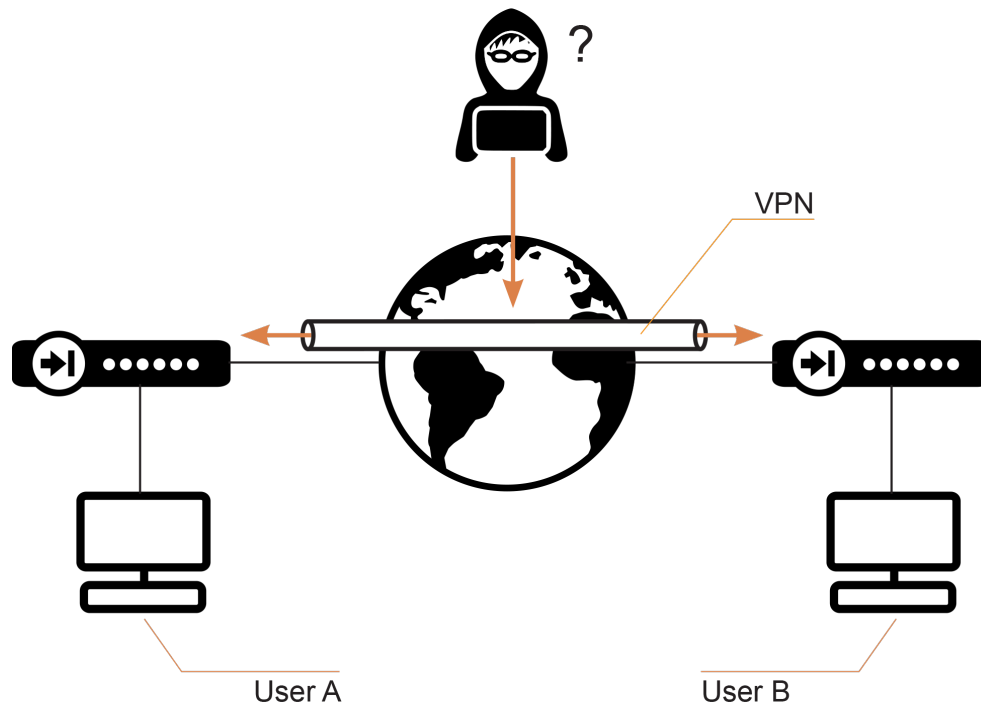


Figure 2.1.1 Encrypted tunnels can be used to safely transfer sensitive information between two locations without risk of exposure

The risk of exposing the content of our text file drops significantly by encapsulating the communication between the two PC's in an encrypted tunnel that uses strong encryption methods. Even if the traffic were to be intercepted by a third party, it is still rendered unreadable to them and therefore protected by strong encryption algorithms.

This defines the basic principle of IPsec, protecting network communication between networks. This also applies for other types of VPN such as PPTP, L2TP and SSL-VPN but the most commonly used method of creating encrypted tunnels is IPsec due to its very strong encryption capability.

Detailed information about IPsec its components

Internet Protocol Security (IPsec) is a set of protocols defined by the Internet Engineering Task Force (IETF) to provide IP security at the network layer. An IPsec based VPN is made up of two parts:

- Internet Key Exchange protocol (IKE)
- IPsec protocols (ESP)

The first part, IKE, is the initial negotiation phase, where the two VPN endpoints agree on which methods will be used to provide security for the underlying IP traffic. Furthermore, IKE is used to manage connections, by defining a set of Security Associations, SAs, for each connection. SAs are unidirectional, so there are usually at least two for each IPsec connection. The second part is the actual IP data being transferred, using the encryption and authentication methods agreed upon in the IKE negotiation. This can be accomplished by using IPsec protocol ESP (Encapsulating Security Payload).

The flow of events can be briefly described as follows:

- IKE negotiates how IKE should be protected
- IKE negotiates how IPsec should be protected
- IPsec moves data in the VPN

Internet Key Exchange (IKE)

Encrypting and authenticating data is fairly straightforward, the only things needed are encryption and authentication algorithms, and the keys used with them. IKE is used as a method of distributing these "session keys" between the two tunnel endpoints, as well as providing a way for the VPN endpoints to agree on how the data should be protected.

A session key is a pair of keys negotiated between the tunnel endpoints that will be used to encrypt data.

IKE has three main tasks:

- Provide a means for the endpoints to authenticate each other
- Establish new IPsec connections (create SA pairs)
- Manage existing connections

IKE is a protocol that belongs to the IPsec protocols suite. Its responsibility is in setting up security associations that allow two parties to send data securely. IKE was introduced in 1998 and was later superseded by version 2 roughly 7 years later. There are a number of differences between IKEv1 and IKEv2. Without going into too much detail, here is a small list of some of the enhancements in IKEv2 compared to IKEv1:

- IKEv2 less bandwidth usage
- IKEv2 supports EAP (Extensible Authentication Protocol) authentication
 - EAP is an authentication framework frequently used in wireless networks and point-to-point connections
- IKEv2 MOBIKE support
 - MOBILE (Mobility and Multihoming Protocol) support, provides a means for a roaming client to change IP address as it switches from one network to another
- IKEv2 built-in NAT traversal support
 - This applies for IKEv1 as well but not natively

- IKEv2 can by default detect whether a tunnel is still alive or not
 - This applies for IKEv1 as well but not natively

Security Associations (SAs)

IKE keeps track of connections by assigning a set of Security Associations, SAs, to each connection. An SA describes all parameters associated with a particular connection, such as the IPsec protocol used (ESP/AH/both) as well as the session keys used to encrypt/decrypt and/or authenticate/verify the transmitted data.

An SA is unidirectional and relates to traffic flow in one direction only. For the bidirectional traffic that is usually found in a VPN, there is therefore a need for more than one SA per connection. In most cases two SAs will be created for each connection, one for the incoming traffic, and the other the outgoing.



Note

This routing table will look different depending on the amount of interfaces and the hardware platform.

IKE Negotiation

The process of negotiating session parameters consists of a number of phases and modes. The flow of events can be summarized as follows:

IKE Phase-1

- Negotiate how IKE should be protected

IKE Phase-2

- Negotiate how IPsec should be protected
- Derive some fresh keying material from the key exchange in phase-1, to provide session keys to be used in the encryption and authentication of the VPN data flow

IKE and IPsec Lifetimes

Both the IKE and the IPsec connections have limited lifetimes. Both are described in terms of time (seconds), and data (kilobytes). These lifetimes prevent a connection from being used for too long, which is desirable from a crypto-analysis perspective. Meaning that if someone wants to decrypt the data after a re-key, they have to start all over again as new encryption keys have been generated.

The IPsec lifetime must be shorter than the IKE lifetime. The difference between the two must be a minimum of 5 minutes. This allows for the IPsec connection to be re-keyed simply by performing another phase-2 negotiation. There is no need to do another phase-1 negotiation until the IKE lifetime has expired.

Lifetime Recommendations

It is recommended that the lifetime values are equal to or greater than the following values:

- IKE lifetime - 600 seconds (10 minutes)
- IPsec lifetime - 300 seconds (5 minutes)

IPsec lifetime - 300 seconds (5 minutes)

If the administrator uses values which are less, cOS Core will accept them but a warning message will be presented during the reconfiguration.

Please keep in mind that this is the minimum value. It is not recommended to use such low values on active tunnels as it means that both IKE and IPsec SA's will re-key very often. And depending on key-length* used and the power of the hardware used it could cause traffic disturbances when the system needs to generate new encryption keys. A common lifetime to use for IKE is 28800 seconds (8 hours) and 3600 seconds for IPsec (1 hour).

*An example of key-length would be for instance using the AES algorithm with a 256 bit key size. AES can use 128, 192 and 256 bit keys. It varies depending on encryption algorithm.

IKE Algorithm Proposals

An IKE algorithm proposal list is a suggestion of how to protect IKE and IPsec data flows. The VPN device initiating an IPsec connection, usually called the initiator, will send a list of the algorithms combinations it supports for protecting the connection and it is then up to the device at the other end of the connection, usually called the responder, to say which proposal is acceptable.

The responding VPN device, upon receiving the list of supported algorithms, will choose the algorithm combination that best matches its own security policies, and reply by specifying which member of the list it has chosen. If no mutually acceptable proposal can be found, the responder will reply by saying that nothing on the list was acceptable, and possibly also provide a hint or clue as to what the problem could be.

This negotiation to find a mutually acceptable algorithm combination is done not just to find the best way to protect the IPsec connection but also to find the best way to protect the IKE negotiation itself. Algorithm proposal lists contain not just the acceptable algorithm combinations for encrypting and authenticating data but also other IKE related parameters. Further details of the IKE negotiation and the other IKE parameters are described next.

IKE Phase-1 - IKE Security Negotiation

An IKE negotiation is performed in two phases. The first phase, phase 1, is used to authenticate the two VPN gateways or VPN Clients to each other, by confirming that the remote device has a matching Pre-Shared Key (or provides a matching/valid Certificate).

However, since we do not want to publish too much of the negotiation in plaintext, we first agree upon a way of protecting the rest of the IKE negotiation. This is done, as described in the previous section, by the initiator sending a proposal-list to the responder.

When this has been done, and the responder accepted one of the proposals, the initiator tries to authenticate the other end of the VPN to make sure it is who we think it is, as well as proving to the remote device that we are who we claim to be.

A technique known as a Diffie Hellman Key Exchange is used to initially agree a shared secret between the two parties in the negotiation and to derive keys for encryption.

Authentication can be accomplished through Pre-Shared Keys (PSK), certificates or public key encryption. Pre-Shared Keys is the most common authentication method today. PSK and certificates are supported by the cOS Core VPN module.

IKE Phase-2 - IPsec Security Negotiation

In phase 2, another negotiation is performed, detailing the parameters for the IPsec connection. During phase 2 we will also repeatedly extract new keying material from the Diffie-Hellman key exchange in phase 1 in order to provide fresh session keys to use in protecting the VPN data flow.

Perfect Forward Secrecy (PFS)

If PFS is used, a new Diffie-Hellman exchange is performed for each phase 2 negotiation. While this is slower, it makes sure that no keys are dependent on any other previously used keys; no keys are extracted from the same initial keying material. This is to make sure that, in the unlikely event that some key was compromised, no subsequent keys can be derived.

In cryptography PFS is a property of key-agreement protocols ensuring that a session key derived from a set of long-term keys cannot be compromised if one of the long-term keys is compromised in the future. The key used to protect transmission of data must not be used to derive any additional keys, and if the key used to protect transmission of data is derived from some other keying material, then that material must not be used to derive any more keys.

In this way, compromise of a single key permits access only to data protected by that single key.

Once the phase 2 negotiation is finished, the VPN connection is established and ready for traffic to pass through it.

IKE and IPsec negotiation can operate in two modes called Main and Aggressive modes. A simplified picture on the Main Mode is shown in figure 2.1.2. We will discuss Main and Aggressive mode in more detail later in the book.

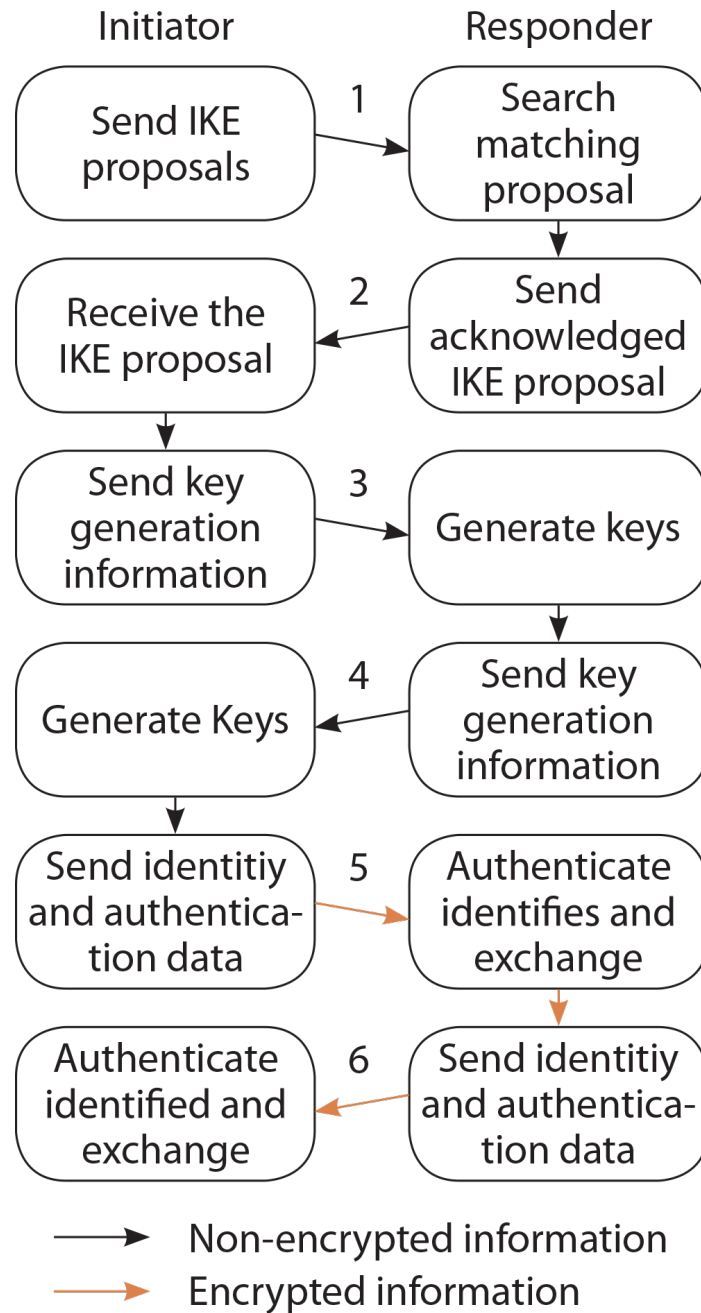


Figure 2.1.2 IKE and IPsec negotiation between initiator and responder (sometimes known as terminator) in Main Mode

2.2. IPsec Tunnel Properties

Below, is a summary of the key properties required of an IPsec tunnel object. Many of these options and settings will be explained in various contexts as we progress in the IPsec related recipes, this part is mainly to be used as reference as it can be a bit overwhelming if read as a whole.

Local and Remote Networks/Hosts

These are the subnets or hosts between which IP traffic will be protected by the VPN. In a LAN-to-LAN connection, these will be the network addresses of the respective LANs. If roaming clients are used, the remote network will most likely be set to all-nets, meaning that the roaming client may connect from anywhere.

The most common VPN is where we connect two sites together (usually called Lan2Lan) or when there is a client/server solution where a VPN client connects to a VPN server to access resources, the client/server solution is commonly called Roaming or Roadwarrior.

Local ID and Remote ID

The local and remote IDs are values sent by the corresponding sides of the tunnel during the IKE negotiation and both can be used with either a pre-shared key or certificate based tunnel.

Local ID

This property of an IPsec Tunnel object represents the identity of the local VPN tunnel endpoint and this is the value presented to the remote peer during the IKE negotiation. The property is set to only a single value but can be left blank when using certificates since the ID will be contained within the host certificate. If the certificate contains multiple IDs, this property can be set to specify which ID in the certificate to use.

The "Enforce Local ID"-property can be enabled so that when cOS Core is acting as responder, the local ID proposed by the initiator must match the Local ID value on the responder. The default behavior is to ignore the proposed ID.

Remote ID

This property can be used to specify an ID list object. An ID list object contains one or more IDs. When using certificates, the certificate sent by a remote peer must contain an ID which matches one of the IDs in the list in order for the peer to be authenticated.

cOS Core applies sanity checks on all remote IDs to ensure they are acceptable. Usually malformed IDs have a problem in the OU name. For example, a faulty remote ID name might be the following:

```
DN=Clavister, OU=One,Two,Three, DC=SE
```

If specified by the administrator, there will be an error message when the cOS Core configuration is committed. The corrected remote ID form is the following:

```
DN=Clavister, OU=One\,Two\,Three, DC=SE
```

We will go into more details about certificates later in this chapter.

Encapsulation Mode

IPsec can be configured using two modes:

Mode-1 - Tunnel Mode.

IPsec tunnel mode is the default mode. With tunnel mode, the entire original IP packet is protected by IPsec. This means IPsec takes the entire original packet, encrypts it, adds a new IP header and sends it to the other side of the IPsec tunnel (Remote Endpoint).

Tunnel mode is most commonly used between gateways (e.g. two Clavister firewalls or other IPsec capable gateways/routers).

In tunnel mode, an IPsec header (AH or ESP header) is inserted between the IP header and the upper layer protocol. ESP and ESP Auth trailer is the same for both Tunnel & Transport mode.



Note

ESP= Encapsulating Security Payload and AH=Authentication Header.

Between AH and ESP, ESP is most commonly used in IPsec VPN Tunnel configuration. **Figure 2.2.1 shows IPsec Tunnel mode with ESP header.**



Note

cOS Core only supports ESP headers, AH is not supported. When IPsec was originally designed, computer power was far less than it is today, particularly in network devices, the idea was to have two protocols that could be used independently or in conjunction in order to better manage the computer resources (and thus, performance) more effectively.

AH by itself was intended for cases where secrecy wasn't required but we wanted to ensure the authenticity of the sender. Imagine routing protocol updates such as DNS or SNMP where the data isn't secret but you want to be sure the sender is who they claim to be.

ESP by itself was intended for secrecy of the payload and has enough integrity features that it does not really need AH. Due to the high requirements of encryption, AH was never implemented as an option in cOS Core due to its lack of strong encryption capabilities. It is very rare to encounter IPsec capable equipment with only support for AH in today's network security market.

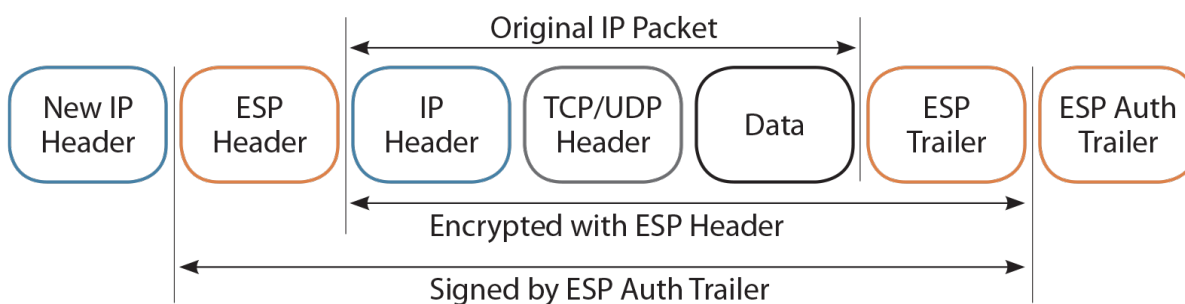


Figure 2.2.1 Tunnel mode ESP header

Mode-2 - Transport Mode

IPsec Transport mode is used for end-to-end communications, for example, for communication between a client and a server or between a workstation and a gateway (if the gateway is being treated as a host).

Transport mode provides the protection of data, also known as IP Payload, and consists of TCP/UDP header + Data, through an AH or ESP header. The payload is encapsulated by the IPsec headers and trailers. The original IP headers remain intact, except that the IP protocol field is changed to ESP (IP protocol 50) or AH (IP protocol 51), and the original protocol value is saved in the ESP trailer to be restored when the packet is decrypted.



Note

ESP trailer consists of the Padding, Pad Length, and Next Header fields.

IPsec transport mode is usually used when another tunneling protocol (like GRE) is used to first encapsulate the IP data packet, then IPsec is used to protect the GRE tunnel packets. IPsec protects the GRE tunnel traffic in transport mode. Figure 2.2.2 shows IPsec Transport mode with ESP header.

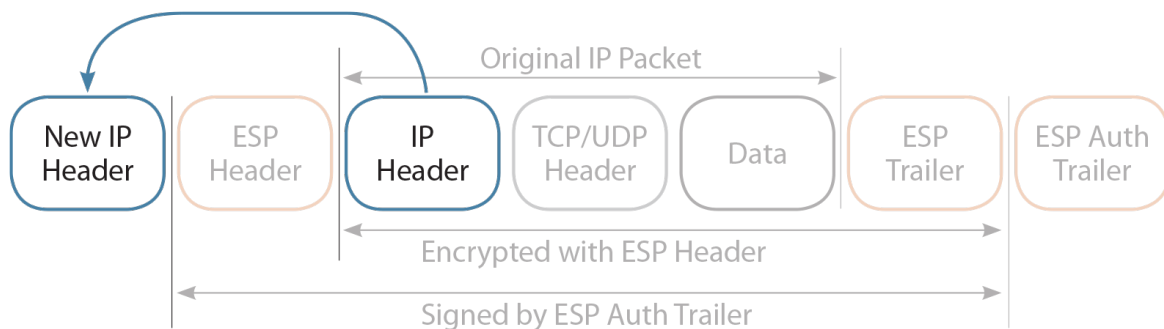


Figure 2.2.2 Transport mode ESP header

Notice that the original IP Header is moved to the front. Placing the sender's IP header at the front (with minor changes to the protocol ID) proves that transport mode does not provide protection or encryption to the original IP header.

ESP is identified in the New IP header with an IP protocol ID of 50.

Remote Endpoint

The remote endpoint (sometimes also referred to as the remote gateway) is the remote device on the other side of the tunnel that does the VPN decryption/authentication of the data that we send through the tunnel and that passes the unencrypted data on to its final destination. Basically the machine we would like to setup an encrypted tunnel against.



Figure 2.2.3 A tunnel endpoint means the point a tunnel is connected to. This is true for both sides, although varies slightly in case of a client/server solution

This field can also be set to "none", forcing the Clavister Security Gateway to treat the remote IP address as the remote endpoint. This is particularly useful in cases of roaming access, where the IP addresses of the remote VPN clients are not known beforehand. Setting this option to "none" will allow anyone coming from an IP address conforming to the "remote network" address discussed above to open a VPN connection, provided they can authenticate properly.

The remote endpoint can be specified as a host name such as `vpn.example.com`. If this is done, the prefix `dns:` must be used. The string above should therefore be specified as `dns:vpn.example.com`.

Using DNS can be advantageous in scenarios where the target remote endpoint is behind a dynamic IP.



Note

The remote endpoint option is not applicable in transport mode.

Main/Aggressive Mode

The IKE negotiation has two modes of operation, main mode and aggressive mode. The difference between these two is that aggressive mode consists of fewer packet exchanges. Aggressive mode does not give identity protection of the two IKE peers, unless digital certificates are used. This means VPN peers exchange their identities without encryption (i.e. in clear text). It is not as secure as main mode, but the advantage to aggressive mode is that it is faster than Main mode.

Main Mode creates an encrypted channel before exchanging the identities.

Aggressive Mode exchanges endpoint IDs in "clear text", while performing DH (Diffie Hellman) exchange and establishing the secure channel. Aggressive mode is less secure than Main Mode and should be less preferred.



Note

It is recommended NOT to use aggressive mode in roaming/roadwarrior scenarios for IKE version 1. The reason for this is the system becomes vulnerable to an IKE amplification attack. Using either tunnel mode or IKE version 2 is strongly recommended in roaming/roadwarrior scenarios.

Going into details about the differences between Main mode and Aggressive quickly becomes complicated. To put it bluntly, using Aggressive mode is less secure but goes faster to negotiate/establish the VPN tunnel. This is however a rare problem in today's network environment with very fast computers and internet speeds.

Main mode should be used whenever possible.

IPsec Protocols

The IPsec protocols describe how the payload data will be processed. The two protocols to choose from are AH, Authentication Header, and ESP, Encapsulating Security Payload. ESP provides encryption, authentication, or both. However, it is not recommended to use encryption only, since it will dramatically decrease security.

Note that AH only provides authentication. The difference from ESP with authentication only is that AH also authenticates parts of the outer IP header, for instance source and destination addresses, making certain that the packet really came from who the IP header claims it is from.



Note

cOS Core does not support AH.

IKE and IPsec Encryption

This specifies the encryption algorithms that can be used in the IKE and IPsec negotiations respectively, and depending on the algorithms, the size of the encryption keys used. The algorithms supported by cOS Core is as follows:

- AES
- Blowfish
- Twofish
- Cast128
- 3DES
- DES

DES is only included to be compatible with other older VPN implementations. The use of DES should be avoided whenever possible since it is considered insecure.

IKE and IPsec Authentication

This specifies the authentication algorithms that can be used in the IKE and IPsec negotiation phase. The algorithms supported by cOS Core is as follows:

- MD5
- SHA1
- SHA256
- SHA512
- AES-XCBC (IKEv2 only)

MD5 is considered insecure and should not be used, SHA1 is still OK to use at the time this is written but the recommendation is to use SHA-256, SHA-512 or AES-XCBC (if IKEv2).

IKE DH Group

This specifies the Diffie-Hellman group to use for the IKE exchange. cOS Core supports DH groups of various strengths ranging from group 1 with 768-bit up to group 18 that uses 8192 bit. Raising the group number from the default (group 2 – 1024 bit) should be done with caution as more computing resources will be used for higher group numbers and could lead to unacceptable tunnel setup times on slower hardware platforms.

IKE Lifetime

This refers to the lifetime of the IKE connection. It is specified in time (seconds) as well as data amount in kilobytes, specifying a lifetime in kilobytes is optional. Whenever one of these expires, a new phase-1 exchange will be performed. If no data was transmitted in the last "incarnation" of the IKE connection, no new connection will be made until someone wants to use the VPN connection again. This value must be set greater than the IPsec SA lifetime.

PFS

With Perfect Forwarding Secrecy (PFS) disabled, initial keying material is "created" during the key exchange in phase-1 of the IKE negotiation. In phase-2 of the IKE negotiation, encryption and authentication session keys will be extracted from this initial keying material. By using PFS, completely new keying material will always be created upon re-key. Should one key be compromised, no other key can be derived using that information.

PFS can be used in two modes: the first is PFS on keys, where a new key exchange will be performed in every phase-2 negotiation. The other type is PFS on identities, where the identities are also protected, by deleting the phase-1 SA every time a phase-2 negotiation has been finished, making sure no more than one phase-2 negotiation is encrypted using the same key.

IPsec Lifetime

This is the lifetime of the VPN connection. It can be specified in both time (seconds) and data amount (optional value in kilobytes). Whenever either of these values is exceeded, a re-key will be initiated, providing new IPsec encryption and authentication session keys. If the VPN connection has not been used during the last re-key period, the connection will be terminated, and re-opened from scratch when the connection is needed again (e.g. when a PC attempts to connect to a resource through the encrypted tunnel).

This value must be set lower than the IKE lifetime.

Diffie-Hellman Groups

Diffie-Hellman (DH) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to establish a shared secret key over an insecure communications channel through a series of plain text exchanges. Even though the exchanges between the parties might be monitored by a third party, the Diffie-Hellman technique makes it extremely difficult for the third party to determine what the agreed shared secret key is and decrypt data that is encrypted using that key.

Diffie-Hellman is used to establish the shared secret keys for IKE, IPsec and PFS in cOS Core. The higher the group number (and in turn key-length/strength), the greater the security, but this will also increase the hardware processing resources required.

The DH groups supported by cOS Core are as follows:

- DH group 1 (768-bit).
 - DH group 2 (1024-bit - the default setting).
 - DH group 5 (1536-bit).
 - DH group 14 (2048-bit).
 - DH group 15 (3072-bit).
 - DH group 16 (4096-bit).
 - DH group 17 (6144-bit).
 - DH group 18 (8192-bit).
-



Warning

Higher DH group numbers (above group 15) will consume a lot of CPU resources. Most hardware platforms will be able to provide sufficient processing resources for the default DH group 2. Higher group numbers will consume progressively more resources and this can mean that tunnel setup for the highest groups can be unacceptably long (tens of seconds) on a slower hardware platform. It could also result in 100% processor utilization from the tunnel setup process and a possible temporary halt of all traffic throughput.

IKE Authentication

cOS Core supports two methods of authentication:

- Pre Shared Key (PSK)
- Certificates

PSK

Using a pre-shared key (PSK) is a method where the endpoints of the VPN "share" a secret key. This is a service provided by IKE, and thus has all the advantages that come with it, making it far more flexible than manual keying.

PSK Advantages

The main advantage of PSK is that it is very easy to configure and setup. Compared to Certificates it can be configured in seconds and does not require any third party programs or solutions.

PSK Disadvantages

One thing that has to be considered when using pre-shared keys is key distribution. How are the Pre-Shared Keys distributed to remote VPN clients and gateways? This is a major issue, since the security of a PSK system is based on the PSKs being secret. Should one PSK be compromised, the configuration will need to be changed to use a new PSK. The pre-shared key can also be configured using a very simple passphrase (such as "test123"), useful for testing purposes but very insecure if used in a live environment.

Certificates

Each VPN gateway has its own certificate, and one or more trusted root certificates. The authentication is based on two things:

- That each endpoint has the private key corresponding to the public key found in its certificate, and that nobody else has access to the private key.
- That the certificate has been signed by someone that the remote endpoint trusts.

Advantages of Certificates

The main advantage of certificates is added flexibility. Many VPN clients, for instance, can be managed without having the same pre-shared key configured on all of them, which is often the case when using pre-shared keys with roaming clients. Instead, should a client be compromised, the client's certificate can simply be revoked. No need to reconfigure every client.

Disadvantages of Certificates

The main disadvantage of certificates is the added complexity. Certificate-based authentication may be used as part of a larger public key infrastructure, making all VPN clients and gateways dependent on third parties. There are many settings and options available to certificates which could make the initial configuration and setup cumbersome to get it working according to specifications.

Any problems with Certificate based tunnels usually require quite a lot of troubleshooting and documentation examination to get it to work properly. The main problem is usually not on cOS Core but rather the third party CA server (or similar) and the many options and settings.

IPsec tunnel protocol ESP (Encapsulating Security Payload)

In IPsec ESP provides origin authenticity, integrity and confidentiality protection of packets. ESP also supports encryption-only and authentication-only configurations, but using encryption without authentication is strongly discouraged. Unlike Authentication Header (AH), ESP in transport mode does not provide integrity and authentication for the entire IP packet. When using Tunnel Mode the entire original IP packet is encapsulated with a new packet header added. ESP protection is provided for the whole inner IP packet (including the inner header) while the outer header (including any outer IPv4 options or IPv6 extension headers) remains unprotected. ESP operates directly on top of IP, using IP protocol number 50.

A detailed overview of ESP when used in tunnel or transport mode is shown in figure 2.2.4

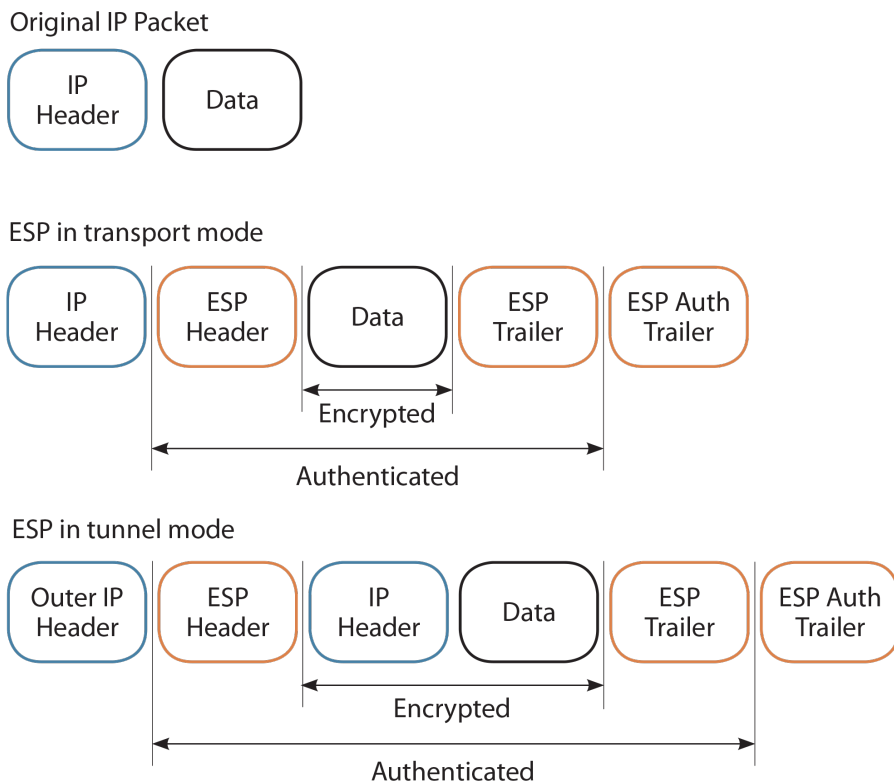


Figure 2.2.4 ESP tunnel/transport mode overview (also see figure 2.2.1 and 2.2.2 for details)

ESP only authenticates the data after the ESP header; thus the outer IP header is left unprotected. The ESP protocol is used for both encryption and authentication of the IP packet. It can also be used to do either encryption only, or authentication only.

NAT Traversal

Both IKE and IPsec protocols present a problem in the functioning of NAT (Network Address Translation). Both protocols were not designed to function with NAT and because of this, the technique called NAT traversal has evolved (also known as NAT-T). NAT traversal is an add-on to the IKE and IPsec protocols that allows them to function when being NATed. cOS Core supports the RFC 3947 standard for NAT-Traversal with IKE. An example of a NAT scenario is shown in figure 2.2.5.



Figure 2.2.5 NAT-T needs to be used if the connecting client is behind a NAT'ing device such as a router

NAT traversal is divided into two parts:

- Extensions to IKE that lets IPsec peers tell each other that they support NAT traversal, and the specific version(s) of NAT it supports.
- Changes to the ESP encapsulation. If NAT traversal is used, ESP is encapsulated in UDP, which allows for more flexible NATing.

Below, is a more detailed description of the changes made to the IKE and IPsec protocols. NAT traversal is only used if both ends have support for it. For this purpose, NAT traversal aware VPN devices send out a special "vendor ID" to tell the other end of the tunnel that it understands NAT traversal, and which specific versions of the draft it supports.

Achieving NAT Detection

To achieve NAT detection both IPsec peers send hashes of their own IP addresses along with the source UDP port used in the IKE negotiations. This information is used to see whether the IP address and source port that each peer uses is the same as what the other peer sees. If the source address and port have not changed, then the traffic has not been NATed along the way, and NAT traversal is not necessary. If the source address and/or port have changed, then the traffic has been NATed, and NAT traversal is used.

Changing Ports once NAT has been detected

Once the IPsec peers have decided that NAT traversal is necessary, the IKE negotiation is moved away from UDP port 500 to port 4500.



Note

IPsec negotiations always start using UDP port 500 and later in the negotiation move on to use the NAT-T port (4500) or IP protocol 50 (ESP).

Changing the port is necessary since certain NAT devices treat UDP packet on port 500 differently from other UDP packets in an effort to work around the NAT problems with IKE.

UDP Encapsulation

Another problem that NAT traversal resolves is that the ESP protocol is an IP protocol (IP protocol ID 50). There is no port information as we have in TCP and UDP, which makes it impossible to have more than one NATed client connected to the same remote gateway and at the same time. Because of this, ESP packets are encapsulated in UDP. ESP-UDP traffic is sent on port 4500, the same port as IKE when NAT traversal is used. Once the port has been changed, all following IKE communication is done over port 4500. NAT keep-alive packets are also sent periodically to keep the NAT mapping alive.



Note

Enforcing NAT-T can sometimes solve certain scenarios if IP protocol 50 is blocked by some equipment between the tunnel endpoints. By changing to UDP the problem can in some cases be bypassed.

NAT Traversal Configuration

Most NAT traversal functionality is completely automatic and in the initiating gateway no special configuration is needed. However, for responding gateways two points should be noted:

- On responding gateways, the Remote Endpoint field is used as a filter on the source IP of received IKE packets. This should be set to allow the NATed IP address of the initiator.
- When individual pre-shared keys are used with multiple tunnels connecting to the same remote gateway which are then NATed out through the same address, it is important to make sure the Local ID property of an IPsec Tunnel object is unique for the tunnel of every connecting client and takes one of the following values:
 - i. Auto - The local ID becomes the IP address of the outgoing interface. This is the recommended setting unless the two gateways have the same external IP address.
 - ii. IP - An IP address can be manually entered.
 - iii. DNS - A DNS address can be manually entered.
 - iv. Email - An email address can be manually entered.

Basically the connecting clients must have some sort of unique identifier that can separate the different clients. Otherwise the terminating gateway will be unable to properly match the connecting clients and it could result in traffic disruptions if the sessions start to be mixed up.

Algorithm Proposal Lists

To agree on the VPN connection parameters, a negotiation process is performed. As a result of the negotiations, the IKE and IPsec security associations (SAs) are established. A proposal list of supported algorithms is the starting point for the negotiation. Each entry in the list defines parameters for a supported algorithm that the VPN tunnel endpoint device is capable of supporting. The initial negotiation attempts to agree on a set of algorithms that the endpoint devices at either end of the tunnel can support.

There are two types of proposal lists, IKE proposal lists and IPsec proposal lists. IKE lists are used during IKE Phase-1 (IKE Security Negotiation), while IPsec lists are using during IKE Phase-2 (IPsec Security Negotiation). Several algorithm proposal lists are already defined by default in cOS Core for different VPN scenarios and a custom user defined list can also be created and used.

Pre-shared Keys

Pre-Shared Keys (commonly known as PSK) are used to authenticate VPN tunnels. The keys are secrets that are shared by the communicating parties before communication takes place. To communicate, both parties prove to each other that they know the secret. The security of a shared secret depends on how "good" a passphrase is. Passphrases that are common words can be vulnerable to dictionary attacks.

The PSK is not sent or exposed in the tunnel negotiation as that would void the whole purpose, but to provide an example of roughly how it works:

1. The initiator of the IPsec tunnel sends a packet with data to the remote endpoint.
 - 1.1. The initiator also makes a hash out of the data packet using its PSK.
2. The remote endpoint takes the data and makes a hash out it using its PSK and sends it back to the initiator.
3. The initiator takes the packet and compares the hashed data using its own hashed data packet. If the two packets are identical it means that the PSK is the same and the tunnel negotiation can proceed into the next phase.

Please note that this is an overly simplified example, the process is much more complex than this.

Random Pre-shared Keys can be generated automatically through the Web Interface but they can also be generated through the CLI using the command "pskgen".



Warning

Beware of Non-ASCII Characters in a PSK on Different Platforms!

If a PSK is specified as a passphrase and not a hexadecimal value, the different encodings on different platforms can cause a problem with non-ASCII characters. Windows, for example, encodes pre-shared keys containing non ASCII characters in UTF-16 while cOS Core uses UTF-8. Even though they can seem the same at either end of the tunnel there will be a mismatch and this can sometimes cause problems when setting up a Windows L2TP client that connects to cOS Core.

Using ID Lists with Certificates

A Typical Scenario

Consider the scenario of traveling employees being given access to the internal corporate networks using IPsec with certificates. The organization administers their own Certificate Authority (CA), and certificates have been issued to the employees. Different groups of employees are likely to have access to different parts of the internal networks. For example, members of the sales force might access servers running the order system, while technical engineers would access technical databases.

The Problem

Since the IP addresses of the traveling employees VPN clients cannot be known beforehand, the incoming IPsec connections from clients cannot be differentiated. This means that the security gateway is unable to correctly administer access to different parts of the internal networks using only the client's IP address.

The ID List Solution

One possible solution to this problem is to use Identification lists (ID lists). A cOS Core ID List object contains one or more ID objects as children. An IPsec Tunnel object can then have its

Remote ID property set to an ID list object. For a particular tunnel to be used by a particular client, the following must be true:

- The ID sent by the remote client must match one of the IDs in the ID list for the tunnel.
- The ID sent by the remote client must also exist as one of the IDs in the certificate the client sends.

When the client connects, cOS Core chooses the IPsec Tunnel object to use as follows:

- The connecting client sends its ID to cOS Core in the IKE negotiation.
- cOS Core scans its list of IPsec Tunnel objects looking for a match for the client.
- As an additional part of the matching process, cOS Core also checks the ID the client sends against the ID List of the tunnel. If it does not find an ID match, it continues searching through the IPsec Tunnel list. Any malformed IDs will be ignored and will also generate log message warnings.
- Once the matching tunnel is found, cOS Core then checks that the certificate the client sends also contains this same ID. If the certificate does, authentication is complete and the tunnel can be established. If the ID is not in the certificate, cOS Core flags that there is an authentication failure and the client connection is dropped.

This means that a particular IPsec Tunnel is only used by a particular client. The cOS Core configuration's IP rules and IP policies can then be designed to control which traffic can flow through which tunnel (the tunnel being an interface in the rule or policy)

Please note that this is one way to solve a differentiated user/tunnel scenario, there are other ways to solve this scenario as well.

Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario

Objective

This is the first recipe, in which we will launch into the IPsec topic by configuring an encrypted Lan to Lan tunnel (also referred to as Lan2Lan or L2L) between two cOS Core firewalls. To keep things simple we will be using pre-shared keys as authentication in many of the initial recipes. We will later move on to describe the use of Certificates as authentication as well. In this recipe we will also go through all the various settings that exists on an IPsec tunnel interface. This recipe can be used as a reference if more information is needed about a specific IPsec interface setting or option.

In this scenario we want to setup an IPsec Lan2Lan tunnel between Stockholm and London in order for devices/machines in Stockholm to talk to units/machines in London and vice versa. An overview of the scenario and the networks used is shown in figure 2.3.1.



Figure 2.3.1 Adding a new IPsec tunnel interface

Detailed Discussion

Requirements

For this first recipe, we have three requirements that need to be completed.

- Users in London should have unrestricted access to Stockholm's local network through the encrypted IPsec tunnel.
- Users in Stockholm should have unrestricted access to London's local network through the encrypted IPsec tunnel.
- Access should be allowed from only one network through the IPsec tunnel from each side.

Initial setup

The primary purpose of an IPsec Lan2Lan tunnel is to connect two networks together so machines behind each Firewall can talk to each other. What we need to do first, is to create the needed network objects on both the Stockholm and London Firewalls.

On the Stockholm Firewall we create the following two address book objects:

1. *Name=London_Endpoint*
Address=198.51.100.50
2. *Name=LondonNet*
Address=192.168.150.0/24

The first object is the public IP address on the London Firewall. The second object is the private network located behind the London Firewall that we want to reach.

On the London Firewall we create similar objects but this time reflect the public IP address of Stockholm and its internal network that London should reach.

1. *Name=Stockholm_Endpoint* *Address=203.0.113.100*
2. *Name=StockholmNet* *Address=192.168.100.0/24*

Configuring the IPsec tunnel on the London Firewall

To add a new IPsec tunnel in the WebUI we go to Network->Interfaces and VPN->VPN and Tunnels->IPsec. Choose "Add" and then "IPsec tunnel" as shown in figure 2.3.2.

Network » Interfaces and VPN » VPN and Tunnels » IPsec

IPsec

Manage the IPsec tunnel interfaces used for establishing IPsec VPN connections to and from this system.



Figure 2.3.2 Adding a new IPsec tunnel interface

The "General" tunnel settings tab

There are a lot of options and settings that can be used on an IPsec tunnel but in order to keep things simple in this first recipe we will only use the absolute basics in order to configure our tunnel. The first tab we are shown, is the **General** tab (1) as shown in figure 2.3.3.

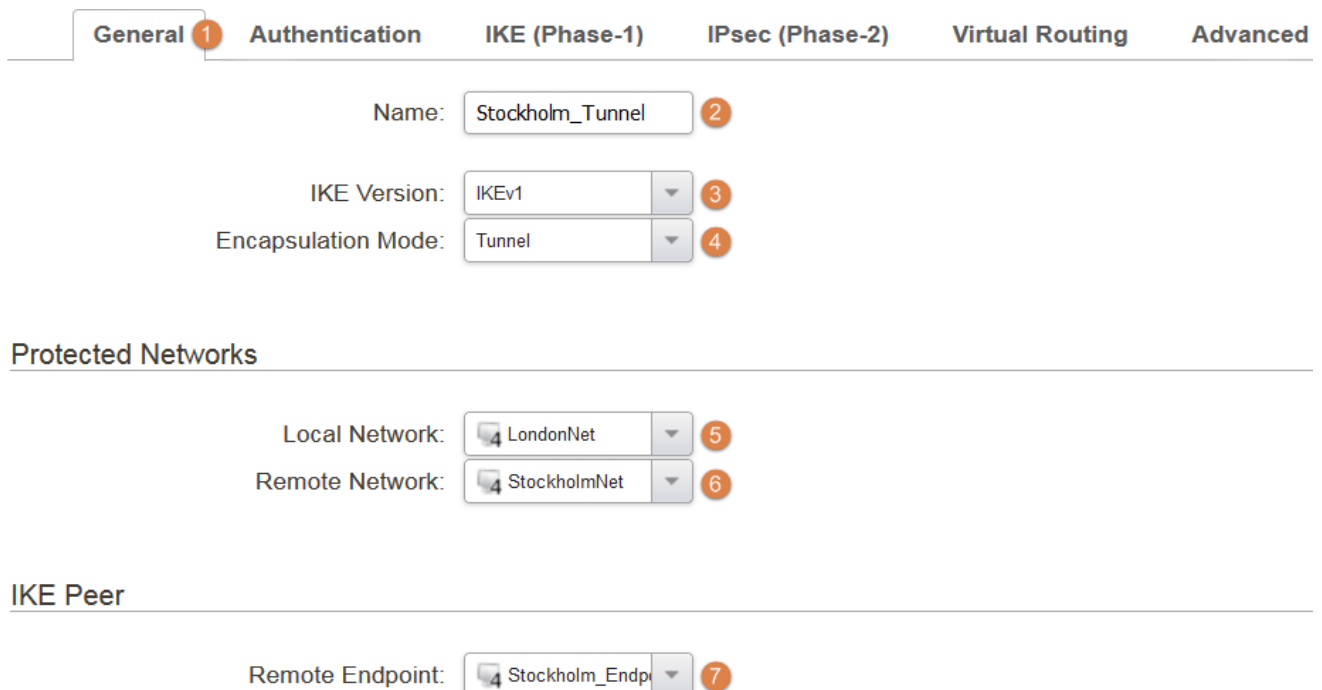


Figure 2.3.3 The options under the General tab of an IPsec tunnel interface

The first thing we do is to give our IPsec tunnel interface a name. In this case we name it **(2)** "Stockholm_Tunnel" as we are currently logged in to the London Firewall and we want to establish a tunnel towards Stockholm.

Next we have an option called "**IKE Version**" **(3)**. This option determines which IPsec IKE version that should be used. Two IKE versions exist, version 1 and version 2. We will use IKEv1 for the coming recipes as currently it is the most commonly used IKE version on the market. Near the end of this recipe (*Changing the tunnel from IKE version 1 to IKE version 2*) we will discuss and provide a little more information about IKEv2.

Encapsulation Mode **(4)** determines which mode we would like to use. There are two choices here, "Tunnel" and "Transport" mode. We will not go into details about the differences between the modes here, for more information please see the *Encapsulation Mode* section in *2.2. IPsec Tunnel Properties*. For this scenario we will be using main mode.

The **Local Network** **(5)** and **Remote Network** **(6)** options are one of the most important settings as it determines what networks should be allowed to communicate between each other in the IPsec tunnel. It is also a fairly common source of problems when it comes to troubleshooting as it can be quite easy to make mistakes in this area. We will discuss the use of multiple network definitions and more in *Recipe 2.6. Accessing satellite offices through central HQ using VPN*.

We select LondonNet as Local Network **(5)** and StockholmNet as the remote network **(6)**. These are network definitions that determine what network(s) we want to interconnect between the two offices. In this scenario we tell the IPsec engine that if users from LondonNet (192.168.150.0/24) want to talk to StockholmNet (192.168.100.0/24) or vice versa it will be allowed by the IPsec engine.

Please note that at this stage the traffic is still not allowed to flow due to Routing and IP policy restrictions, we will get to that further down.

The last option, **Remote Endpoint** **(7)** determines where the tunnel endpoint is. As we want to interconnect London and Stockholm, we need to choose/input the IP address of the device that the London firewall should contact, in order to establish the encrypted an IPsec connection. In our example we will use our previously created object called "Stockholm_Endpoint" (203.0.113.100) and this tells the Firewall where it should connect, as shown in figure 2.3.4.

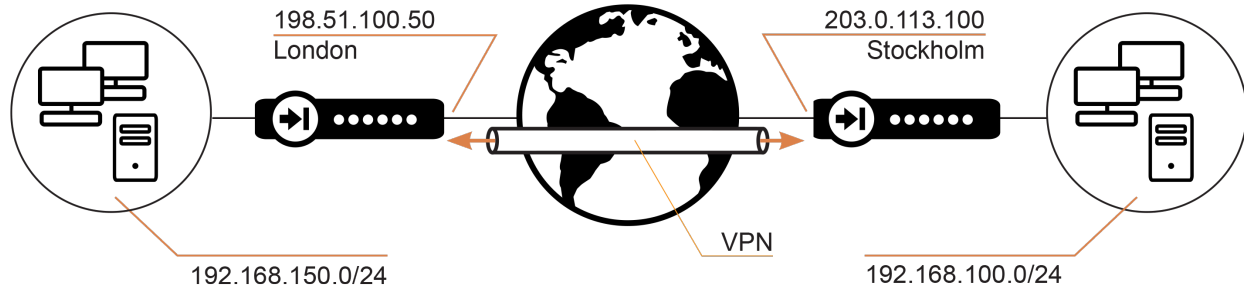


Figure 2.3.4 The IPsec tunnel will be setup between the two endpoints in London and Stockholm

The “Authentication” tunnel settings tab

The **Authentication** tab (1) consists of options and features related to how the tunnel should either authenticate itself or authenticate incoming negotiations from clients and other remote endpoints as shown in figure 2.3.5.

The screenshot shows the 'Authentication' tab (1) of the IPsec tunnel configuration. The 'Authentication Method' is set to 'Pre-shared Key' (2). Under the 'Pre-Shared Key' section, the 'Pre-shared key' is set to 'PSK-Secret' (3). The 'Authenticated Identities' section includes 'Local ID' (4), 'Remote ID' set to '(None)' (5), and 'Enforce local ID' (6). The 'EAP (IKEv2)' section has 'Require EAP for inbound IPsec tunnels' (7) unchecked. The 'XAuth (IKEv1)' section has 'Off' selected (8).

General **Authentication** (1) IKE (Phase-1) IPsec (Phase-2) Virtual Routing Advanced

Authentication Method: Pre-shared Key (2)

Pre-Shared Key

Pre-shared key: PSK-Secret (3)

Authenticated Identities

Local ID: (4)

Remote ID: (None) (5)

Enforce local ID: (6)

EAP (IKEv2)

Require EAP for inbound IPsec tunnels: (7)

XAuth (IKEv1)

Off (8)

Require IKE XAuth user authentication for inbound IPsec tunnels

Pass username and password to peer via IKE XAuth, if the remote gateway requires it

Figure 2.3.5 The IPsec tunnel's Authentication options

The first option we have in this tab, is the option to select **Authentication Method**. There exist two authentication methods today which are Pre-shared key (PSK) and Certificate (2).

We will discuss Certificates in much more details in *Recipe 2.7. Using Certificates as authentication on a Lan2Lan tunnel*, so for now we will use a Pre-Shared key as our choice of authentication.

The **Pre-Shared key (3)** means that we create a shared secret that can be either a passphrase or a hex-key on both London and Stockholm Firewalls. This secret key must be the same on both Firewalls so to make sure that the key is correctly set on both Firewalls, for testing/lab purposes we will, in this scenario, use a very simple passphrase of "test123". This is of course a very weak passphrase key and in a live environment such a weak phrase should never be used. We must make sure that the key is strong since regular words and phrases are more vulnerable to dictionary attacks.

This is all that is needed on the IPsec tunnel for the absolute basics. All the other options have standard default values that enable us to continue with the next step of our scenario right now. We will continue to list the contents of all the remaining options and tabs, in order to give a clear explanation of what all the options do. This explanation can be skipped if you are already familiar with these options and settings and jump directly to the *Configuring the IPsec tunnel on the Stockholm Firewall* section to configure the Stockholm side of the IPsec tunnel.

The **Local ID (4)** determines the Identify of the IPsec tunnel when initiating the tunnel towards the remote endpoint (Stockholm in this case). If the value is left blank the Firewall will use the Public IP of the London Firewall (198.51.100.50) but it can be set to almost anything, a name, a domain, a network. It is an identifier for the IPsec tunnel that is not necessarily enforced on the other side, in our example here, we leave it blank.

The **Remote ID (5)** is similar to Local ID but if configured/used we require that incoming IPsec negotiations match this tunnel and must present/use a specific ID in order to establish the connection. It is a method of adding another layer of authentication in order to gain access to protected resources. We will not use Remote ID in this example, so we leave it at default value (none).

Enforce **Local ID (6)**, this option determines whether the Responder Local ID value for incoming negotiations should match this tunnel or not. There are two Local ID values, "Local ID Responder" and "Local ID Initiator". If we are the initiator we can send a Local ID Responder identity based on the configured Remote ID setting (5). If more than one object exists in the Remote ID object, it will use the first object.

Require EAP for inbound IPsec tunnels (7) enables the use of Extensible Authentication Protocol, or EAP. EAP is an authentication framework frequently used in wireless networks and point-to-point connections. We will not cover the use of EAP in this book

XAuth (8) is an extension to the normal IKE exchange and is a way to require XAuth username/password for incoming connections. Or if we are the initiator, send a XAuth username/password to the remote endpoint. This is primarily used for roadwarrior/roaming client scenarios and will not be used in this recipe. We will cover the use of XAuth in *Recipe 2.4. Configuring IPsec roaming / road warrior IPsec*.

The “IKE (Phase-1)” tunnel settings tab

As mentioned in 2.1 IPsec Explained, tunnel negotiation consists of two phases, the first phase is called IKE and we will now go through the various settings and options as shown in figure 2.3.6

The screenshot shows the configuration interface for an IPsec tunnel, specifically the IKE (Phase-1) settings tab. The interface is organized into three main sections:

- Proposal:**
 - Diffie-Hellman group:** A list of available groups (01 to 18) is shown on the left, and group 02 (1024-bit) is selected in the 'Selected' list on the right. Buttons for '+ Include' and 'x Remove' are present.
 - Algorithms:** Set to 'High'.
 - Lifetime:** Set to 28800 seconds.
 - Mode:** Set to 'Main mode'.
- IKE Peers Settings:**
 - Outgoing Routing Table:** Set to 'main'.
 - Local Endpoint:** Set to '(None)'.
 - Incoming Interface Filter:** Set to 'any'.
- Miscellaneous:**
 - Dead Peer Detection:** Checked.
 - NAT Traversal:** Set to 'Only if needed'.
 - Auto Establish:** Unchecked.
 - DS Field:** Set to 0.

Figure 2.3.6 The IPsec tunnel's IKE (Phase-1) settings and options

One very practical aspect with both phase options is that if we want to setup a tunnel with the absolute basics, we do not need to change anything at all here. All the required options and suggestions on encryption methods and algorithms have been pre-selected by cOS Core. We will however go through all the settings starting with the **Diffie-Hellman group** selection (1).

By default cOS Core suggests the use of DH group 2, which has a key-length of 1024 bits, but we can select whatever group we want here. It is not limited to one group, but the group placed at the top will be the group used in the negotiation if we are the initiator of the VPN tunnel connection. For more information about DH please see *2.1. IPsec Explained*.



Warning

Using a group with very large key-length could cause problems on smaller devices (or VSG's) with limited CPU power, as it could cause a stall of network traffic passing through the Firewall when the DH key needs to be generated or renegotiated. Moderation is advised.

Algorithms and **lifetimes** (2) determine which encryption algorithms that should be used when negotiating the IKE phase-1. We will use the default proposal list called "High". This proposal list contains a pre-defined set of proposals that currently look as in figure 2.3.7.

Encryption Algorithms			
	Preferred	Min	Max
<input type="checkbox"/> DES	56	56	56
<input type="checkbox"/> 3DES	192	192	192
<input type="checkbox"/> CAST128	128	128	128
<input type="checkbox"/> Blowfish	128	128	448
<input type="checkbox"/> Twofish	128	128	256
<input checked="" type="checkbox"/> AES (Rijndael)	128	128	256

Integrity Algorithms
<input type="checkbox"/> MD5
<input type="checkbox"/> SHA1
<input checked="" type="checkbox"/> SHA256
<input checked="" type="checkbox"/> SHA512
<input checked="" type="checkbox"/> AES-XCBC (IKEv2 Only)

Figure 2.3.7 Details on the IKE & IPsec proposal list called "High"



Note

The recommended proposal list is subject to change as there may be some new algorithms and updated recommendations in the future.

Lifetime determines how long the negotiated IKE keys will be valid until a new set of keys needs to be created/negotiated.

Mode (3) determines which mode the Firewall should use when negotiating the VPN tunnel. The two choices are Main and Aggressive mode. The default value is Main mode. More information about Main and Aggressive mode can be found in the *2.1. IPsec Explained*. Simply put, aggressive mode is faster and can negotiate the tunnel with fewer negotiation steps/packets being sent between the initiator and responder but it is also slightly less secure. Main mode is recommended which is the value that we will use in this recipe.

Outgoing Routing Table (4) determines the routing table that cOS Core will use for the tunnel negotiation and encrypted traffic. It is not traffic that is being sent inside the encrypted tunnel. In other words, the routing table used to find the tunnel endpoint. (Can't understand these last two sentences)

Local Endpoint (5) specifies if the IPsec tunnel should only accept incoming negotiations towards a specific IP address in cOS Core. It will be used as the source IP for outgoing negotiations. The most common use for this option is when the Firewall has more than one public IP address and we want the IPsec tunnel to use another IP than the default interface IP address for either incoming or outgoing tunnel negotiations.

Incoming Interface Filter (6), as the name implies this is a filter for incoming IPsec negotiations. If an interface has been selected here, the IPsec tunnel will only attempt to match an incoming negotiation that arrives on the specified interface.

Dead Peer Detection (7) is a function that monitors the status of IKE/ESP traffic on an already established IPsec tunnel. If no IKE or ESP traffic has been observed within a reasonable time (usually around 20 seconds) it starts sending "Are you there?" queries (DPD-R-U-THERE) to the tunnel endpoint. If the tunnel endpoint is there and alive, it will respond with a DPD-R-U-THERE-ACK back, basically saying "I am here, don't tear down the tunnel please".

Several DPD queries are sent and if the remote endpoint does not respond to them due to one reason or the other, the IPsec tunnel will be torn down.

This function requires that the remote endpoint have support for DPD. It is rather unusual to find IPsec capable equipment today that does not support DPD.

NAT Traversal, also known as NAT-T, (**8**) is a way to handle situations where the client or remote endpoint is behind a device that performs NAT (Network Address Translation). This setting determines how the Firewall should behave when encountering NAT or if it should force a NAT-T negotiation whenever a tunnel is negotiated. Forcing NAT-T can be very useful in some scenarios where the ISP or something in between the Firewall and remote endpoint is not allowing the required IP protocol. By forcing NAT-T we can make the Firewall use the NAT-T port (UDP 4500) instead of the IP protocol (IP Proto 50).

Auto Establish (9) is a function that attempts to keep all tunnel network combinations up at all times. As an example, let's say, we have 5 networks configured as local network and 2 networks as remote networks. This means that there will be a total of 10 possible network combinations, Auto-Establish will attempt to keep all 10 established at all times.

This feature is quite aggressive and if a network SA is not up, it will attempt a new negotiation about once every second. If there are more than 10 network combinations, Auto-Establish should be avoided. We will not be using Auto-Establish in any of the recipes.

DS Field (10) is the Differentiated Service Field, it is primarily used for bandwidth prioritization together with Pipes and Traffic Shaping. It can, for instance, be used to increase or decrease the priority of the “negotiation packets” used by the IPsec tunnel. We will not cover the use of DS or traffic shaping in this book.

The “IPsec (Phase-2)” tunnel settings tab

The second phase of the tunnel negotiation, also known as IPsec phase, has the following options as shown in figure 2.3.8.

General Authentication IKE (Phase-1) **IPsec (Phase-2)** Virtual Routing Advanced

Proposal

1 Perfect Forward Security:

Available	Selected
14 (2048-bit)	None (No PFS)
15 (3072-bit)	01 (768-bit)
16 (4096-bit)	02 (1024-bit)
17 (6144-bit)	05 (1536-bit)
18 (8192-bit)	

+ Include × Remove ^ v

2 Algorithms: High

3 Lifetime: 3600 seconds

4 Lifetime: 0 kilobytes

Protected Networks Settings

5 Setup SA per: Network

6 Config Mode Pool: (None)

Miscellaneous

7 DS Field:

Figure 2.3.8 The IPsec tunnel’s IPsec (Phase-2) settings and options

Similar to IKE Phase-1, by default we do not need to change anything at all here if we want to establish a standard/simple tunnel. We can leave all the options at default values, because there is no input requirement in this section.

If **Perfect Forward Secrecy (1)** is used, a new Diffie-Hellman exchange is performed for each phase 2 negotiation. While this is slower, it makes sure that no keys are dependent on any other previously used keys. If we are the initiator, no PFS will be used as the "none" PFS group is placed at the top of the selection list. We would however accept PFS for group 01, 02 and 05 for incoming negotiations.



Note

It might be a good idea to change this option and remove "none" to make sure that PFS is always used in order to increase the security. In this recipe we will not change anything but leave things at their defaults.

Algorithms (2) determine which encryption algorithms that should be used when negotiating the IPsec phase-2. We will use the default proposal list called "High". This proposal list contains a pre-defined set of proposals that currently look as in Figure 2.3.7.



Note

Currently the default IKE and IPsec proposal lists called "High" are identically configured.

Lifetime in seconds (3) determines how long a newly negotiated IPsec (phase-2) tunnel should be considered valid. When the lifetime expires the system will perform a re-key and the counter will be reset. Please note that when it comes to IPsec Phase-2, there may be multiple negotiations done on the same tunnel based on how many networks are configured as local and remote network. We will go into more details regarding multiple networks on an IPsec tunnel in *Recipe 2.6. Accessing satellite offices through central HQ using VPN.*

Lifetime in kilobytes (4) is an optional settings where we can set an additional lifetime of the IPsec tunnel. In addition to only having a lifetime based on seconds we could also make the tunnel valid until a certain amount of data has been passing through the tunnel. The value that reaches zero first (Time or KB) will cause a tunnel re-negotiation/re-key in Phase-2.



Note

For the lifetime in KB, the data is calculated in both directions of the tunnel (the sum of data received and data sent).

Setup SA per (5), this option determines how the IPsec tunnel should establish the IPsec Security Associations (SA's). By default we setup an SA based on network but we have the option to use either Host or Port as well. As an example, an SA setup using the "network" option may look like the following CLI output:

```
IPsec Tunnel      Local Network      Remote Network      Remote Endpoint
-----
Stockholm_Tunnel  192.168.150.0/24  192.168.100.0/24  203.0.113.100
```

In the above CLI output we have one SA established between the network in London and Stockholm. It is basically a network definition of who can communicate with whom.

Using Host (or Port) means that there will be a tunnel negotiation between London and Stockholm for every host that needs to communicate. This means at worst case if we have two /24 networks it would mean $254 \times 254 = 64\,516$ number of IPsec SA negotiations that may need to be performed. We would hit the license limit very fast in such a scenario.

Using SA per Host or Port is very unusual, the recommendation is to use SA per network unless absolutely necessary to do otherwise. An example of when it would be preferable to use SA per host is if there is only communication between two hosts through the IPsec tunnel, such as the scenario of using L2TPv3 or GRE inside IPsec where communication is only performed between two hosts / IP addresses.

Config Mode Pool (6) is primarily used in a Roadwarrior/roaming scenario where we want to give the connecting client an IP address once connected, which would then be used to connect to allowed resources (e.g. an internal Terminal server). We will discuss and use Config Mode Pools in *Recipe 2.4. Configuring IPsec roaming / road warrior IPsec*.

The **DS field (7)** is present in phase-2 as well, the DS field in phase-1 enables us to prioritize the bandwidth of the negotiation (i.e ISAKMP) while in phase-2 it enables us to prioritize the bandwidth of the encrypted packets (i.e ESP). We will not discuss or use the DS field in this book.

The “Virtual Routing” settings tab

The Virtual Routing tab (1) is shown in figure 2.3.9 containing options related to Virtual Routing scenarios from where we want to control the traffic that traverses the tunnel in some special way.

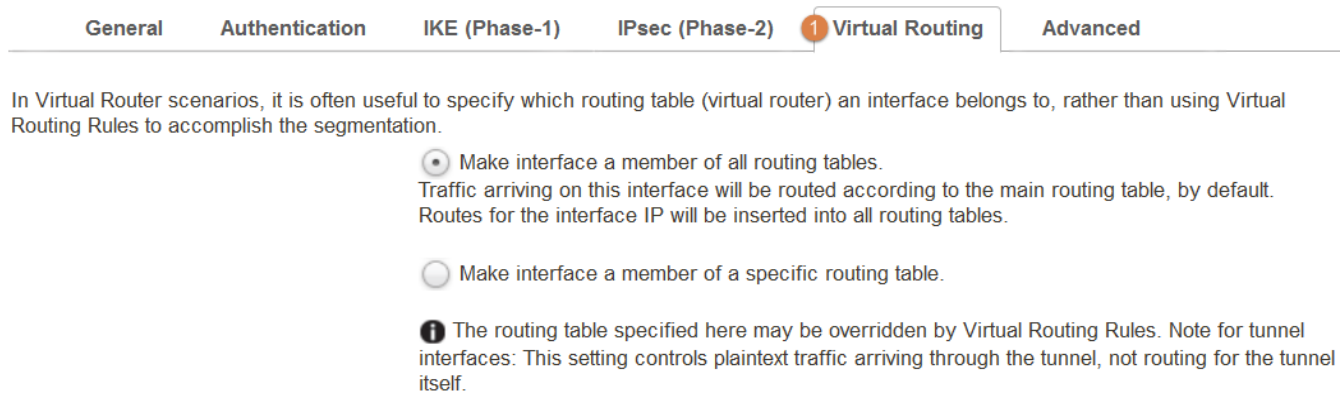


Figure 2.3.9 The “Virtual Routing” tab options

There may be scenarios or situations where the traffic inside the IPsec tunnel should use a specific routing table or exit to the internet using a secondary internet connection. The Virtual Routing tab can then be used to change the routing table that the IPsec tunnel by default will use to access relevant network(s). We will not go into further details about Virtual Routing in this book.

The “Advanced” settings tab

The advanced tab contains a couple of important options that even though they are placed in the advanced tab are quite frequently used. The advanced tab options are shown in figure 2.3.10.

The screenshot shows the 'Advanced' settings tab with the following options:

- Routing**
 - 1 Add route dynamically:
 - 2 Add route statically:
 - 3 Route Metric:
 - 4 Plaintext MTU:
- Tunnel Monitor** 5
 - Tunnel Monitoring:
- IP Addresses** 6
 - Automatically pick the address of a local interface that corresponds to the local net.
 - Specify address manually:

Figure 2.3.10 The “Advanced” tab options

All three of our first options are related to routing. The option “**Add route Dynamically**” (1) is primarily used in scenarios where the route towards the tunnel’s remote network should only be available once the tunnel is established. It is primarily used in roadwarrior/roaming scenarios with IPsec clients but it is also used in some Lan2Lan tunnel scenarios. An example of a Lan2Lan scenario is where the remote endpoint is behind a dynamic IP address so we are not able to establish the tunnel from our side. By having this option, we make sure that the route towards the remote network is only available once the tunnel has been successfully established.

The second option called “**Add route statically**” (2) means that when the IPsec interface has been created and the remote network option has been configured, cOS Core will automatically add a route in the <main> routing table that points to the remote network. This way we do not have to worry about manually having to add the route once the IPsec interface has been created.

In this recipe we will leave the option at default which is “enabled”. A route in the main routing table will now be created as it is shown in figure 2.3.11.



Note

The two route options can be enabled at the same time but there is no real scenario where this would be useful (the behavior of this setting may be subject to change in future versions of cOS Core).

# ▲	Type	Interface	Network	Gateway	LocalIP	Metric
1	Route IPv4	Stockholm_Tunnel	StockholmNet			90

Figure 2.3.11 The route that is created automatically if the “add route statically” option is enabled

The **Route Metric (3)** option determines the metric of the route that is either statically or dynamically created. Briefly explained, Metric is a way to tell cOS Core which route should be the primary, secondary etc, in case there are routes for networks that are identical. The route with the lowest metric will always be consulted first.

The default value of the IPsec metric is 90 and we will leave it at its default value.



Note

A physical Ethernet interface has default value 100. This means that if a route conflict exist, the tunnel route would be used first.

The **PlainText MTU (4)** setting controls the maximum payload size being sent inside the IPsec tunnel before they need to be fragmented. Even though the setting says “PlainText” it is important to note that it is basically all packets being sent inside the encrypted tunnel. This included packets that are encrypted such as HTTPS, ZIP files and even other IPsec ESP packets in case of a scenario where we want to send IPsec packets inside another IPsec tunnel.

“Plaintext” is defined as the Original IP Packet as shown in figure 2.3.12.

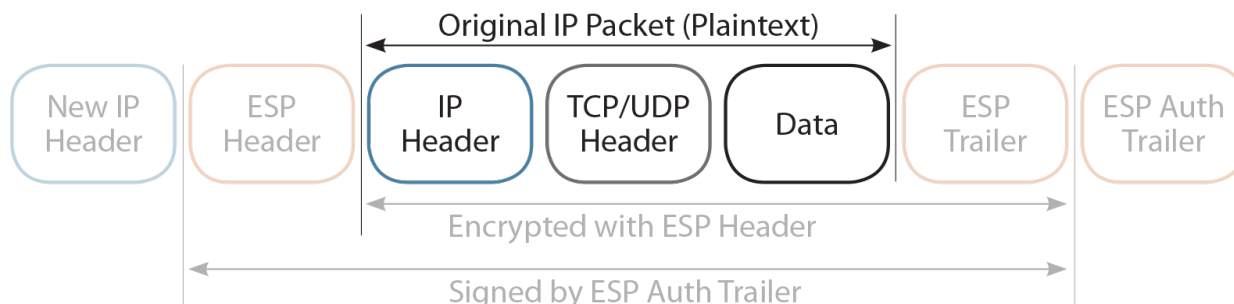


Figure 2.3.12 What is counted/included in the Plaintext calculation

Additional Information about the Plaintext MTU

The Plaintext MTU setting assumes that the Ethernet interface MTU is set to 1500. If we use an example with a lower value on the interface MTU e.g. 1400 it means that any encrypted ESP IPsec packets above size 1320 bytes need to be fragmented by the IPsec engine itself. This operation is more CPU resource demanding than fragmenting packets before they are sent to the IPsec interface so it is recommended that the interface MTU and the Plaintext MTU setting on the IPsec tunnel is correctly aligned to avoid ESP fragmentation.

If we use the default value of Plaintext MTU of 1420 it means 80 bytes up to the Ethernet MTU of 1500. If we make a break down to show the composition of each of these 80 bytes, we will use AES with SHA1 encryption in this example:

- 20 bytes is for IP header
- 8 bytes for UDP header
- 24 bytes for ESP + AES
- 14 bytes for ESP trailer
- 12 bytes for ICV (SHA1)

This decomposition accounts for a total of 78 bytes. The total size varies depending on the encryption algorithms used.

The next option is the **Tunnel Monitor (5)**, this is a feature that enables us to monitor a host that is part of the IPsec tunnel's remote network using ICMP/Ping. If the target host does not respond, cOS Core will tear down all IKE and IPsec SA's in question present in the tunnel. The IPsec engine will then try to reestablish the tunnel. This can be very useful to get a quick tunnel

re-connection in case of network/traffic problems inside the tunnel itself. Then the administrator does not have to perform a manual delete of the tunnel but rather cOS Core can perform this operation itself if it detects a problem.

The last option is called **IP Addresses (6)**. This option determines the IP address that cOS Core should use when traffic is either initiated from the Core itself or when we want to do some address translations such as NAT into the tunnel. By default cOS Core will try to use an interface IP address that corresponds to the chosen local network but that is not always possible.



Note

*Any IP address set as **IP Address (6)** will become automatically core routed.*

Configuring the IPsec tunnel on the Stockholm Firewall

We have now configured the London side of the IPsec tunnel. We must now configure a tunnel on the Stockholm side that matches the tunnel configured in London, as shown in figure 2.3.13.

Name:	<input type="text" value="London_Tunnel"/>
IKE Version:	<input type="text" value="IKEv1"/> ▼
Encapsulation Mode:	<input type="text" value="Tunnel"/> ▼
Protected Networks	
Local Network:	<input type="text" value="StockholmNet"/> ▼
Remote Network:	<input type="text" value="LondonNet"/> ▼
IKE Peer	
Remote Endpoint:	<input type="text" value="London_Endpoint"/> ▼

Figure 2.3.13 The configuration of the IPsec tunnel from Stockholm to London

An important detail to remember is when it comes to the networks. In our initial description of this recipe we have chosen the following networks:

```
Name=StockholmNet
Address=192.168.100.0/24
```

```
Name=LondonNet
Address=192.168.150.0/24
```

This means that StockholmNet is the local network and LondonNet is the remote network at the Stockholm side. But on the London side this is reversed, for easier understanding this is visualized in figure 2.3.14.

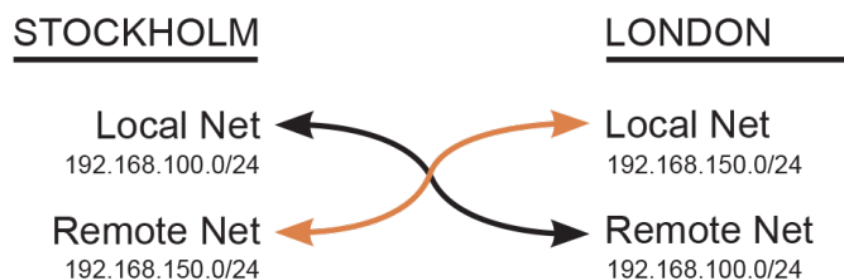


Figure 2.3.14 The relation between Local and Remote network on each side

On the Stockholm site, the remote endpoint is the public IP of the Firewall in London (198.51.100.50) to which the tunnel will link in order to establish the connection but also to accept incoming negotiations from it in case London is the initiator of the IPsec tunnel.

The last thing we need to do for the IPsec part is to create and add the same Pre-Shared key as used on the London tunnel, as shown in figure 2.3.15. As was indicated earlier, we simply use a passphrase consisting of the word "test123".

Pre-shared key:

Figure 2.3.15 The Pre-shared key used on the London Firewall

Configuring Routing

Since we used the option "Add route statically" on the IPsec interface both in London and Stockholm we do not have to make any changes or add routes to the routing table. This is handled automatically by cOS Core in this scenario.

Configuring IP Policies

We have now configured the IPsec tunnels on both sides, making sure that the routing is OK. The last thing we have left to do is to configure the IP policies needed to allow the traffic to but also from the IPsec tunnel interface.

What we want to do is to let the network in London and Stockholm communicate with each other without any restrictions. For this, we need a total of two rules on each side.



Note

We will not go into details about how to configure IP policies or rules as that has been covered in the first edition of the cookbook and also in the cOS Core administrator guide.

The two IP rules on the Stockholm side

On the Stockholm side we create two IP Policies as shown in figure 2.3.16.

Name	Log	Src If	Src Net	Dest If	Dest Net	Service
▶ Stockholm_To_London	✓	Lan	Lan_net	London_Tunnel	LondonNet	all_services
▶ London_To_Stockholm	✓	London_Tunnel	LondonNet	any	Lan_net	all_services

Figure 2.3.16 The two IP policies needed to communicate to/from the VPN interface

The first rule allows traffic to be initiated from the local internal network of Stockholm to the remote network in London that is routed on the IPsec tunnel. All services are selected as we want unrestricted access between these two networks.

The second rule allows traffic to be initiated in the reverse direction, meaning that users in London can also initiate traffic from their side in order to reach the internal network located in Stockholm.

One item that stands out on our rules is the destination interface of the second rule on both sides. It is set to be "Any" instead of the Lan interface. The reason for this is primarily for testing

purposes but it may also in some operational cases this may be necessary. "Any" is used because there is one IP address on the internal network that is **not** routed on the Lan interface but rather the Core interface. And that is the Lan interface IP address itself.



Note

For more information about the Core interface please see cookbook edition one and the administrator guide for cOS Core.

By setting the rule that allows traffic **from** the IPsec interface to be "Any" we also allow traffic to be sent to the IP address of the interface. This is very useful for testing purposes as we can then use the interface IP to test that the tunnel is working by sending a ping towards this particular IP address in either direction.

The two IP rules on London side

On the London side we create two rules as shown in figure 2.3.17.

Name ^	Log	Src If	Src Net	Dest If	Dest Net	Service
▶ London_To_Stockholm	✓	Lan	Lan_net	Stockholm_Tunnel	StockholmNet	all_services
▶ Stockholm_To_London	✓	Stockholm_Tunnel	StockholmNet	any	Lan_net	all_services

Figure 2.3.17 The two IP policies needed to communicate to/from the VPN interface

In London we create similar rules, allow traffic to be initiated from London's internal network towards the network in Stockholm and also vice versa.

Verifying that the VPN tunnel is working using the CLI

Once all the IPsec tunnels, routing and IP policies are configured it is time to test to see whether the tunnel is working. The easiest way to test this is to connect to cOS Core CLI and then ping a host routed beyond the IPsec tunnel. If we use the Stockholm firewall as our first unit to test from we initiate the following CLI command:

```
Stockholm:/> ping 192.168.150.1 -verbose

Sending 1 4-byte ICMP ping to 192.168.150.1 from 192.168.100.1
... using route "192.168.150.0/24 via London_Tunnel, no gw" in PBR table "main"
ICMP Reply from 192.168.150.1 seq=0 time=<10 ms TTL=255

Ping Results: Sent: 1, Received:1, Avg RTT: 10.0 ms
```

The ping issued in this instance is the interface IP address on the London firewall (192.168.100.1). cOS Core will perform a route lookup to try to locate where this IP address is and has determined that it should be sent to the "London_Tunnel" interface. Since we allowed this communication by setting "Any" as the destination interface in our IP Policy (see figure 2.1.16) we have a very useful IP to use for the initial testing. This, of course, can be any host on the London network that responds to ping but that may not always be the case.



Note

The first packet sent into the tunnel will trigger the tunnel initiation sequence, this means that the first packet (depending on timing) will in most cases have higher latency than subsequent packets.

We can also perform a similar ping test from the London firewall to Stockholm to verify that traffic can be initiated in both directions:

```
London: /> ping 192.168.100.1 -verbose
```

```
Sending 1 4-byte ICMP ping to 192.168.100.1 from 192.168.150.1
... using route "192.168.100.0/24 via Stockholm_Tunnel, no gw" in PBR table "main"
ICMP Reply from 192.168.100.1 seq=0 time=<10 ms TTL=255
```

```
Ping Results: Sent: 1, Received:1, Avg RTT: 10.0 ms
```



Note

In order to test the initiating sequence, e.g. making sure the tunnel can initiate from both sides, the tunnel has to be torn down before it can be properly initiated from a specific side.

A similar test can of course also be performed in the WebUI under Status->Tools->Ping, but the CLI provides a greater level of detail that can be very useful if we need to perform any troubleshooting. A similar ping done from the WebUI on the London firewall is shown in figure 2.3.18.

Ping

IP Address:

Number of Packets:

Packet Size:

Results of pinging to 192.168.100.1:		
Seq	Roundtrip	TTL
0	10 ms	255

1 packets transmitted, 1 packets received, 0% packet loss.
Round trip time average: **10 ms**.

Figure 2.3.18 Performing a ping test using the ping tool in the WebUI

To check the status of the IPsec tunnel itself we can use either the WebUI under Status->IPsec or use the CLI command "IKE" to see the status of the phase-1 IKE SA or "IPsec" to see the status of the phase-2 IPsec SA's (if more than one network). Below is the output displayed from London when using the IKE and IPsec commands when the tunnel has been established:

```
London:/> ike
--- Active IKE SAs:

Interface Local Endpoint Remote Endpoint Local ID Remote ID
-----
Wan      198.51.100.50 203.0.113.100 198.51.100.50 203.0.113.100

1 of 1 active SAs displayed

London:/> ipsec
--- Active IPsec SAs:

IPsec Tunnel Local Network Remote Network Remote Endpoint
-----
Stockholm_Tunnel 192.168.150.0/24 192.168.100.0/24 203.0.113.100

1 of 1 active SAs displayed
```

Changing the tunnel from IKE version 1 to IKE version 2

When it comes to selecting which IKE version to use, it is to the decision of the administrator. We are moving more and more towards IKEv2 and it will not be long until IKEv2 is selected by default when creating new IPsec tunnels.

Configuration of IKEv2 based IPsec tunnels is almost identical to the method used for IKEv1 with some subtle differences such as:

- Authentication using XAuth is only possible with IKEv1. Authentication with IKEv2 must use EAP.
- The AES-XCBC authentication algorithm is only supported by IKEv2. If AES-XCBC is used in a proposal list with IKEv1, it will be skipped. If AES-XCBC is the only algorithm in the proposal list with IKEv1, the tunnel setup will fail.
- The Encapsulation Mode property of an IKEv2 tunnel can only be "Tunnel". This means that IKEv2 cannot be used together with L2TP. The intended replacement for L2TP is to use a pure IKEv2 roaming VPN tunnel together with Certificates and EAP authentication.

In our example scenario we currently have a fully working Lan2Lan tunnel using IKEv1 between Stockholm and London. If we want to modify this tunnel to use IKEv2 we must do the following:

1. Open the IPsec tunnel on London & Stockholm.
2. Locate the option to change IKE version, as shown in figure 2.3.19.

The image shows a configuration interface for an IPsec tunnel. It consists of three rows of fields:

- Name:** A text input field containing the text "London_Tunnel".
- IKE Version:** A dropdown menu with "IKEv2" selected. The dropdown is highlighted with an orange border.
- Encapsulation Mode:** A dropdown menu with "Tunnel" selected. This field is grayed out.

Figure 2.3.19 Changing the IKE version on the IPsec tunnel in the Stockholm Firewall

3. Then change the IKE version from IKEv1 to IKEv2.
 - 3.1. Please note that the Encapsulation Mode presents as grayed out, this is because only tunnel mode is supported in IKEv2.

And that's it! We do not need to do any other changes. In this scenario, the proposal lists, network configuration etc. are all the same and can be used for IKEv2 as well.

Whenever possible, IKEv2 should be used.

Requirements check

As a last step in our first recipe we will do a quick check for whether we can meet the requirements we defined at the start of the recipe. If we break them down one by one:

- Users in London should have unrestricted access to Stockholm's local network through the encrypted IPsec tunnel.
 - In this recipe, the routing is automatically handled and setup by cOS Core. As long as we make sure to define and use the correct network on the IPsec tunnel's Remote Network setting the routing should be adequate.

Secondly we create two IP policy rules that allows traffic to/from the IPsec tunnel (see figure 2.3.16) without any restrictions when it comes to ports or protocols.

Status: Fulfilled.

- Users in Stockholm should have unrestricted access to London's local network through the encrypted IPsec tunnel.
 - Similar to the previous requirement the routing is handled automatically and the IP policies allow the communication to/from the IPsec tunnel. London can talk to Stockholm and vice versa without any port restrictions.

Status: Fulfilled.

- Access should be allowed to only one network through the IPsec tunnel from each side.
 - As we have only defined one network for use on both the Local and Remote networks on each IPsec tunnel for both London and Stockholm, it will not be possible to access any other network. To further lock down this requirement we have in our IP policies defined specifically which network should be allowed access to/from the IPsec tunnel interface on both the London and Stockholm Firewalls as shown in figures 2.3.16 and 2.3.17.

Status: Fulfilled.

Recipe 2.4. Configuring IPsec roaming / road warrior IPsec

Objectives

In this recipe we will configure cOS Core to accept incoming connections / negotiations from IPsec VPN clients. The definition of roaming or road-warrior varies depending on vendor but the basic principle is that one or more clients connect to the IPsec server (in our case cOS Core) from locations all across the country (or globe). Compared to a Lan2Lan tunnel, the administrator will not know from which specific endpoint IPs the clients are making the connection, which means we have to approach this problem differently. A client that can connect from a random (dynamic) or unknown IP is commonly known as Roaming or road warrior client. We will be using the term "Roaming" in this book.

The scenario is as shown in figure 2.4.1. The objective of this recipe is to configure cOS Core as the Roaming "server" to which clients around the world connect in order to access protected resources located behind the LAN interface of the Firewall.

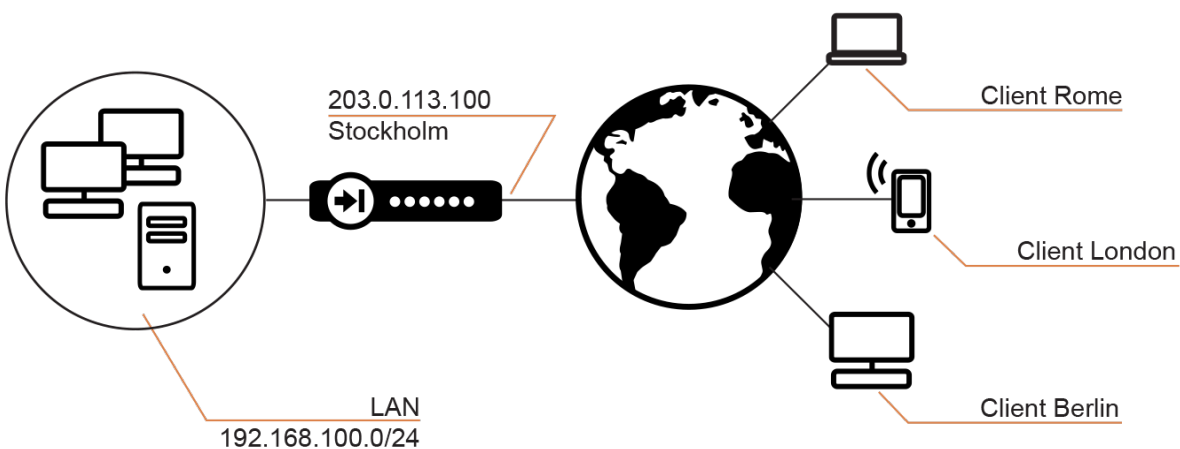


Figure 2.4.1 A common IPsec roaming scenario with clients around the globe

Detailed Discussion

Before we start to configure the new IPsec tunnel we will have a look at our current network setup on the Firewall we will use as the Roaming server. We have the following external IP address and internal network:

1. Name=Stockholm_Endpoint
Address=203.0.113.100
2. Name=Lan_Net
Address=192.168.100.0/24

Requirements

Before we start configuring the new tunnel we must look at what kind of requirements we may have. Since we want to keep things simple we only have two requirements in this scenario:

- Connected clients should be able to access the entire internal network (Lan_Net).
- Connected clients should be able to access the Internet through the tunnel.

In order to satisfy the second requirement, the administrator may consider applying network policies such as Anti-Virus, Application Control and more for connected clients. By letting clients use the VPN tunnel to surf/connect to the Internet we can increase the overall security of e.g. a sales representative that travels all around the world and may risk getting exposed to insecure Internet connections. By configuring the clients to access the internet through the VPN tunnel we add another layer of security apart from having Firewall and Anti-Virus software installed locally on the client devices.

Creating the roaming IPsec interface, general tab

The first thing we need to do is to create the IPsec interface itself. This is done under `Network->Interfaces->IPsec->Add`. Once the IPsec interface is created, we will proceed with configuring the new Roaming tunnel interface as shown in figure 2.4.2.



Note

We will not go into details about what each setting does, for more information about each setting in the IPsec interface section please see Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario.

The screenshot shows the configuration interface for an IPsec roaming interface. It features three tabs: **General**, **Authentication**, and **IKE (Phase-1)**. The **General** tab is selected. The configuration fields are as follows:

- Name:** Roaming_IPsec
- IKE Version:** IKEv1 (marked with a red circle 1)
- Encapsulation Mode:** Tunnel (marked with a red circle 2)
- Protected Networks:**
 - Local Network:** all-nets (marked with a red circle 3)
 - Remote Network:** CFG_Pool (marked with a red circle 4)
- IKE Peer:**
 - Remote Endpoint:** (None) (marked with a red circle 5)

Figure 2.4.2 Creating the IPsec roaming interface

For our Roaming tunnel, we will use the following settings. First the IKE Version (**1**) we will select IKEv1. But if the client has support for IKEv2 and it contains all the feature sets we require, it is recommended to use IKEv2. For now we will use IKEv1 as it is still the most common version to use for IPsec tunnels. In respect to configuration the difference between IKEv1 and IKEv2 is made

in this scenario only by the selection chosen in this box, so it can easily be switched to use IKEv2 in the future assuming of course that the VPN clients support it.

The Encapsulation mode (2) will use the tunnel model. No change from the default setting. For more information about tunnel and transport mode please see the *2.2. IPsec Tunnel Properties* section.

Local Network (3) will be configured to be all-nets. The reason why we want to use all-nets is because of the second requirement we had earlier. We want clients to be able to surf the internet through the tunnel. As we do not know which IP address the client would like to access on the internet, we must allow the client to request any IP through the tunnel.

We must remember that the Local Network on the Roaming server is the Remote Network for the client. From the client's perspective, it needs to be able to access every IP and network, and it will be configured to use the IPsec tunnel to achieve that. See figure 2.3.14 in the Basic IPsec Tunnel recipe for an example of the correlation between Local and Remote network.

Remote network (4) is where we configure the network from which the client will connect and use as source network. In this case we have chosen the object called "CFG_Pool". This is a special object that is used by the Config Mode pool. We will discuss the usage of Config Mode further down in this recipe. Basically we restrict the clients to only be able to use an IP/network as source that is part of the Config Mode pool. By doing this we minimize the risk of a client using a source IP or network that conflict with a server or host on our internal network when connected. This should not happen when using CFG Mode as we tell the connecting client what IP it should use but it never hurts to add a second layer of IP/network verifications.

The Remote Endpoint (5) will be left at the default value, which in this case is <none>. This means that we do not specify any network(s) for use as the remote endpoint for the tunnel. This would be the same as if we had chosen "all-nets", which means all IP addresses. We must do this as we do not know from which IP our client(s) will connect from and is also the reason why it is referred to as a roaming IPsec tunnel. The client may "roam" around the world and connect from a large number of different IP addresses. An example would be a traveling sales representative who connects to his work network from various hotels around the world.

Sidenote: It is possible to combine Geolocation with this to only allow access to the IPsec server from specific countries. But this is a scenario that will not be discussed in this book as it requires the advanced setting "IPsecBeforeRules" to be disabled.

Creating the roaming IPsec interface, authentication

The next step in the configuration is to configure an authentication source. To keep things simple we will be using a pre-shared key together with XAuth username/password. First we create and set a pre-shared key on the IPsec tunnel as shown in figure 2.4.3.

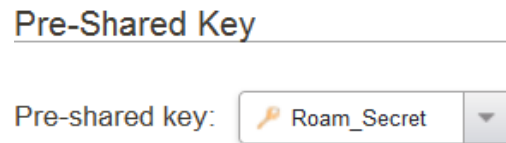


Figure 2.4.3 Adding a pre-shared key to the roaming tunnel

This Pre-shared key must be provided to all clients that want to connect to our roaming “server”. It is, however, recommended not to send the pre-shared key in a normal mail but use an SMS service or similar. For example, send the connection information (IP of the server etc) via e-mail but the PSK in an SMS.

The second feature we want to activate on the Authentication tab is XAuth as shown in figure 2.4.4.

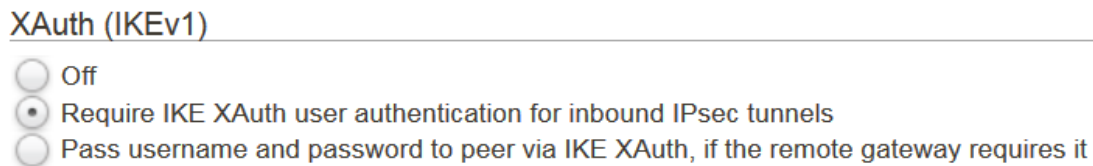


Figure 2.4.4 Enabling XAuth on the IPsec tunnel

XAuth is an extension to the normal IKE exchange and is a way to require that connecting users input a valid username/password in order to establish the VPN connection. Since we want to know who connects to our Roaming IPsec server we enable this option. We will explain on how to link this particular feature to a specific user database later in this recipe.

Creating the roaming IPsec interface, phase-1

For the phase-1 settings we do not change anything from default, we leave every setting as is. For a figure of how the default setting is configured please see figure 2.3.6 in previous recipe.

Creating the roaming IPsec interface, phase-2

For phase-2 we do not change any of the settings except for the Protected Networks Settings. Here we need to specify a Config Mode Pool as shown in figure 2.4.5.

Protected Networks Settings

Setup SA per:

Config Mode Pool:

Figure 2.4.5 Selecting a Config Mode Pool to use for our roaming tunnel

Config Mode is similar to a DHCP server for VPN clients. It is a method of assigning IP, Netmask, DNS and more to connecting clients as shown in figure 2.4.6. We will now go through the various Config Mode Pool settings.

Roam_Pool

An IKE Config Mode Pool will dynamically assign the IP address, DNS server, WINS server etc. to the VPN client connecting to this gateway.

Name:

IP Pool Type

1 Pre-defined IP Pool Object
 Static IP Pool

2 IP Pool:

3 Netmask:

Optionals

4 DNS:

5 NBNS/WINS:

6 DHCP:

7 Subnets:

8 Prefixes:

Figure 2.4.6 Config mode options

The first option (1) is to select whenever we want to use a static IP pool or a pre-defined IP pool object. A pre-defined IP Pool object is a dynamic object which consists of IP leases that are pre-fetched from a DHCP Server.

In order to keep things simple we use the default "Static IP pool" type. For this type we must define an IP pool to use. In our address book we create an IP4 address object called "CFG_Pool" with the following value:

```
192.168.240.1-192.168.240.100
```

We then use this newly created object as our "IP Pool" **(2)** in our Config Mode Pool.

The Netmask **(3)** determines the size of the network. In our scenario the size of the internal network is /24 so then the netmask will be 255.255.255.0.

DNS **(4)** address is the address of the DNS server we want the connected IPsec client to use. We will use our external DNS server that we received from our ISP, but it might as well be an internal DNS server, it will be up to the administrator depending on the scenario.

There are also some less frequently used options that exist in the Config Mode Pool options, these options are the following.

The NBNS/WINS **(5)** option enables us to set an IP of the Windows Internet Name Service (WINS) servers that will be handed out to connecting clients. It can be used in Microsoft environments which uses NetBIOS Name Servers (NBNS) to resolve IP addresses to NetBIOS names.

DHCP **(6)** enables us to specify the IP address of a DHCP server in case the connecting client sends any DHCP requests. These requests will be forwarded to the specified IP address.

The Subnets **(7)** option enables us to send additional networks to the client. This will be needed in more advanced split tunneling scenarios where the connecting client needs access to more than one network. This option will be discussed and used further in *Recipe 2.5. Configuring IPsec roaming split tunneling with multiple networks..*

The Prefixes **(8)** option is the IPv6 equivalent of the previous Subnets (7) option. If we want to send additional IPv6 network(s) to the client this is the option to use. Since we are not using IPv6 in this scenario we will leave it blank.

Creating the roaming IPsec interface, Advanced

We will not change anything from default in the Virtual Routing tab so we move on to the Advanced tab as shown in figure 2.4.7.

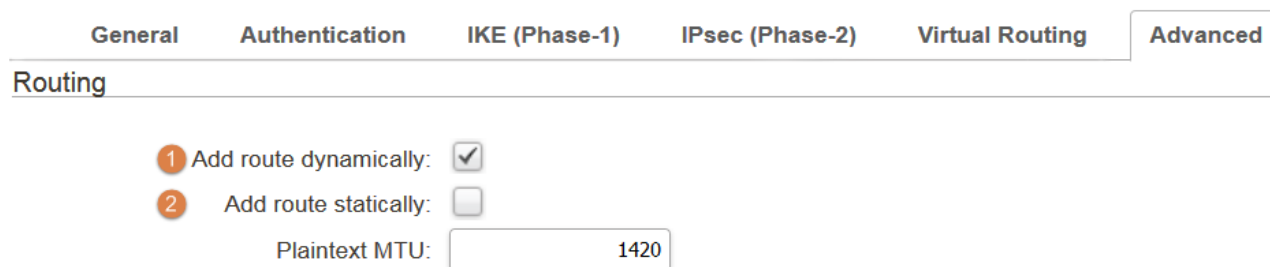


Figure 2.4.7 The advanced settings on the roaming IPsec tunnel

There are two options in the advanced tab that we need to change from the default. The first option is the “Add route Dynamically” that we want to enable, the second is the “Add route Statically” that we want to disable.

As we are connecting with a roaming client that can arrive from any source IP we are using Config Mode Pool to hand out an IP address to the client. In order to keep the various clients separated we want cOS Core to keep track of which IP belongs to what IPsec tunnel session, and the first option (1) does just that. cOS Core achieves this by dynamically adding a route to the client for the IP it receives from the Config Mode Pool. Then cOS Core will know which IP belongs to what IP tunnel session. This route only exists as long as the client is connected, when the client disconnects the route disappears.

This is also the reason why we need to disable the second option as such a route would first of all be for the entire Config Mode Pool range and it exists statically, meaning that it still persists after the client(s) has disconnected from the server.

Once a client is connected the dynamic route will be shown in the CLI output of the routing table like this:

Flags	Network	Iface	Gateway	Local IP	Metric
D	192.168.240.1	Roaming_IPsec			0

The flag “D” means “Dynamically added”. In this case it is a route added automatically by the IPsec engine once the client has successfully connected. The network is the IP address the connecting client received from the Config Mode Pool.

This concludes the IPsec part of the setup, we will now continue with the username/password part of the negotiation.

Adding username and password requirement

We have so far configured the IPsec tunnel but we want to add another layer of security by forcing our users to input a username and password in order to establish the tunnel.

Creating and adding users to the Local User Database.

The first step is to create a new Local User Database. To accomplish this, we go to System->Device->Local User Databases and create a new database as shown in figure 2.4.8.

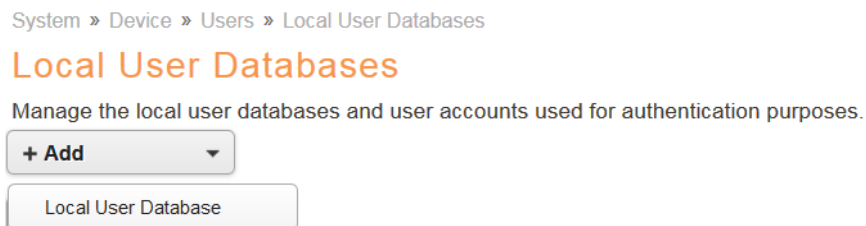


Figure 2.4.8 Creating/adding the Local User Database

Once the database is created, we call the database “Roaming_Users” we can start adding users to the database. The General tab contains the name of the database and a comment field while the second tab, as shown in figure 2.4.9, contains the ability to add/remove and edit users in the database.

Roaming_Users

A local user database contains user accounts used for authentication purposes.

Name ▲	Groups	IP Pool	Networks	Strong Password	Comments
Bob				No	
Ann				No	

Figure 2.4.9 The user summary in the Local User database

There are additional options available on each user but to keep things simple we will not discuss or use them in this recipe, we simply create a few users and set their passwords.

User Authentication, creating the rule

Once the database has been created we now want to create the user authentication rule. Previously we activated XAuth on our IPsec tunnel but we now must link this function to a user authentication rule. To do that we go to (in the WebUI) Policies- >User Authentication->Authentication Rules and create a new User Authentication rule as shown in figure 2.4.11.

Policies » User Authentication » Rules » Authentication Rules

Authentication Rules

The Authentication Rules specifies from where users are allowed to authenticate to the system, and how.

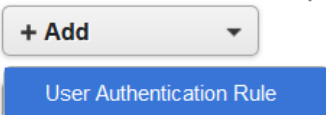


Figure 2.4.10 Creating the User Authentication Rule

User Authentication, general options tab

Once the rule is created we are greeted but several tabs of options as shown in figure 2.4.12.

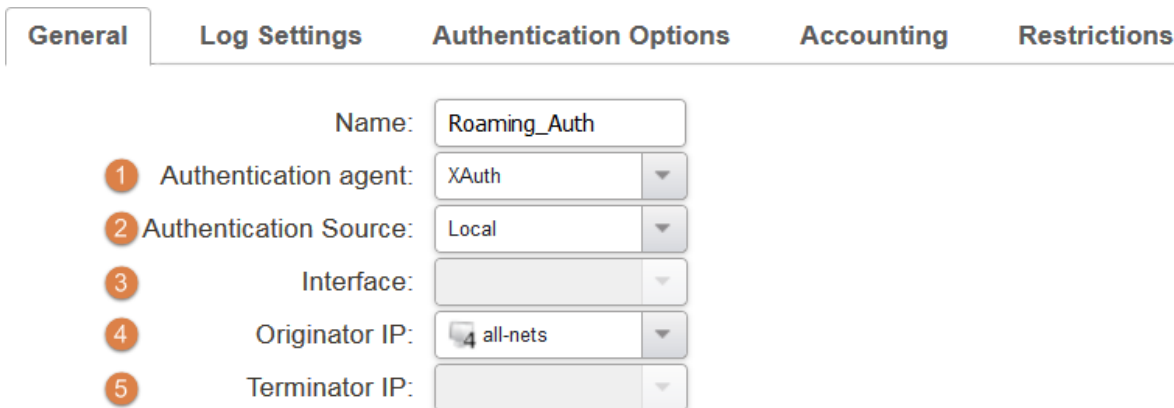


Figure 2.4.11 The General options on the User Authentication rule

We will go through all the tabs (some briefly) in order to give details on how we can connect our IPsec tunnel XAuth function to a User Authentication rule.

We start with selecting the Authentication Agent (**1**). This determines what type of authentication we want our Authentication rule to trigger on. Since we have activated XAuth on our IPsec tunnel we select the XAuth authentication agent.

There are of course other Authentication Agents such as HTTP, HTTPS, EAP etc. but they will not be used in this recipe.



Note

*The XAuth agent is a slight exception to other User Authentication rules, because even though we can create multiple XAuth agent rules, only the one at the top will trigger. This means that we cannot use different authentication databases based on an IPsec interface. The selected database on the **first** XAuth rule will be used for all IPsec interfaces that use XAuth.*

The Authentication source (**2**) determines what kind of authentication database we want to use for our users. We have multiple choices such as Local, LDAP, Radius etc. To keep things simple we will use a Local database which means that the database containing the username and passwords for our users will be located in cOS Core's configuration. This solution is viable for about 20-40 users but any more users than that and it is recommended to use a system that is better suited for large scale user database handling such as Radius or LDAP.

The interface (**3**) is a way to use different user databases, and is based on the interface on which the user authentication request will be received. Due to the limitations using XAuth, mentioned in the previous note, this option is grayed out and it will not be possible to choose anything here when using XAuth as the agent.

Originator IP (**4**) determines which IP addresses/network that will be allowed to connect and authenticate towards the roaming IPsec tunnel. This means that only clients that belong to the Originator IP/Network will be allowed to authenticate using XAuth. Since this is a roaming tunnel we do not know from which IP the clients are connecting from, we will leave this setting on its default value which is All-nets. We allow anyone that can get past the phase-1 negotiation of the IPsec tunnel to perform a login attempt.

Terminator IP (**5**), this setting can (depending on Authentication Agent (field 1)) limit incoming authentication attempts to only be triggered if the client connects to a specific IP address

(endpoint). As we are using IPsec and the XAuth agent this part is handled by the IPsec tunnel interface and the reason why the option is grayed out.

User Authentication, log settings tab

The Log Settings tab contains an option that activates cOS Core to generate log events if a user performs a failed authentication attempt or is logged in/out. It is strongly recommended to leave this option at the default value, which is enabled.

User Authentication, authentication options tab

The Authentication Options tab contains options related the Authentication Source we selected earlier (see *Figure 2.4.11*) and is shown in figure 2.4.13.

Once the rule is created we are greeted but several tabs of options as shown in figure 2.4.12.

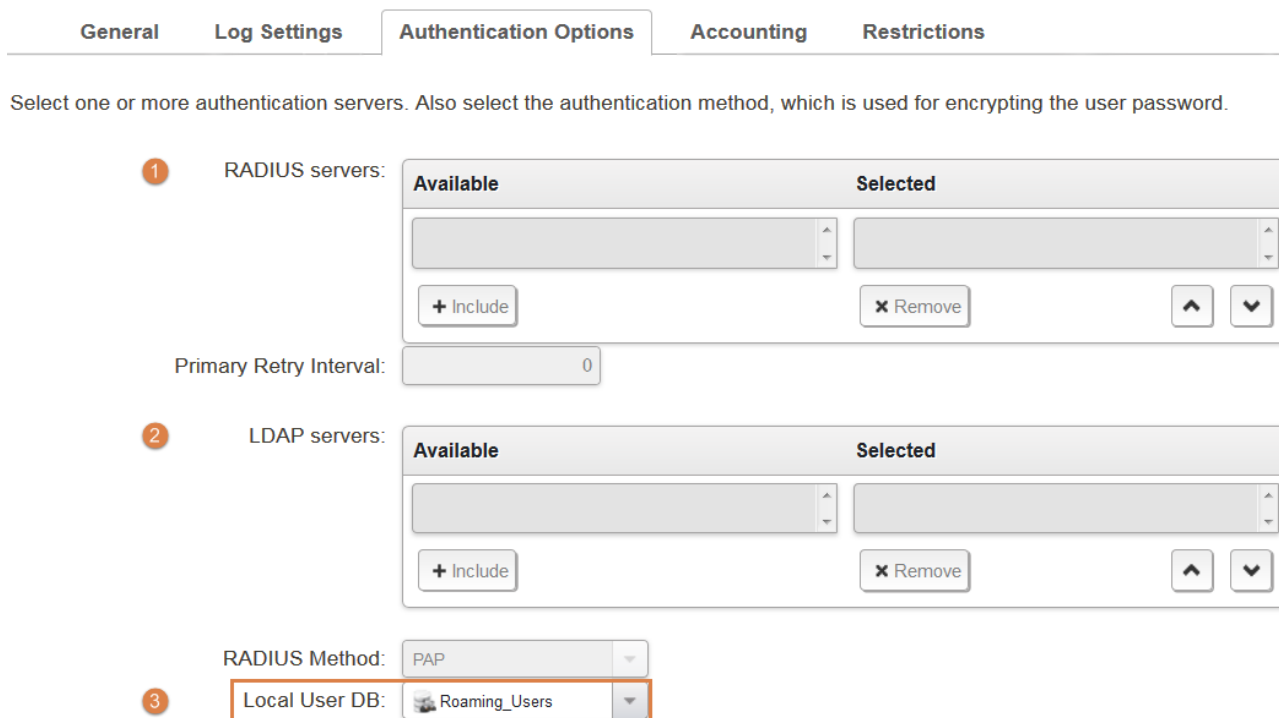


Figure 2.4.12 The Authentication Options on the User Authentication rule

Since we have not selected either Radius (1) or LDAP (2) as our authentication agent these choices are grayed out. We do, however, have to select which Local User Database (3) to use.

A local User Database means that the user database is stored in the cOS Core configuration file directly without the need to contact any external server as with Radius or LDAP.

By default there exists one pre-defined local database called "AdminUsers". This database is the database used when connecting to cOS Core using SSH or the WebUI. We can use this database if we so choose, but it would be much better to create a new local database that is specifically used for our roaming clients (also we do not want our VPN users to be able to login to the WebUI of the Firewall).

User Authentication, accounting options tab

The accountings tab is used if we want to use Radius accounting. Radius accounting is, simply put, a way to track how much data a user has sent/received. It can be useful for scenarios involving data transfer limits where a user is only allowed to send/receive certain amount of data before the need to buy more.

We will not use Radius accounting in this scenario so we will not go into any detail about the various options under the Accounting tab.

User Authentication, restrictions option tab

The Restrictions tab contains options related to user restrictions such as idle timeouts as shown in figure 2.4.10.

General Log Settings Authentication Options Accounting Restrictions

Timeouts

1 Idle Timeout: 1800 seconds

2 Session Timeout: seconds

3 Use timeouts received from the authentication server.

i Note that if no timeouts are received, OR if this checkbox is unchecked, the above settings will be used.

Multiple Username Logins

4 Allow multiple logins per username
 Allow one login per username, disallow the rest.
 Allow one login per username, disallow the rest without any exception. (strict)
 Allow one login per username,

Replace existing user if idle for more than: 10 seconds

Figure 2.4.13 The Restriction options on the User Auth rule

The first option is Idle Timeout (**1**), this setting controls how long a client can be left idle (without generating and traffic) before he/she is automatically disconnected from cOS Core. We leave it at the default 1800 seconds (30 minutes) value.

Session Timeout (**2**) controls how long the client is allowed to stay connected no matter if he/she is idle or not. This means that even if the client is being active, he/she will get disconnected once the configured session timeout reaches zero. We will not use any session timeout in this scenario. It is quite unusual to use this particular option but it can be useful if there is a large amount of users connected on a daily basis to the server and to avoid that partially idle users are clogging up the system needlessly.

The “User timeouts received...” (**3**) option is an option only available when Radius or LDAP is used and a timeout value is sent in the reply from the authentication server. As we are using Local Database this option is grayed out and will not be used.

Lastly we have various options that let us control how to handle Multiple Username Logins (**4**). It will be up to the administrator to decide which option that would be applicable to the situation/scenario. For this recipe we will use the default “Allow multiple logins” which means that the same user can login multiple times using the same account. It can be advantageous in case the user is on a shaky connection and risk getting disconnected on and off, then instead of having to wait until the previous user session expires he/she can simply login and create another session. The drawback would be if the username/password is compromised it can be used to login multiple times from the same account, significantly increasing illegal access to the compromised network.

Creating the IP policies needed for network access

Before we create the IP policies that determine what kind of access our connected VPN clients should have, let’s again review the two requirements we had for access mentioned initially in this recipe:

Requirement-1: Connected clients should be able to access the entire internal network (Lan_Net).

Requirement-2: Connected clients should be able to access the Internet through the tunnel.

Up until now we have configured the IPsec roaming tunnel and the needed pool, and the User Authentication rule. Clients can now connect to cOS Core but they are still unable to access any resources. To do that we need to configure some IP policies based on our requirements.

First we create the rule that give our client access to the internal network (Lan_Net) as shown in figure 2.4.14.

# ^	Name	Log	Src If	Src Net	Dest If	Dest Net	Service	Address Translation
1	▶ Roaming_To_Lan	✓	Roaming_IPsec	CFG_Pool	any	Lan_net	all_services	

Figure 2.4.14 The first IP Policy that provides connected VPN clients with access to the internal Lan network

Please note that there is no address translation used. There is no need to mask the IP address from which the clients are connecting when accessing the internal network. NAT should be avoided whenever possible for logging purposes as we can more easily track potential problems or even attackers.



Note

The reason why we set Destination Interface to “Any” is because we would like to be able to send ping/ICMP packets to the Lan interface IP itself for testing purposes (or for management). Since this IP is routed on Core it will not trigger if we select “Lan” as the destination interface. If there is no such need, we can set destination interface to be the Lan interface instead.

The second rule is to fulfill the second requirement of providing connected clients with access to the Internet using the VPN tunnel as shown in figure 2.4.15.

# ^	Name	Log	Src If	Src Net	Dest If	Dest Net	Service	Address Translation
1	▶ Roaming_To_Lan	✓	Roaming_IPsec	CFG_Pool	any	Lan_net	all_services	
2	▶ Roaming_To_Wan	✓	Roaming_IPsec	CFG_Pool	Wan	all-nets	all_services	SRC:NAT

Figure 2.4.15 The second IP Policy that provides connected VPN clients with access to the Internet through the VPN tunnel

Since we are allowing clients to reach the Internet, we must NAT the traffic from the connected VPN clients. This means that when these clients connect to the Internet their source IP will be the public IP of the Firewall and not their PC at home. This is the same principle that many

anonymizer service providers are using to protect or hide users by connecting to a VPN server that then mask their source IP using NAT.

And that concludes this recipe on how to configure cOS Core to act as a server for incoming roaming VPN clients.

Questions & Answers:

Question: Why is the CFG mode pool not part of the local network in this scenario?

Answer: The reason for this is because the IPsec tunnel interface does not have ARP or ProxyARP support. Meaning that if we give out IP address in the internal network, internal hosts would send an ARP query to try locate the host they believe belongs to their own network segment, and if cOS Core does not reply to those queries, the communication will fail. By handing out IP addresses in a completely different subnet IP range the internal clients will simply ask their default gateway, which in this scenario will be cOS Core and then the communication will work without any problems.

Question: But I want my connected clients to be in the same subnet, is there no alternative solution?

Answer: There is an alternative solution. We can use a behavior in cOS Core where it will respond to ARP queries even on the route the ProxyARP is not configured on. This is briefly how it can be configured.

1. Change the CFG mode pool to be part of the local network, e.g. 192.168.1.200-192.168.1.220
2. Create a route in the routing table that looks like this:

```
Route Roaming_IPsec CFG_Pool ProxyARP=Lan
```

When clients now connect, cOS Core will setup a single-host route for the IP the client receives from the IP pool but since the ProxyARP and the routes are using separate sub-processes they are looked at separately. So even though the ProxyARP route does not primarily match (as it is bigger than the single-host route), it will still respond to the ARP query and the scenario workaround would work.

Question: Why is “all_services” used on the IP policies. Is that wise?

Answer: This is primary for testing purposes that all_services is used as it includes all IP protocols, including TCP, UDP and ICMP. When configuring this for a live production environment it is recommended to either use a service group or create multiple IP policies only for the specific services/ports that we want to allow such as DNS, HTTP, HTTPS, FTP etc.

It is also a good opportunity to apply additional security functions on the traffic towards the Internet such as Anti-Virus, Web Content Filtering, Intrusion Detection etc. to protect clients that are traveling around the world from malicious content. Not to mention to increase security and protection of the servers that is being accessed through the VPN tunnel.

Recipe 2.5. Configuring IPsec roaming split tunneling with multiple networks.

Objective

The objective of this recipe is to use the previous recipe as base but with slightly different requirements. In this recipe, we want our roaming users to be able to reach three internal networks instead of just one, also we will not allow internet access through the VPN connection.

The scenario is depicted in figure 2.5.1.

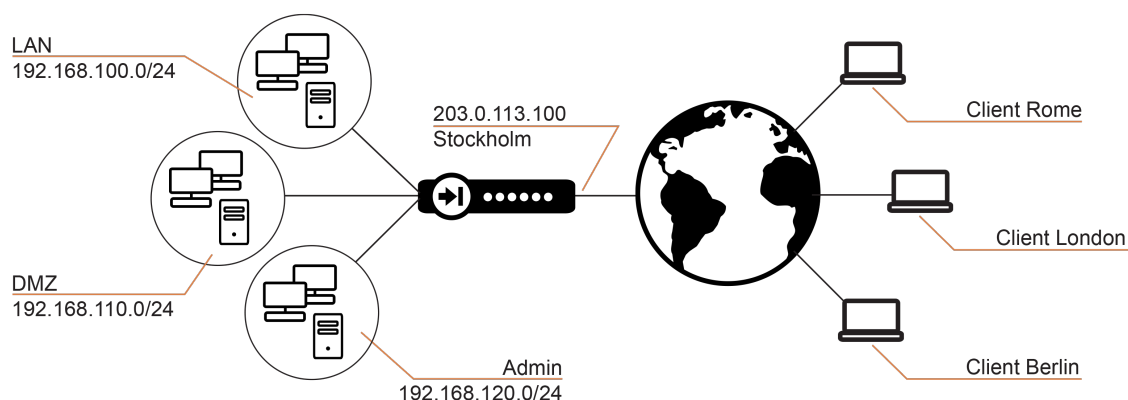


Figure 2.5.1 A roaming VPN scenario with clients around the globe that wants access to multiple internal networks

Detailed discussion

Requirements

We have the following requirements for this scenario:

- **Requirement:** Connected clients should be able to reach all the internal networks as shown in figure 2.5.1.
- **Requirement:** Connected clients should **not** be able to reach the internet through the VPN connection. They should use their normal ISP connection when surfing the internet.

Initial setup

This recipe will be using the previous recipe as base and assumes the roaming VPN connection is already configured with the previous recipe parameters.

We will use the following objects that we have created in the address book:

1. Name=Lan_Net
Address=192.168.100.0/24
2. Name=DMZ_Net
Address=192.168.110.0/24
3. Name=Admin_Net
Address=192.168.120.0/24

Modifying the IPsec tunnel

In our previous recipe we used the network configuration as shown in figure 2.5.2 on our IPsec tunnel.

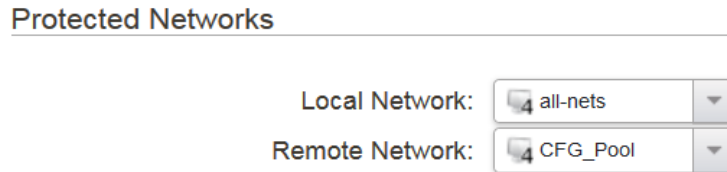


Figure 2.5.2 The current network configuration on the roaming IPsec tunnel

The local network was set to be "all-nets", just as our previous requirement demanded, so that they should be able to surf to the internet. Since we no longer allow "all-nets" due to the new requirement, the Local Network definition needs to change to reflect only the specific networks to which the roaming VPN users should be granted access.

To accomplish this we create an address book group with the 3 local networks as shown in figure 2.5.3. We call this group "Local_Net_Group".

Local_Net_Group

An IP4 Address Group is used for combining several IP4 Address objects for simplified management.

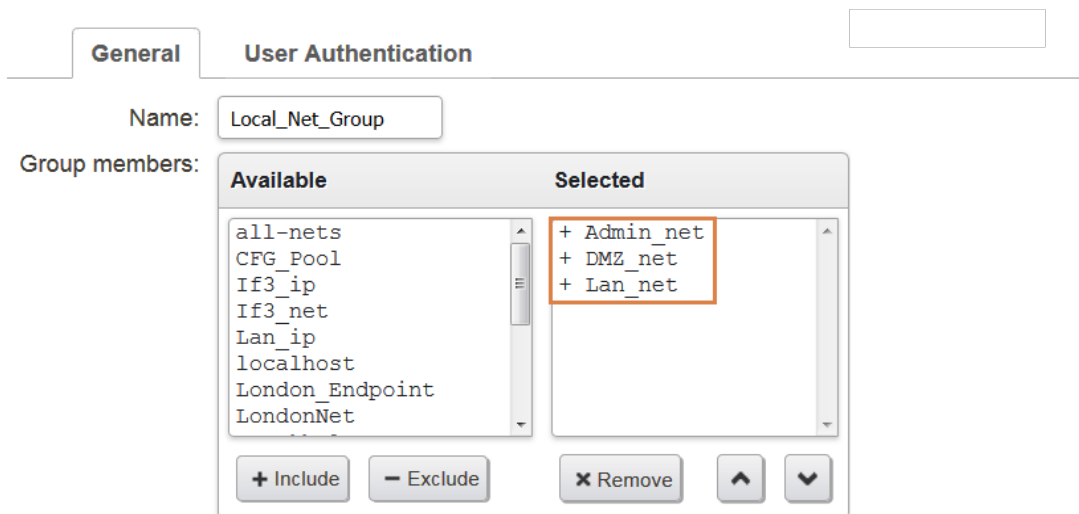


Figure 2.5.3 The network group consisting of the 3 internal networks as members

Once the group is created we change the Local Network on our IPsec tunnel to use the group object instead of all-nets as shown in figure 2.5.4.

Protected Networks

Local Network:

Remote Network:

Figure 2.5.4 Using the newly created group object as the Local Network on the IPsec tunnel

Modifying the Config Mode pool

We have so far modified the IPsec tunnel to restrict the VPN clients to gain access to only 3 specific networks. But we also must inform the connecting VPN clients of this, otherwise the VPN clients will attempt to surf the Internet through the tunnel and that is something we do not allow nor want them to do. If the client attempts to connect, the connection attempt will fail as well as cOS Core will not allow the client to use a larger network than that configured.

To accomplish this we modify the Config Mode pool object we created in the previous recipe to look as in figure 2.5.5.

Optionals

1 DNS:

NBNS/WINS:

DHCP:

2 Subnets:

Prefixes:

Figure 2.5.5 Add the newly created group as Subnets on the Config Mode Pool object

By adding the newly created group object as the Subnets (**2**) we will inform any client that connects, what kind of network(s) that can be found beyond the IPsec tunnel.



Important

This depends heavily on the feature set of the IPsec client whether it supports using multiple subnets or not. An alternative if it does this is not required, would be to configure a bigger local network on the IPsec tunnel such as 192.168.0.0/16 or 192.168.100.0/19 that contains all 3 networks, assuming of course it does not cause any conflicts with an existing network.

In this scenario, the DNS parameter (1) is optional, it may or may not be needed. It will be up to the administrator to decide whether a DNS server should be provided to the client or not. We cannot use our previously defined DNS server, as the IP address to that DNS server requires access to a network we no longer allow (true Internet access basically).

It might be useful to provide the IP to an internal DNS server for easier access to locally named resources, but it is optional.

Modifying the IP policy rule(s).

The last thing we need to do is to modify our IP Policies to incorporate the new requirements. As we do not want our users to surf to the Internet through the IPsec tunnel we remove the IP policy rule that allows Internet access. This rule would not be possible to trigger in any event, because the IPsec engine would not allow it

The only IP policy we need is a rule that allows access from the IPsec interface to the internal networks as shown in figure 2.5.6.

# ▲	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	▶ Roaming_To_Lan	✓	📶 Roaming_IPsec	📶 CFG_Pool	📶 any	📶 Local_Net_Group	📶 all_services

Figure 2.5.6 The IP policy needed for access to the internal network group from the IPsec tunnel interface

We set the destination interface as "**Any**" as the networks we want to reach are all behind different interfaces. We also set the destination network to be the network group we created earlier in order to give our clients access to all three networks.

Additionally, there are some alternatives on how this can be configured.

1. Use “**Any**” as destination interface then use the Network group as destination network (above example).
2. Create an interface group of Lan, DMZ and Admin interfaces and use the interface group as destination interface and the network group as destination network.
3. Create three different IP policies, one for each destination interface and network.

Which method to use, is to the decision of the administrator. All three methods have their advantages/disadvantages.

Requirement check

This concludes this recipe. As a last step we will look through the initial requirements to see if they can now be met. We will go through them one by one.

- Connected clients should be able to reach all the internal networks as shown in figure 2.5.1.
 - We modified our IP Policies, Config Mode Pool and IPsec interface configuration to allow connections from the clients towards all the desired internal networks.

Status: Fulfilled.

- Connected clients should **not** be able to reach the internet through the VPN connection. They should use their normal ISP connection when surfing the internet.
 - We limited the amount of networks that should be reachable through the tunnel by modifying the subnets on the Config Mode Pool and the local networks on the IPsec tunnel interface. This means that the clients will have to use their normal ISP connection for any traffic that is not directed towards one of the specific networks we have defined.

Status: Fulfilled.

Questions & Answers:

Question: What about the Config Mode pool, it currently contains only IP addresses from the first network?

Answer: This is unfortunately unavoidable, the IP pool object used on the Config Mode pool can be set to be a range of multiple networks but the IPsec engine would hand out the IP address one by one in sequence and not depending on the network the user is attempting to access. The easiest solution is to configure the IP pool to be in a range that is **not** a part of any of the destination networks. Then we also avoid any potential problems with ARP (as mentioned in the previous recipe).

Recipe 2.6. Accessing satellite offices through central HQ using VPN

Objective

A very common Lan2Lan tunnel scenario is when a company has a central HQ together with a couple of satellite offices. The satellite offices should not only be able to communicate with the HQ network but should be able to connect and communicate with the other satellite offices networks.

The objective of this recipe is to configure IPsec tunnels from satellite offices to a central HQ site as shown in figure 2.6.1, with the added option that the satellite offices should be able to communicate with each other as well using the tunnel towards HQ.

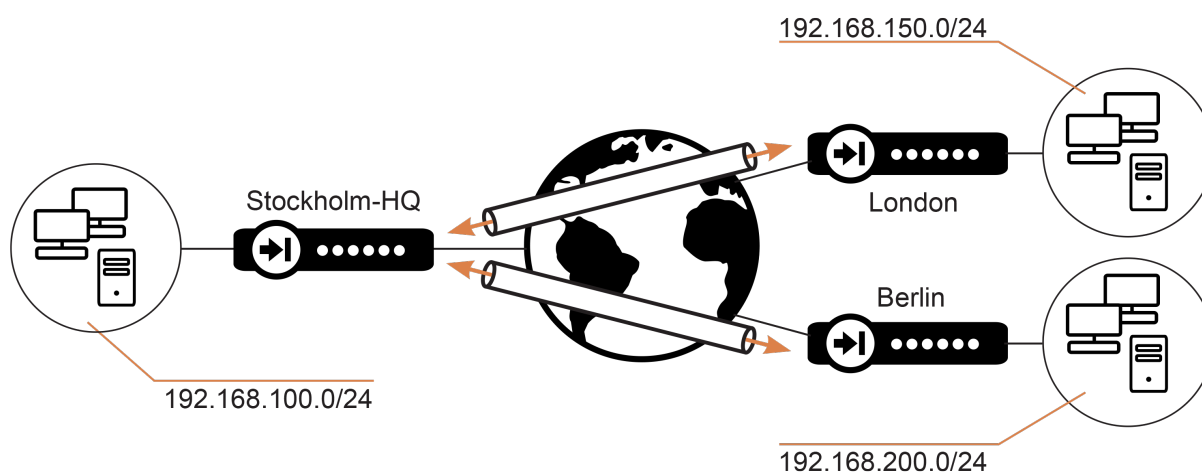


Figure 2.6.1 A Lan2Lan tunnel scenario where the satellite offices access each other using the IPsec tunnel to HQ

This means that there will be no Lan2Lan tunnel connected directly between the satellite offices. We will use the tunnel towards HQ for that. So, if for instance the London office wants to communicate with the Berlin office, the traffic would first go from London to HQ and then from HQ to Berlin. Intentionally we use HQ as a stopping point to reach the target office as shown in figure 2.6.2.

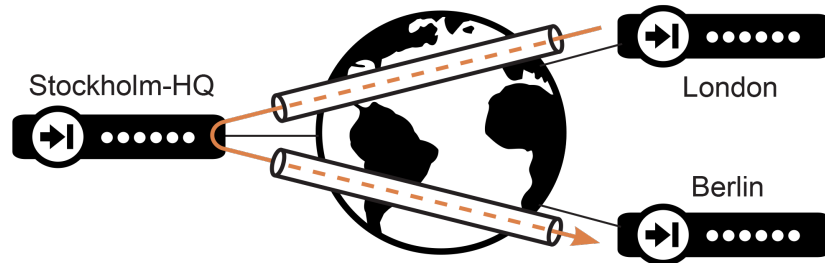


Figure 2.6.2 An example of the traffic flow when a connection is initiated from London to Berlin

Detailed discussion

Requirements

First we have a look at what kind of requirements we would like to implement, for this scenario we have the following requirements:

- HQ should be able to reach both the Berlin and London networks
- Berlin should be able to reach both London and HQ networks
- London should be able to reach both Berlin and HQ networks
- Public Internet access from each satellite office should **not** go through the VPN tunnel.

For this recipe, we assume that you are already familiar with setting up a standard Lan to Lan tunnel (if not please see *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*). We will not go into specific IPsec tunnel settings but rather go almost immediately into the tunnel network configuration.

This type of scenario can quickly become complex, having a small network schematic (sketch) available before starting to configure is recommended in order to avoid a potential network mix-up.

Initial setup

On each Firewall we have the following objects created:

```
Stockholm-Net  Address=192.168.100.0/24
Stockholm-IP   Address=203.0.113.100
London-Net     Address=192.168.150.0/24
London-IP     Address=198.51.100.150
Berlin-Net    Address=192.168.200.0/24
Berlin-IP     Address=192.0.2.200
```



Note

The single-IP addresses are just documentation examples.

We will be using Pre-shared keys as authentication for the Lan2Lan IPsec tunnels.

IPsec tunnel configuration, London

For the IPsec tunnel in London we want it to be able to access the networks in Stockholm and Berlin. The difficult part here is that the London Firewall will not be aware that the Berlin network is not actually behind the Stockholm-HQ Firewall. Instead, the London firewall will believe that the Berlin network is behind Stockholm-HQ.

London Local Network:

```
London-Net      192.168.150.0/24
```

The networks that London wants to be able to reach beyond the IPsec tunnel to Stockholm-HQ:

```
Stockholm-Net  192.168.100.0/24
Berlin-Net     192.168.200.0/24
```

To illustrate this further in figure 2.6.3, we show how the London firewall will view the network configuration.



Figure 2.6.3 The tunnel network configuration from the London firewall's perspective

The tunnel network configuration from the London firewall's perspective.

Since we want to reach two networks beyond the IPsec tunnel we create a network group consisting of Stockholm-Net and Berlin-Net, we call this group LondonHQ-Net as shown in figure 2.6.4.

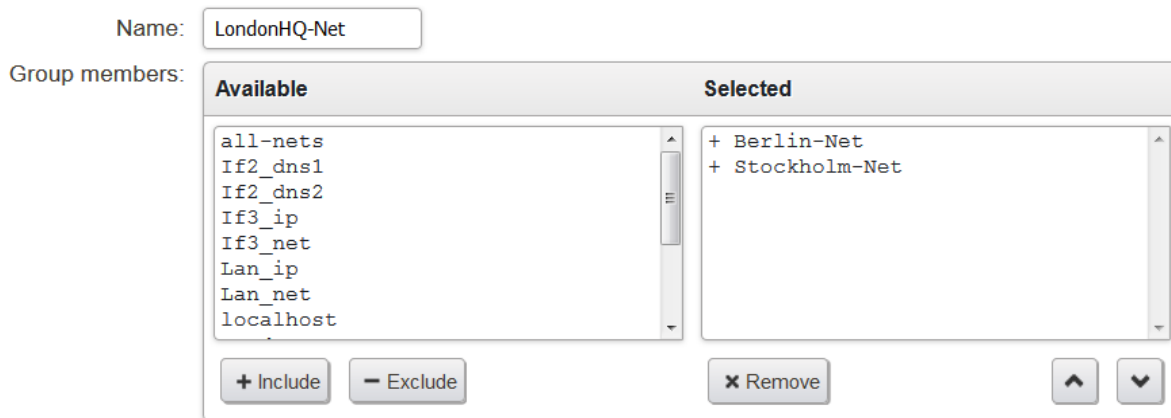


Figure 2.6.4 The network group for the network beyond the IPsec tunnel used on the London Firewall

Naming the objects can easily get confusing here, if we call the group object "HQ-Net" only and then use the same name for the network group object on the Berlin Firewall or at Stockholm-HQ, it can cause mistakes as they have the same name but contain different networks.

By calling the object "LondonHQ-Net" we give a hint that this object is for use on the London Firewall and contains the remote networks located behind the tunnel towards HQ. It will, however, be up to the administrators to name the object in a way they feel is working for their environment and network structure.

On our London to Stockholm-HQ IPsec tunnel, we use our newly created network group as the Remote Network on the IPsec tunnel as shown in figure 2.6.5.

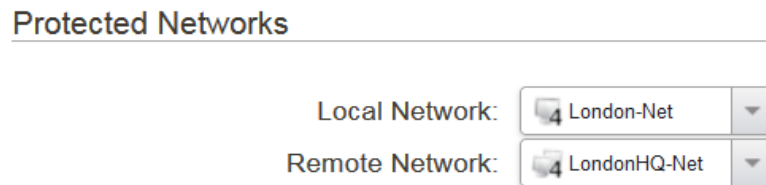


Figure 2.6.5 Using the network group on the IPsec tunnel from London to Stockholm-HQ

IPsec tunnel configuration, Berlin

For the IPsec tunnel in Berlin we want it to use a similar setup as in London. The difference in Berlin is that we want to access the networks in Stockholm and London. The Berlin Firewall will not be aware that the London network is **not** behind the Stockholm-HQ Firewall. Instead, it will believe that the London network is behind Stockholm-HQ.

Berlin Local Network:

Berlin-Net 192.168.200.0/24

The networks that Berlin want to be able to reach beyond the IPsec tunnel to Stockholm-HQ:

Stockholm-Net 192.168.100.0/24

London-Net 192.168.150.0/24

Similar to the London firewall we illustrate this further, in figure 2.6.6, by showing how the Berlin firewall will view the network configuration.

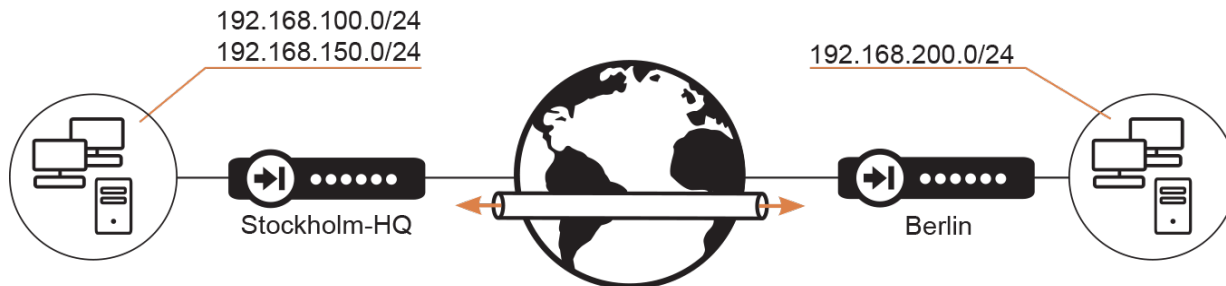


Figure 2.6.6 The tunnel network configuration from the Berlin firewall's perspective

As we want to reach two networks beyond the IPsec tunnel we create a network group consisting of Stockholm-Net and London-Net, we call this group BerlinHQ-Net as shown in figure 2.6.7.

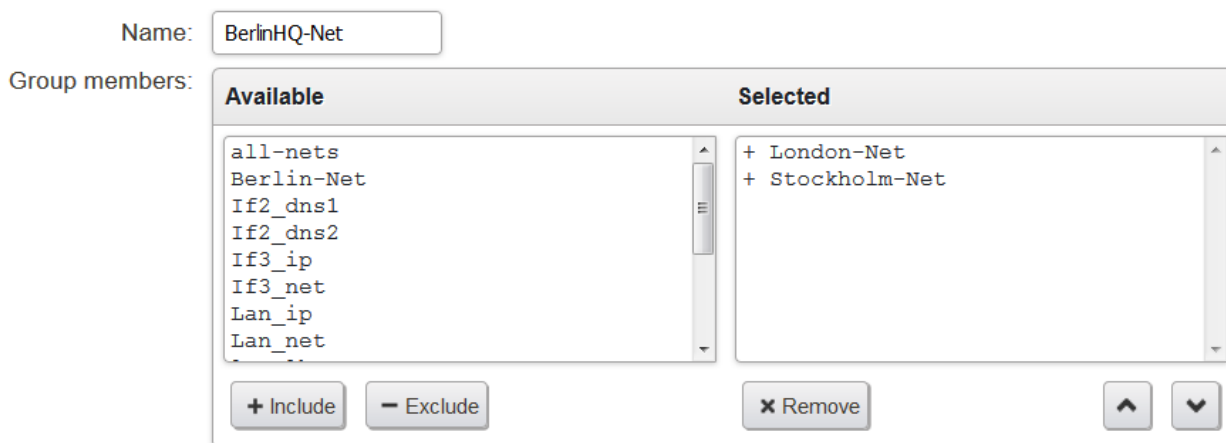


Figure 2.6.7 The network group for the network beyond the IPsec tunnel used on the Berlin Firewall

Lastly on our Berlin to Stockholm-HQ IPsec tunnel, we then use our newly created network group as the Remote Network on the IPsec tunnel as shown in figure 2.6.8.

Protected Networks

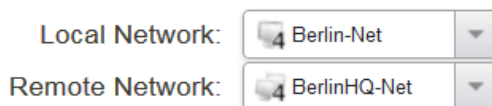


Figure 2.6.8 Using the network group on the IPsec tunnel from Berlin to Stockholm-HQ

IPsec tunnel configuration, Stockholm – HQ

Now we are coming to the IPsec network configuration on the Stockholm-HQ site, and here is where things start to get a little more complex and easy to make mistakes. We want to allow the following communication combinations:

```
Stockholm->London
Stockholm->Berlin
London->Stockholm
London->Stockholm->Berlin
Berlin->Stockholm
Berlin->Stockholm->London
```

We will start by configuring the tunnels and then go into more details about what it means. The first step is to create two new object groups in the address book as shown in figure 2.6.9.

HQ_London_Net	Stockholm-Net, London-Net
HQ_Berlin_Net	Stockholm-Net, Berlin-Net

Figure 2.6.9 The two network groups created on Stockholm-HQ

These two groups consists of the two network combinations we need to use on our two IPsec tunnels, one on the tunnel to London and the other on the Berlin tunnel.

On the tunnel towards London we use the HQ_Berlin_Net object as our local network as shown in figure 2.6.10.

Protected Networks

Local Network: HQ_Berlin_Net

Remote Network: London-Net

IKE Peer

Remote Endpoint: London-IP

Figure 2.6.10 Using the HQ_Berlin_Net group as local network on the IPsec tunnel towards London

The remote network will be the network we want to reach behind the London tunnel.

On the tunnel towards Berlin we use the HQ_London_Net object as our local network as shown in figure 2.6.11.

The screenshot shows a configuration interface for IPsec. It is divided into two main sections: "Protected Networks" and "IKE Peer".

- Protected Networks:** This section contains two dropdown menus. The first is labeled "Local Network:" and is set to "HQ_London_Net". The second is labeled "Remote Network:" and is set to "Berlin-Net".
- IKE Peer:** This section contains one dropdown menu labeled "Remote Endpoint:" which is set to "Berlin-IP".

Figure 2.6.11 Using the HQ_London_Net group as local network on the IPsec tunnel towards Berlin

Now that both IPsec tunnels are configured, we have all that we need in order to connect the network in London to Berlin (and vice versa) using the existing tunnel towards HQ.

Further explaining the scenario

The reason why we must use a group of two networks as the Local Network on HQ (or as remote networks in London/Berlin) is based on how the tunnel is configured in London and Berlin. If we take the London firewall as an example:

In order for London to reach Berlin we must first tell the Firewall in London that beyond the IPsec tunnel there are two networks, the network in Stockholm and the network in Berlin. The London Firewall does not know whether the Berlin network is past another tunnel or not, the only thing it knows is that there are two networks somewhere on the other side of the IPsec tunnel.

The Stockholm-HQ Firewall must also be configured to take into account that e.g. London will attempt to access two networks, the Stockholm-HQ's own internal network but also a network that is located beyond the Berlin tunnel.

Therefore on Stockholm-HQ's tunnel towards London, the Local Network must correspond to the London tunnel towards Stockholm's Remote Network.

Confusing? That is one of the primary reasons why it is highly recommended to create a detailed network schematic with all the network details in order to avoid mistakes in the IPsec network definitions. Being able to double-check towards a network schematic when configuring the tunnels, routes etc. can be invaluable in these scenarios.

To further clarify the Local and Remote network setting on the IPsec tunnel we can think of them as network definitions. We ask ourselves the following questions:

Local network

- Which local network(s) do we want the other side of the tunnel to access?

Remote Network

- Which network(s) on the other side of the IPsec tunnel do we want our local network to access?

To further explain the connection between Local and Remote network for the Stockholm-London tunnel, see the example in figure 2.6.12.

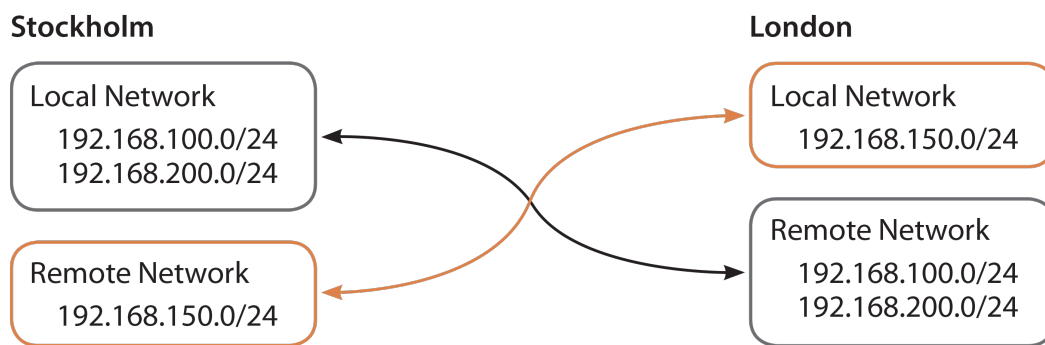


Figure 2.6.12 The correlation between Local and Remote network on the Stockholm <-> London IPsec tunnel

Verifying routing configuration

By default the IPsec tunnel interfaces on all sides will automatically create a route in the <main> routing table for the Remote Network. The route will tell cOS Core behind which interface the network can be found. If we look at the routing for the tunnels on all our sites it should display as follows:

Stockholm-HQ:

<u>Network</u>	<u>Interface</u>	<u>Metric</u>
192.168.150.0/24	London_Tunnel	90
192.168.200.0/24	Berlin_Tunnel	90

London:

<u>Network</u>	<u>Interface</u>	<u>Metric</u>
192.168.100.0/24	Stockholm_Tunnel	90
192.168.200.0/24	Stockholm_Tunnel	90

Berlin:

<u>Network</u>	<u>Interface</u>	<u>Metric</u>
192.168.100.0/24	Stockholm_Tunnel	90
192.168.150.0/24	Stockholm_Tunnel	90

The routing output can be examined in either the WebUI under Network->Routing->Routing Tables->Main or connect to the CLI and use the command "Routes".

IP Policy configuration, London

Now we come to the last part of the configuration. So far we have created the IPsec interfaces, defined the various networks and checked that the routing looks ok. The last part is to create the IP policies needed to allow the traffic to/from our various offices and IPsec interfaces.

We start with the London configuration. We want to be able to connect to/from Stockholm and Berlin but also the other way around by allowing Stockholm and Berlin to be able to access the internal network in London. For this we only need two rules as shown in figure 2.6.13.

# ▲	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	▶ London_To_Stockholm	✓	Lan	Lan_net	Stockholm_Tunnel	LondonHQ-Net	all_services
2	▶ Stockholm_To_London	✓	Stockholm_Tunnel	LondonHQ-Net	any	Lan_net	all_services

Figure 2.6.13 Configuring the needed IP policy's to allow traffic to/from the IPsec tunnel on the London Firewall

The first rule allows traffic from the internal Lan network and interface towards the two networks located behind the IPsec tunnel (Stockholm and Berlin nets). We use the network group object that was created in earlier steps for easier configuration.

The second rule allows traffic from the IPsec tunnel interface, specifically from the Stockholm and Berlin offices as we again use the network group object as the source network, we use any as the destination interface as we would like to be able to reach the Core interface on Lan as well for e.g. ICMP pings.

We use all-services mainly for testing purposes but it is also very common to allow everything between "internal" networks as they are considered trusted. It will be up to the administrator to decide what kind of restrictions that should be in place.

IP Policy configuration, Berlin

The Berlin configuration is almost identical do the London configuration with the single difference that there is an alternative network group object used. The network Berlin wants to access is the Stockholm-HQ and the London networks. We create two rules to allow the traffic to/from the IPsec tunnel as shown in figure 2.6.14.

# ▲	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	▶ Berlin_To_Stockholm	✓	Lan	Lan_net	Stockholm_Tunnel	BerlinHQ-Net	all_services
2	▶ Stockholm_To_Berlin	✓	Stockholm_Tunnel	BerlinHQ-Net	any	Lan_net	all_services

Figure 2.6.14 Configuring the needed IP policy's to allow traffic to/from the IPsec tunnel on the Berlin Firewall

The "BerlinHQ-Net" object group consists of the network objects for Stockholm-HQ and London.

IP Policy configuration, Stockholm – HQ

We now come to the IP policy configuration on the Stockholm-HQ Firewall. This one will be slightly different as it is the central node that will act as a kind of bridge that allows London and Berlin to talk to each other.

To accomplish this we need a total of four rules as shown in figure 2.6.15.

# ▲	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	▶ Stockholm_To_London	✓	Lan	Lan_net	London_Tunnel	London-Net	all_services
2	▶ Stockholm_To_Berlin	✓	Lan	Lan_net	Berlin_Tunnel	Berlin-Net	all_services
3	▶ London_To_HQBerlin	✓	London_Tunnel	London-Net	any	HQ_Berlin_Net	all_services
4	▶ Berlin_To_HQLondon	✓	Berlin_Tunnel	Berlin-Net	any	HQ_London_Net	all_services

Figure 2.6.15 Configuring the needed IP policies to allow traffic from the offices to talk to HQ and also their office colleagues

Before we go into details about the various rules lets first repeat what the destination network groups consists of, recommend having a look again at figure 2.4.1 for easier understanding.

```
HQ_Berlin_Net:
    192.168.100.0/24 (Stockholm-Net)
    192.168.200.0/24 (Berlin-Net)
```

```
HQ_London_Net:
    192.168.100.0/24 (Stockholm-Net)
    192.168.150.0/24 (London-Net)
```

Rule 1: This rule allows the local network in Stockholm-HQ to talk to the network in London.

Rule 2: This rule allows the local network in Stockholm-HQ to talk to the network in Berlin.

Rule 3: This rule allows the local network in London to talk to the network in both Stockholm-HQ and Berlin.

Rule 4: This rule allows the local network in Berlin to talk to the network in both Stockholm-HQ and London.

An alternative configuration for rule 3 and 4 would be to create two rules instead of one rule and instead of using a network group as destination network use the /24 network objects.

This means that there would be a total of 6 rules instead of 4. Both versions work just fine, it will be up to the administrator to decide which one to use.

We are not using any NAT (network address translation) on our rules as there is no need to hide or mask the source IP between the various networks.

Testing the connection using ping simulations

One of the most powerful troubleshooting tools we have in cOS Core is the ability to simulate and test rules using the “ping” command in the CLI. To verify that our current setup is working we can use the ping simulation to test the system to verify that rules, routing and that the correct interface(s) are triggering the way we want.

For a more in-depth explanation and examples of ping simulations please see Appendix-A.

Let us look at two examples of a ping simulation syntax and output that we use on Stockholm-HQ to verify that the rules and routing are working as expected.

First ping simulation example – Simulate that traffic arrives from the London tunnel with a destination in the Berlin network range.

Syntax:

```
ping 192.168.200.50 -srcif=London_Tunnel -srcip=192.168.150.50 -verbose
```

Output:

```
Rule and routing information for ping:
    allowed by rule "London_To_HQBerlin"
```

```
Sending 1 4-byte ICMP ping to 192.168.200.50 from 192.168.150.50
sent via route "192.168.200.0/24 via Berlin_Tunnel, no gw" in PBR table
"main"
```

```
ICMP Reply from 192.168.200.50 seq=0 time=<10 ms TTL=255
```

Result comment:

The above ping simulation simulates that traffic arrives on Stockholm-HQ initiated from the London network (-srcip) and arrives from the London Tunnel interface (-srcif). The verbose flag provides us with greater details regarding which rule that triggers and the route that it uses (see highlighted areas).

The source IP in the test can be anything as long as it's part of the London network. The destination host (192.168.200.50) is also an active machine so we got a reply, but sometimes it may be enough to simply test the rules without expecting a reply.

Second ping simulation example – Simulate that traffic arriving from the Berlin tunnel with a destination in the London network range.

Syntax:

```
ping 192.168.150.77 -srcif=Berlin_Tunnel -srcip=192.168.200.125 -verbose
```

Output:

```
Rule and routing information for ping:  
    allowed by rule "Berlin_To_HQLondon"
```

```
Sending 1 4-byte ICMP ping to 192.168.150.77 from 192.168.200.125  
sent via route "192.168.150.0/24 via London_Tunnel, no gw" in PBR table  
"main"
```

```
Ping Results:  Sent: 1, Received:0, Loss: 100%
```

Result comment:

The above ping simulation simulates that traffic arrives on Stockholm-HQ initiated from the Berlin network (-srcip) and arrives from the Berlin Tunnel interface (-srcif). Even though the target host we try to reach (192.168.150.77) behind the London tunnel does not reply, it still provides us with lots of details such as which IP policy that triggers and the destination route. Based on the above output everything looks good. If the host is supposed to reply the problem could be that the IPsec tunnel does not establish or that there is some problem with the network in London. Common causes for such a problem include broken switches, not allowed by the local firewall (e.g. the Windows firewall) or that the machine is simply not powered on.

The idea is to get clues where we need to look for troubleshooting potential problems, and here the ping simulation can be an invaluable tool.

Requirement check

Looking at the requirement list at the start of the recipe we can now do a check whether our requirements are fulfilled:

- HQ should be able to reach both the Berlin and London networks
 - We have setup tunnels to both London and Berlin and we allow the necessary networks to communicate between the two offices.

Status: Fulfilled.
- Berlin should be able to reach both London and HQ networks
 - We have configured the needed rules, routing and tunnels on Stockholm-HQ and London to allow the communication from Berlin.

Status: Fulfilled.
- London should be able to reach both Berlin and HQ networks
 - We have configured the needed rules, routing and tunnels on Stockholm-HQ and Berlin to allow the communication from London.

Status: Fulfilled.
- Public Internet access from each satellite office should not go through the VPN tunnel.
 - The IP Policies configured on all sides (Stockholm-HQ, London and Berlin) only allow traffic to/from the specific networks. There is no IP policy that allows access to all-nets (all IP addresses) through the IPsec tunnel(s). The IPsec tunnels are also configured only with specific networks so even if someone would attempt to send a packet towards a network outside the Local or Remote Network definitions on the various IPsec tunnels, the packet would automatically be dropped by the IPsec engine, the rules and routing. Basically 3 very large independent barriers of verification.

Status: Fulfilled.



Note

We will describe a scenario configuring internet access through the IPsec tunnel in Recipe 2.9. Making public internet access go through IPsec tunnel towards HQ.

Alternative IPsec tunnel network definition using all-nets (advanced)

Question: In this recipe we configure IPsec tunnels that use a group of networks as either their Local or Remote network definition. As the Local and Remote networks are only definitions of what kind of network(s) that will be allowed to traverse the tunnel, why not allow all networks (using the all-nets object) for both local and remote network?

Answer: Using all-nets as local or remote network is required in some scenarios such as when you want users to be able to surf to the Internet through the tunnel (see *Recipe 2.9. Making public internet access go through IPsec tunnel towards HQ*), but for most scenarios this is not needed. There are however several advantages of using all-nets as the network but also some drawbacks. To list some of the advantages and disadvantages:

Advantages of using all-nets as local and remote network:

1. Only one IPsec security association (SA) needs to be negotiated between the two Endpoints. Very advantageous if more than 3 networks are used in either Local or Remote network. Highly recommended.
2. Easier to route additional networks through the tunnel as no change is needed on the IPsec tunnel on either side
3. Tunnel Monitor would be able to trigger and affect all networks instead of just the one monitored in case of problems.
4. Less use of the VPN license parameter.

Disadvantages of using all-nets as local and remote network:

1. All networks will be allowed through the tunnel. 1 out of 3 checks if traffic should be allowed is not performed, less secure. The 3 checks are "IPsec Network definitions", "Routing" and "Rules".
2. Configuring the use of all-nets as Remote Network can cause network outage if not careful. If the checkbox for adding automatic route is not removed (depending on scenario) it could cause the Internet route to point to the IPsec tunnel instead of the Internet Service Provider.
3. Configuring the use of all-nets as Local Network can cause problems for traffic that needs to be NAT'ed into the tunnel or traffic that is to be initiated from cOS Core itself (such as logs or the Tunnel Monitor). An IP address must be manually configured on the tunnel otherwise the IP sender will be 127.0.0.1 as cOS Core will be unable to determine which source IP it should use.

Recipe 2.7. Using Certificates as authentication on a Lan2Lan tunnel

Objective

The objective of this recipe is to configure a Lan2Lan tunnel but instead of using Pre-shared keys as the authentication we will be using Certificates. We will first go through what Certificates are and some of the mechanics of using Certificates as an authentication method, then setup a Lan2Lan tunnel using first self-signed certificates and then with CA (Certificate Authority) signed.

Detailed Discussion

In all our IPsec recipes so far we have been using pre-shared keys as the authentication method. Pre-shared keys refer to a method where two tunnel endpoints use a pre-defined key that can be either a passphrase or a hexadecimal key. It will be to the decision of the administrator to determine what this key should be, meaning that the key may be as simple as just having the letter "AAAAAA" in it, which would not be a very secure authentication key.



Note

The process of using PSK is much more complex than the above example, it is simply not sending the key in plaintext but uses an algorithm with random data and hashes to make sure the key is identical between two endpoints. The point is that the simpler the key is, the quicker it will be to break it.

Certificates use a different approach to the authentication by adding whole new layers of authentication to the process. Certificates used by IPsec can also be referred to as X.509 (more information about X.509 further down). A certificate consists of the following parts:

- A public key.
- The "identity" of the user, such as name and user ID.
- Digital signatures that verify that the information enclosed in the Certificate has been verified by e.g. a CA (Certificate Authority).
- An expiration date

By combining the above information together, a Certificate is a public key with identification attached, coupled with a stamp of approval by a trusted party.

Introduction to Certificates

This part describes a lot of the basics of certificates. If already familiar with the concept we can skip this section and jump directly to the Requirements section later in this recipe.

In cryptography, X.509 is a standard that defines the format of public key certificates. X.509 certificates are used in many Internet protocols, including TLS/SSL, which is the basis for e.g. HTTPS, the secure protocol for browsing the web. They are also used in offline applications, like electronic signatures. An X.509 certificate contains a public key and an identity (a hostname, or an organization, or an individual).

Public key cryptography (or asymmetric cryptography) is any cryptographic system that uses pairs of keys: public keys which may be distributed widely, and private keys which are known only to the owner. This accomplishes two functions:

1. **Authentication:** Where the public key is used to verify that a holder of the paired private key sent the message.
2. **Encryption:** Where only the holder of the paired private key can decrypt the message encrypted with the public key.

This is illustrated in figure 2.7.1.

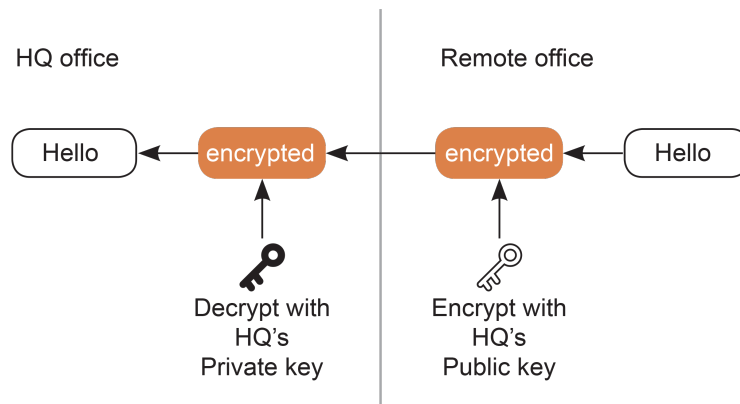


Figure 2.7.1 An example of how a public and private key can be used

Two ways of signing a certificate

A certificate can be signed using two methods, either signed by a Certificate Authority (CA) or self-signed. When a certificate is signed by a CA, or validated by another means, someone holding that certificate can rely on the public key it contains to establish secure communications with another party, or validate documents digitally signed by the corresponding private key.

A self-signed Certificate is a certificate that is issued by the individual using it, as an example cOS Core can create and use certificates signed by cOS Core itself. It is issued with software that the user has and controls. This can be good for testing environments but it also has some major drawbacks. Essentially, without going into too much detail, when we receive a certificate we expect it to be issued by a party we trust (could even be a third party). When we sign our own certificates we are vouching for our own identity.

As an example, self-signing a certificate is the same thing as handing a driver's license we made ourselves to a police officer that's pulling you over. It might have your real identifying information on it, but the police officer is not going to just take your word for it that it is valid.

It is possible to generate self-signed certificates directly from cOS Core, either from the WebUI or using the centralized management tool InControl.

Certificate Authority

A certificate authority (CA) is a trusted entity that issues certificates to other entities. The CA digitally signs all certificates it issues. A valid CA signature in a certificate verifies the identity of the certificate holder, and guarantees that the certificate has not been tampered with by any third party.

A CA is responsible for making sure that the information in every certificate it issues is correct. It also has to make sure that the identity of the certificate matches the identity of the certificate holder.

An example of a CA is Microsoft CA or XCA.

If we continue the police officer analogy. A CA signed certificate is the same thing as handing a driver's license that is issued by the government to a police officer that's pulling you over. It contains all the information the policy officer expects in a format he/she is familiar with. If needed the police officer can also query the government database to verify the driver license validity (See CRL below).

Certificate request

If we do not want to use Self-Signed certificates and instead use a CA server we first need to create a certificate request. This can be done in either cOS Core or using third party software. Once the certificate request has been created it needs to be exported (from e.g. cOS Core) and imported into the CA server that signs the Certificate request. The signed certificate can then be imported and used in cOS Core.

Certificate expiration date and validity time

A certificate is not valid forever. Each certificate contains values for two points in time between which the certificate is valid. When creating a certificate (or certificate request towards a CA) a validity time must be set in the certificate properties. A start time/date and an end time/date must be defined in the certificate. The start time determines when the certificate can start to be used and the end time determines when the certificate has expired and can no longer be used.



Note

It is very important that the time be correctly set on the unit generating, signing or using the certificate. If for instance the certificate is generated on the CA server to be valid in the past, the certificate will not work when installed on a unit with correct date set. Problems with certificates, for example in VPN tunnel establishment, can be due to an incorrect system date or time.

If a CA server is used, the CA server has the final word when it comes to setting the validity time. The requester may suggest a validity time but it will be up to the CA server to decide if it should grant the suggested time or override it. The latter is the most common as CA servers usually have a set validity time set in advance.

The Certificate Revocation List (CRL)

A Certificate Revocation List (CRL) contains a list of all certificates that have been canceled before their expiration date. They are normally held on an external server which is accessed to determine if the certificate is still valid. The CRL is downloaded from the CA server and cOS Core performs the validation of the certificate against the list. The ability to validate a user certificate in this way is a key reason why certificate security simplifies the administration of large user communities.

CRLs are published on servers that all certificate users can access, using either the LDAP or HTTP protocols. Revocation can happen for several reasons. One reason could be that the keys of the certificate have been compromised in some way, or perhaps that the owner of the certificate has lost the rights to authenticate using that certificate or perhaps because they have left the company. Whatever the reason, server CRLs can be updated to change the validity of one or many certificates.

CRL can be very useful if we want to deny a user the possibility to make a connection. By revoking the certificate the connecting client certificate is no longer trusted when cOS Core checks the CRL list.

Certificates will usually contain a CRL Distribution Point (CDP) field, which specifies one or more URLs with the path from which the relevant CRL can be downloaded. A CA usually updates its CRL at a given interval. The length of this interval depends on how the CA is configured. It can be between an hour and several days.

Certificate Chains

A CA can also issue certificates to other CAs. This can lead to a chain-like certificate hierarchy. Each certificate in the chain is signed by the CA of the certificate directly above it in the chain. The certificates between the root and host certificates are called Intermediate Certificates and consist physically of a single certificate file containing a public key.

The Certification Path refers to the path of certificates leading from one certificate to another. When verifying the validity of a host certificate, the entire path from the host certificate up to the

trusted root certificate anchor has to be available. For this reason, all intermediate certificates between the root certificate and the host certificate must be loaded into cOS Core.

As an example shown in figure 2.7.2 we have 3 CA servers, the root/anchor and two intermediate CA servers. The Berlin firewall is using certificates signed by the Intermediate-1 CA server and London/Stockholm is using certificates signed by the Intermediate-2 CA server.

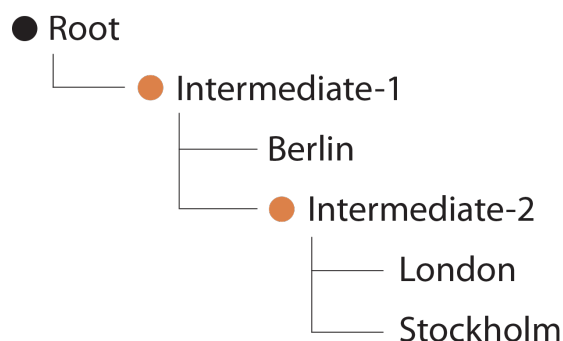


Figure 2.7.2 The tree structure for Intermediate CA servers

If we setup a tunnel between the various offices (as in *Recipe 2.8. Using ID list to separate IPsec tunnels that are behind dynamic IP*) but instead use Certificates the scenario would still work fine even if Berlin is signed by a different intermediate CA server. The reason it works is because the certificate chain is kept intact. London and Stockholm will trust the certificate Berlin is sending as Berlin is signed by a CA in London and Stockholm's certificate path/chain.

It is similar for the Berlin office. Berlin will trust the certificate London & Stockholm is sending as they are using a CA in the same certificate path/chain.



Note

In cOS Core IPsec VPN, the maximum length of a certificate chain is 4.

Root Certificates and Host Certificates

If a certificate is used for authentication, then it can be referred to as a Host Certificate but in cOS Core it is sometimes referred to as a Gateway Certificate. The certificate will consist physically of two files, a .cer file containing the public key and a .key file containing the private key. Both files must be loaded into cOS Core.

If the host certificate is CA signed, then the Root Certificate provided by the signing CA will also need to be loaded into cOS Core. This is just a single .cer file containing the public key of the CA. The private key of the CA is stored on the CA only. Self-signed certificates will not have a corresponding root certificate.

Trusting Certificates

When using certificates, cOS Core trusts anyone whose certificate is signed by a given CA. Before a certificate is accepted, the following steps are taken automatically each time to verify the validity of the certificate:

- Construct a certification path up to the trusted root CA.
- Verify the signatures of all certificates in the certification path.
- Fetch the CRL for each certificate to verify that none of the certificates have been revoked.

ID Lists

In addition to verifying the signatures of certificates, cOS Core can also use an ID list object when authenticating a connecting IPsec peer. An ID list contains all IDs that are allowed access through a specific IPsec tunnel. An ID is sent by the peer during the IKE negotiation and if a matching tunnel is found with this remote ID, authentication is then performed by checking to see if the certificate sent by the client contains that ID. We will discuss the use of ID lists as we progress in this recipe.

Certificates in cOS Core

A certificate in cOS Core is stored as a key-ring object of the type Certificate. There is always one certificate object already predefined in cOS Core which is the self-signed certificate used by cOS Core when accessing the WebUI for management using HTTPS. It is also used for SSL VPN.

This same self-signed certificate can also be used for e.g. IPsec tunnels, but it is recommended to create/use a different certificate just to avoid confusion.

Requirements

For this recipe we will only have one requirement and that is that we want to use Certificates instead of Pre-shared key as authentication.

Initial setup

We will base this recipe on *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*. We will assume that a Lan2Lan tunnel is already configured and is working between two endpoints (Stockholm and London) using Pre-shared keys as shown in figure 2.7.3.

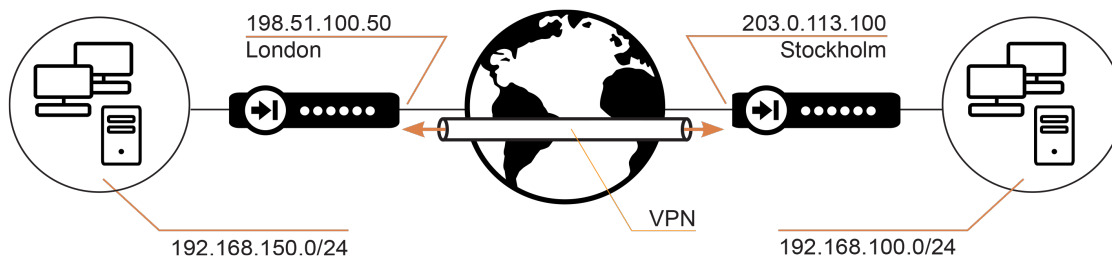


Figure 2.7.3 The initial setup, a Lan2Lan tunnel between London and Stockholm

IPsec tunnel configuration using self-signed certificates

We will start by keeping the scenario as simple as possible. We will create all the needed certificates directly in cOS Core to get our Lan2Lan tunnel up and running using certificates as authentication.

All the certificates in this recipe will be self-signed certificates, which in this case means that they will be signed by cOS Core itself. As previously mentioned by using self-signed certificate we vouch for our own identify. This can be useful for testing purposes but for live environments it is highly recommended to use a real CA server to sign the certificates.

Creating the certificates on the London Firewall

To add/create a new certificate we go to the key-ring on the London Firewall as shown in figure 2.7.4.

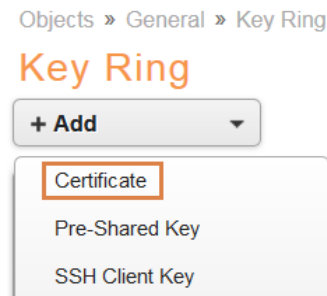


Figure 2.7.4 Certificates are handled in the key ring in the object section

Once the key ring type has been selected we are greeted with the options shown in figure 2.7.5. Please note that the options may vary depending on the choices.

Certificate

An X. 509 certificate is used to authenticate a VPN client or gateway when establishing an IPsec tunnel.

	Name:	<input type="text" value="London_Cert"/>
1	CRL checks:	<input type="text" value="Disabled"/>
2	Manual CRL dist. points:	<input type="text" value="(None)"/>

Certificate Information

3	Certificate type:	N/A
	Public Key Algorithm:	Unknown
	Certificate Authority:	No

Manage Certificate

4	Source:	<input type="text" value="Generate (RSA)"/>
5	Common Name:	<input type="text" value="London"/>
6	Certificate Bit Length:	<input type="text" value="2048"/>
7	Certification Authority:	<input type="checkbox"/>

Figure 2.7.5 The various options available when creating the certificate directly in the WebUI

In this recipe we want to use self-signed certificates created by cOS Core itself. We start with the CRL Checks **(1)** option. This option determines whether we should use CRL lists or not. Since we are using self-signed certificates we have no CRL server that stores the CRL list so we disable this option.

The same goes for the manual CRL dist. points **(2)** as CRL checks are not used. There is also no need to specify the location of any distribution points. This option can be used if the distribution point in the certificate properties needs to be overridden.

Details about the various certificate properties **(3)** are not available as we are currently in the process of generating the certificate. Once the certificate has been generated there will be various details available under the Certificate Information section if the object is opened again after creation.

The source **(4)** determines the source of the certificate. We have two options, either create our own (Self-signed) or upload a certificate created by someone else (such as a CA server or another cOS Core). Since we want to create our own certificate we have selected the option to Generate (RSA).

Common Name **(5)** is a way for the certificate to contain an identifier, a way to distinguish the certificate from other certificates. In our scenario we simply use the name of the Firewall as the common name, London. It is not the same as the initial name as that is the name for the certificate object in cOS Core, the Common Name is added into the certificate itself. If the certificate is later exported and looked at, the Common Name "London" will be visible.

Certificate Bit Length **(6)** determines the key size of the certificate. It is recommended to use the current maximum when creating self-signed certificates which is currently 2048 bit (support for higher bit lengths will be implemented in the future).



Note

Depending on the hardware, generating a large certificate key can, in some cases, take time and could cause a network interruption while the key is being generated.

Once the self-signed certificate is created it will be tagged as a type "Local" as shown in figure 2.7.6.


Name	Type	Type
 London_Cert	Certificate	Local

Figure 2.7.6 Once a self-signed certificate has been created directly in the WebUI it will be of the type "Local"

A certificate can be one of two types, "Local" and "Remote". A certificate tagged as Local means that it also contains the private key. A certificate tagged as Remote only has the public key.

As we created a self-signed certificate directly in cOS Core we also created a private key in the process, the certificate then automatically got tagged as type "Local".

Creating the certificates on the Stockholm Firewall

On the Stockholm Firewall we repeat the process, the only difference is we use a different name and use "Stockholm" as the common name in the certificate property.

Exporting & Importing the certificates from both Firewalls

Once the certificate has been created on both the Stockholm and London Firewalls we now have to export the self-signed certificate from London and import it to the Stockholm Firewall and vice versa from Stockholm to London. To export a certificate we open the certificate in question and click the download certificate button as shown on the Stockholm Firewall in figure 2.7.7.

Objects » General » Key Ring » Stockholm_Cert

Stockholm_Cert

Name:

CRL checks:

Manual CRL dist. points:

Certificate Information

Certificate type: Local

Public Key Algorithm: RSA

Certificate Authority: No

Version: v3

Keysize: 2048 bit

Signature Algorithm: rsa-pkcs1-sha1

Issuer: CN=Stockholm

Subject: CN=Stockholm

Valid from: 2016 Nov 22nd, 00:00:00 GMT

Valid to: 2036 Nov 22nd, 00:00:00 GMT

Figure 2.7.7 Downloading the certificate once it has been created on the Stockholm Firewall

We only want to download/export the certificate and not the key (the private key) as the private key is only used to decrypt data received from the remote endpoint and also the private key should never be needlessly exposed. When using CA signed certificate this is very important.

The reason why the private key should never be exposed is because it can then be used to generate additional certificates, and then the whole certificate trust chain breaks.

Once the certificate has been downloaded from both Firewalls we need to import them. The London certificate is imported to the Stockholm Firewall and the Stockholm Certificate is imported to the London Firewall. The process of uploading/importing a certificate is performed by adding a new certificate in the key-ring and then selecting Upload as shown on the London Firewall in figure 2.7.8.

Name:

CRL checks:

Manual CRL dist. points:

Manage Certificate

Source:

Certificate:

Private Key: No file selected.

Figure 2.7.8 Adding/uploading a previously created certificate

Once the certificate has been uploaded it will automatically be tagged as a Remote type certificate as it does not contain the private key, as shown in figure 2.7.9.


Name	Type	Type
 Stockholm_Cert	Certificate	Remote

Figure 2.7.9 A certificate without the private key will be tagged as type "Remote"

Changing the IPsec configuration on both Firewalls to use the new certificates

We now have everything we need to complete the change from pre-shared key to certificate on our Lan2Lan tunnel.

We open our previously configured PSK IPsec tunnel and go to the authentication tab as shown in figure 2.7.10.

The screenshot shows the configuration interface for an IPsec tunnel. It is divided into three numbered steps:

- 1 Authentication Method:** A dropdown menu is set to "Certificate".
- 2 Gateway certificate:** A dropdown menu is set to "Stockholm_Cert".
- 3 Root Certificate(s):** A window with two columns: "Available" and "Selected".

Available	Selected
HTTPSAdminCert	London_Cert
	Stockholm_Cert

 Below the lists are buttons for "+ Include" and "X Remove".

Figure 2.7.10 Changing from PSK to certificates provides us with some new options

First we need to change the Authentication Method (**1**) from Pre-shared key to Certificates. Once this is done we need to first specify which Gateway certificate (**2**) we should use on the Firewall. The Firewall in question in figure 2.5.10 is the Stockholm Firewall. When selecting the gateway certificate the requirement is that it must be a certificate of the type "Local". Which means it must also contain the private key. This is a must as the private key must be used when decrypting the data sent from the remote endpoint.

For the root certificates (**3**) we select both the London and the Stockholm certificates. We must add the Stockholm certificate in two places (both Gateway and root), the reason for that is because we are using self-signed certificates, this means that we will vouch for ourselves when it comes to certificate validation if/when the remote endpoint (London) sends encrypted data packets.

We now repeat the certificate selection process on the London firewall, except London will use the London_Cert (with its private key) as its gateway certificate.

All done, we can now establish the Lan2Lan tunnel using a certificate pair instead of using a pre-shared key.

Additional information, tunnel establishment process for self-signed certificates

To further explain how certificates work when establishing an IPsec tunnel with self-signed certificates let us make an example of the setup process. We want to initiate the IPsec tunnel from Stockholm to London.

1. Stockholm sends a tunnel negotiation request to London where Stockholm sends the public certificate of the Stockholm firewall (Stockholm's Gateway certificate without the private key) in the authentication phase.
2. London receives the request and tries to match the Stockholm certificate towards its certificate repository to verify whether the certificate can be trusted or not.
3. London finds the Stockholm certificate in its certificate repository (and therefore trusts it) and proceeds with the tunnel negotiation by sending its own Gateway certificates to Stockholm.
4. Stockholm receives the London certificate and performs a similar operation where it looks in its certificate repository to see if it can find a match.
5. Stockholm finds the London certificate in its certificate repository (Root certificates in this case) and the authentication phase is now concluded. If all goes well with the rest of the negotiation the tunnel will now be established.

Question: What about the private key? When will that be used?

Answer: For example, if we send an ICMP ping from a host behind Stockholm to a host behind London (in an already established IPsec tunnel), the public key of the London certificate will be used to encrypt the data.

cOS Core will use the London public key, as that is what has been negotiated with the remote endpoint (London) when the IPsec tunnel was established. The Stockholm Firewall will then know which certificate and public key it needs to use to encrypt the data.

When the London Firewall receives the data packet it will use its private key of the London certificate to decrypt the data.

Only with the private key can the data inside the tunnel be decrypted. That is why the Firewall that initiates the data transfer must use the public key of the remote endpoint to encrypt the data.

When the ICMP reply returns from the machine behind the London firewall, the London firewall will then encrypt the reply using the public key of the Stockholm Firewall. *Figure 2.7.1* is a good visual representation of this.

IPsec tunnel configuration using CA signed certificates

We have now gone through the basics of using certificates on a Lan2Lan tunnel. We will now continue to strengthen the security by using certificates that are signed by a CA server instead of using self-signed ones. This is the preferred method when using certificates as self-signed certificates should at best be used for testing purposes.

We will assume that a third party CA server is already configured as this book will not go into details on how to configure systems outside cOS Core.

In order to modify our current setup and use CA signed certificate between Stockholm and London we need two things on each side:

1. The CA server's root certificate
2. A certificate that has been signed by the CA server

Importing the CA server root certificate

The first thing we need to do is to export the root certificate from the CA server and then import it to each site (Stockholm and London). The certificate exported from the CA server will not contain the private key, as a CA server's private key is a secret that should be closely guarded, especially if the CA server in question is at the top of the certificate chain. Some companies even power down the root/top CA server and only power it up to generate e.g. a new intermediate certificate or to renew the lifetime of an existing certificate, then power down the server again.

Once the CA server's root certificate has been downloaded we need to import it to cOS Core. The procedure is the same as when importing the self-signed certificates. In the address book's keyring we add a new certificate and choose to upload a certificate as shown in figure 2.7.11.

Certificate

An X. 509 certificate is used to authenticate a VPN client or gateway when establishing an IPsec tunnel.

1	Name:	<input type="text" value="Anchor"/>
2	CRL checks:	<input type="text" value="Conditional"/> ▼
	Manual CRL dist. points:	<input type="text" value="(None)"/> ▼

Certificate Information

Certificate type: N/A
 Public Key Algorithm: Unknown
 Certificate Authority: No

Manage Certificate

3	Source:	<input type="text" value="Upload"/> ▼
4	Certificate:	<input type="button" value="Browse..."/> Anchor.cer
	Private Key:	<input type="button" value="Browse..."/> No file selected.

Figure 2.7.11 Uploading the CA server root certificate

For our root certificate we choose the name (1) "Anchor" for this object as we only have one CA server and it will be at the top of the certificate chain. But in case we want to add any intermediate certificate in the future it will be easier to keep track of which CA server is at the top if we name it Anchor. But it will be up to the administrator to choose the object name theme based on their own preference and requirements.

When using CA signed certificates it may be a good idea to start using CRL checks (2). There are three choices related to CRL check:

1. **Enforced** – This option means that the associated CRL must be checked before the certificate can be used for authentication. If the associated CRL cannot be retrieved, perhaps because the CA server is offline/unreachable, then the certificate will not be possible to validate and the authentication will fail.

2. **Conditional** – This option means that cOS Core will attempt to check the CRL but if it fails to fetch the CRL list due to the CA server being offline/unreachable it will continue/accept the authentication anyway as long as the negotiated certificate is valid and signed by the same CA.
3. **Disabled** – This option means that no CRL check will be performed.

The CRL path is available in the certificate properties itself and may look something like this:

```
URL=http://something.myCAserver.se/rev.crl
```

Assuming, of course, that the CRL is used at all, we will use the value “Conditional” in this recipe to avoid problems with the initial setup. When setting up a tunnel for the first time it is highly recommended to keep things simple and at a later stage add more advanced features such as enforcing the CRL list. If the tunnel stops working after the enforcement change we know immediately where the problem is.

When uploading a certificate we choose “upload” (3) as the source and point out the path to the CA server's root certificate (4). When the root certificate has been upgraded it will look as in figure 2.7.12.


Name	Type	Type
 Anchor	Certificate	Remote

Figure 2.7.12 A certificate without the private key will be tagged as type “Remote”

Generating the certificate request

Now we need to generate a certificate request and have the request signed by the CA server. Unfortunately it is not possible to generate the certificate request itself directly from the WebUI in cOS Core. The certificate request (and the private key) needs to be generated either from Clavister InControl, the CA server or another third party program with the ability to generate a certificate request. A certificate request from InControl is shown in figure 2.7.13.



Choose Certificate Type

Create self-signed certificate
Others need a copy of the public certificate file in order to verify it.

Create certificate request
Certificate requests are first passed to certificate authorities for signing. When signed, others only need a copy of the public CA cert to verify your certificate.

General Parameters

Name:

Comment:

Figure 2.7.13 Creating a certificate request using Clavister InControl

Once the certificate request has been created it needs to be exported to the CA server and then signed by the CA server's root certificate (the one we call Anchor in this example).

Important: When the certificate request is generated and is about to be exported, the private key must also be exported as this is needed on each installation in order to decrypt the packets.

Once all the certificates have been signed and relevant files exported, we end up with the following list of needed files:



Important

The process of using PSK is much more complex than the above example, it is simply not sending the key in plaintext but uses an algorithm with random data and hashes to make sure the key is identical between two endpoints. The point is that the simpler the key is, the quicker it will be to break it.

Once all the certificates have been signed and relevant files exported, we end up with the following list of needed files:

```
London.cer  
London.key  
Stockholm.cer  
Stockholm.key  
Anchor.cer
```



Note

There are many different types of encoding for the exported certificates, in our case we are using the export .cer format (Binary DER encoded). Other formats are supported as well such as .crt. Password protected certificates cannot be imported as there is no support in cOS Core to input a certificate password.

Importing the signed certificates & finalizing the tunnels

We now have all the required CA signed certificates and keys needed to establish our tunnel. We have previously imported the CA root (Anchor) certificate and now perform a similar certificate upload operation for our signed certificates as well.

On the London Firewall we import the London.cer, London.key (and Anchor.cer).

On the Stockholm Firewall we import the Stockholm.cer, Stockholm.key (and Anchor.cer).

We then go into the IPsec tunnel on each side and use our newly imported certificates on the IPsec tunnel as shown in figure 2.7.14.

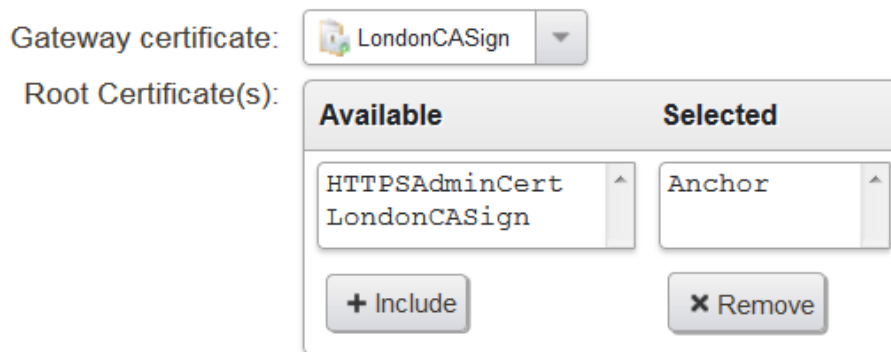


Figure 2.7.14 Selecting the CA signed certificate as the Gateway certificate and the CA root (Anchor) as root on the London Firewall tunnel

The Gateway certificate contains the CA signed certificate request and private key and as the Root certificate. We select our CA server's root certificate (the one we call Anchor).

We repeat the process on the Stockholm Firewall using the Stockholm.cer and key file.

Additional information, tunnel establishment process for CA signed certificates.

To further explain how certificates work when establishing an IPsec tunnel with CA-signed certificates let us make an example of the setup process. We want to initiate the IPsec tunnel from Stockholm to London.

1. Stockholm sends a tunnel negotiation request to London where Stockholm sends the public certificate of the Stockholm firewall (Stockholm's Gateway certificate without the private key) in the authentication phase.
2. London receives the request and tries to match the Stockholm certificate towards its certificate repository to verify whether the certificate can be trusted or not.
3. London sees that the Stockholm certificate is signed by the root certificate (Anchor) in its certificate repository (and therefore trusts it) and proceeds with the tunnel negotiation by sending its own public Gateway certificate, without the private key, to Stockholm.
4. Stockholm receives the London certificate and performs a similar operation where it looks in its certificate repository to see if it can find a match.
5. Stockholm sees that the London certificate is signed by the root certificate (Anchor) in its certificate repository (and therefore trusts it). The negotiation can proceed and the authentication phase is now concluded. If all goes well with the rest of the negotiation, the tunnel will now be established using CA signed certificates.

The private key is used in a similar way as when using Self-Signed certificates, see the Q&A in the "*Additional information, tunnel establishment process for self-signed certificates*" section for details.

Recipe 2.8. Using ID list to separate IPsec tunnels that are behind dynamic IP

Objective

The objective of this recipe is to configure two Lan2Lan tunnels using certificates where we do not know from which source IP the incoming negotiation will arrive. We will use ID lists to make sure that incoming tunnel negotiations are matching the IPsec tunnel we want, and achieve a way to separate incoming negotiations.

Detailed Discussion

A common scenario that frequently occurs involves a situation as shown in figure 2.8.1.

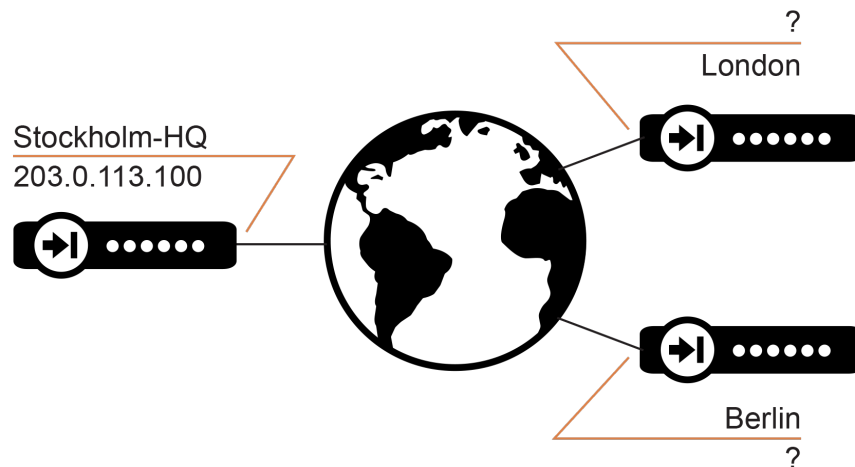


Figure 2.8.1 The Firewalls in London and Berlin are using dynamic IP, meaning we will be unable to determine what their exact IP addresses are.

In this scenario we have our 3 sites, Stockholm-HQ, Berlin and London. What is special with this scenario is that both the Berlin and London firewalls are behind dynamic IP (e.g. DHCP or PPPoE) so we do not know from which source IP they will connect as they could/will change their IP as much as once per day.

But we still want to be able to separate London and Stockholm on the HQ firewall by using two different IPsec tunnels and one way to accomplish this would be to use ID lists.

There is of course more than one solution to handle this particular scenario but we will solve it using certificates and ID lists.

The IPsec tunnel configuration on Stockholm-HQ

The IPsec tunnel configuration on the Stockholm-HQ Firewall is configured as shown in figure 2.8.2.









# ▲	Name	Local Net	Remote Net	Remote Endpoint	Local Endpoint	Auth
1	 Berlin_Tunnel	 StockholmNet	 BerlinNet	 all-nets		Certificate
2	 London_Tunnel	 StockholmNet	 LondonNet	 all-nets		Certificate

Figure 2.8.2 The IPsec tunnels currently configured on the Stockholm-HQ Firewall

As we can see the remote endpoint is configured to use all-nets. This is because we do not know from which IP the Berlin or London Firewalls will connect because they are behind a dynamic IP that may change on a daily basis.

The idea is to use ID lists to make sure that when the Berlin or London tunnel initiates a connection towards Stockholm-HQ, it will match the correct IPsec tunnel interface.



Note

As long as a tunnel has the remote endpoint "all-nets" (or none) it means that it is unable to initiate the tunnel, it can only accept incoming connections as it does not know where to connect.

ID lists, description

A cOS Core ID List object contains one or more ID objects. An ID object can be an IP, a DNS name, common name etc. An IPsec tunnel interface can then have its Remote ID property set to an ID list object. For a particular tunnel to be used by a particular remote endpoint, the following must be true:

- The ID sent by the remote endpoint must match one of the IDs in the ID list for the tunnel.
- The ID sent by the remote endpoint must also exist as one of the IDs in the certificate that the remote endpoint sends.

When the remote endpoint (e.g. Berlin) performs a connection attempt, cOS Core in Stockholm chooses the IPsec Tunnel object to use as follows:

- The connecting remote endpoint (Berlin) sends its ID to cOS Core (in Stockholm) in the IKE negotiation.
- cOS Core scans its list of IPsec Tunnel objects looking for a match for the remote endpoint.
- As an additional part of the matching process, cOS Core also checks the ID that the remote endpoint sends against the ID List of the tunnel. If it does not find an ID match, it continues searching through the IPsec Tunnel list. Any malformed IDs will be ignored and will also generate log message warnings.
- Once the matching tunnel is found, cOS Core then checks that the certificate that the remote endpoint sent also contains this same ID. If the certificate does, authentication is complete and the tunnel can be established. If the ID is not in the certificate, the incoming connection attempt is rejected.

Basically, ID lists is a way to make sure that the correct tunnel interface matches. If the first tunnel does not match cOS Core can continue to try match other tunnels in the IPsec tunnel list. ID lists are matched in phase-1 (IKE) of the tunnel negotiation.



Note

Tunnels with pre-shared key can also do this to an extent but it requires that the same pre-shared key is used on all involved tunnels, this method is not considered secure. Instead, certificates are the better alternative, as they offer more authentication strength and options to make sure the device sending the incoming connection attempt is the one that it claims it to be.

Choosing the certificate property to use as identifier

When we create a certificate request towards the CA server we have the option to input additional parameters into the certificate properties as a Subject Alternative Name (SAN). This is optional but can be used as an identifier in our ID list. For SAN we can use DNS names, IP or email addresses.

Our current certificate properties are as follows:

London certificate property:

```
Subject : CN=London, O=Support, C=UK  
DNS=Support
```

Stockholm certificate property:

```
Subject : CN=Stockholm, O=Sales, C=SE  
DNS=Sales
```

Berlin certificate property:

```
Subject : CN=Berlin, O=Marketing, C=DE  
DNS=Marketing
```

What we need here is something unique to use as identifier, in the above list we have a good match on either the Distinguished Name (Subject) or our special subject alternative DNS Name. We will use the DNS as the unique identifier in our ID list in order to keep things simple.



Note

The use of DNS names in the above mentioned certificates is just an identifier, they are not required to be resolved by a DNS server.

Creating the ID list on the Stockholm-HQ Firewall

An ID list is created in the cOS Core WebUI under Objects->VPN Objects->IKE ID lists. Once the ID list has been created we need to add an identifier/identity (ID) to the list as shown in figure 2.8.3.

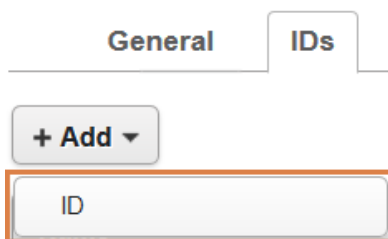


Figure 2.8.3 Adding an ID to an ID list

Once we add the ID we are greeted with all the various ID options as shown in figure 2.8.4.

Objects » VPN Objects » IKE ID Lists » London_ID » London

London

An ID is used to define parameters that are matched against the subject field in an X.509 certificate when establishing an IPsec tunnel.

1

Name: London

2

Type: DNS

IP Address:

3

Hostname: Support

Common name:

Organization name:

Organizational unit:

Country:

Locality name:

Email address:

Advanced

Property	Value

Figure 2.8.4 The properties of an ID in the ID list

This will be the ID used to make sure that when the London Firewall initiates the tunnel, it will match the correct IPsec tunnel interface on Stockholm-HQ. We select the type (1) DNS as we want to use the Subject Alternative Name as the unique identifier in the certificate properties. When London initiates the tunnel towards Stockholm-HQ it will send (in the negotiation) the Subject Alternative Name "Support", which will then match this ID list's DNS hostname (2).

The Advanced property (3) is for less common fields such as serial number that can exist as a certificate property. Then it can be added manually (custom field).

ID lists only need to be created and used on the Stockholm-HQ Firewall. The reason for this is because it is the instance acting as the tunnel terminator. The Stockholm-HQ Firewall also has a static IP making it ideal to act as the central node in this scenario as its public IP will not change.

We also create a second ID list for use on the Berlin tunnel with an ID that matches the unique DNS property in the Berlin certificate (Marketing). Once this is done we have two ID lists, one for London and one for Berlin.

Using the ID lists on the IPsec tunnels

Now we continue by assigning our previously created ID lists to our two tunnels. The only thing we need to do is to assign the correct ID list to the corresponding tunnel, as shown in figure 2.8.5 on the London tunnel in the Stockholm-HQ Firewall.

Authenticated Identities

1 Local ID:

2 Remote ID:

3 Enforce local ID:

Figure 2.8.5 Assigning the London ID list to the tunnel to/from London

Local ID (1) we leave blank as cOS Core will by default use the public IP of the Stockholm firewall as its Local ID unless we specify something here. We have no need to change or modify the Local ID in this scenario.

In Remote ID (2) we select London as this is the IPsec tunnel towards London. Or rather from London as the Stockholm-HQ Firewall will be unable to initiate the tunnel. The reason for this is because the remote endpoints on both our tunnels at Stockholm-HQ are using "All-nets" as its remote endpoint. The Stockholm firewall will be unable to determine which IP it needs to contact in order to establish the tunnel.

The Enforce Local ID (3) setting can be enabled so that when cOS Core is acting as the IPsec tunnel terminator (endpoint), the ID proposed by the initiator must match the Local ID value on the Stockholm firewall. We will not enforce Local ID in this scenario and will leave it unchecked.

Configuration complete, a tunnel match example

The configuration is now complete. If either London or Berlin initiates the tunnel towards HQ, cOS Core will only match it towards a specific tunnel if both the traffic selectors (Local/Remote network) and the ID list match. The ordering of the IPsec tunnel is not important in this scenario. We can have as many IPsec tunnels as we want all using "all-nets" as the remote endpoint and it can match each tunnel just fine. As long as we have defined which ID list to use on each IPsec tunnel in order to separate them.

To make a tunnel matching the example we currently have our two IPsec tunnels configured like this on Stockholm-HQ:

1. Berlin_Tunnel StockholmNet Berlin-Net All-nets ID=Marketing
2. London_Tunnel StockholmNet LondonNet All-nets ID=Support

Let us say that London will attempt to initiate the tunnel from its side towards Stockholm HQ, in order for this to match tunnel number two on Stockholm it needs to fulfill the following requirements:

- The Remote endpoint must match
- The certificate must be trusted
- The chosen identifier in the ID list must exist in the certificate properties
- The Local and Remote network combination must match

Using the above requirement list as a checklist let us see how the matching process goes for our London tunnel. The tunnel matching process is done from the top to the bottom of the list of IPsec interfaces, starting with tunnel number one, until a match is found (if any).

1. **Does the remote endpoint match?**

The remote endpoint is defined as all-nets, so yes the remote endpoint matches.

2. **Is the certificate trusted?**

No, it is not. The ID in the certificate contains "Support", it does not match the ID used in the first tunnel.

3. **Does the ID in the local ID list exist in the incoming certificate?**

No, it does not. The ID in the certificate contains "Support", it does not match the ID used in the first tunnel.

4. **Does the Local / remote network combination match?**

Partially, the Local network in Stockholm matches but not the remote network. This however means that the traffic selectors (network selectors) do not match.

cOS Core now knows that the initiating London tunnel does not match the first local IPsec interface and will continue to process through the list of interfaces to find a tunnel match. This is just an example. The selection process is faster than this as cOS Core will not necessary perform the tunnel match one at a time but rather match all parts of a particular phase towards all tunnels in parallel. The example shown below is a partial output from the CLI command "ike – snoop –match" when the London tunnel attempts to initiate.

```
Searching for a suitable tunnel to match traffic selectors
Tunnel Berlin_Tunnel      : The remote traffic selectors do not match.
Tunnel London_Tunnel      : Match.
```



Note

There are many CLI commands that can be used for troubleshooting and to get additional information. We will not go into any great details about CLI commands.

Questions & answers

Question: But if the remote endpoint is "all-nets", can only one side initiate the tunnel?

Answer: Yes, that is correct. That is one of the drawbacks of using "all-nets" as the remote endpoint. It is recommended to configure the tunnel monitor on the satellite offices to make sure they always try to keep the tunnel established at all times from their end.

Question: What about roaming IPsec clients? If I have multiple roaming tunnels all with all-nets as remote endpoint, can I use ID lists to separate e.g. 100 users for each tunnel?

Answer: Yes, this is one of the greatest strengths of using certificates with ID lists. We can configure multiple roaming tunnels and make sure that the correct user matches the correct tunnel. An ID list can contain hundreds if not thousands of entries to make sure the incoming negotiation matches the correct IPsec tunnel. It is a great method to be able to terminate an IPsec tunnel towards cOS Core for users belonging to different groups e.g. different companies in order to separate them using different IPsec interfaces.

Question: What about mixing Pre-shared key and Certificate tunnels? Could that cause problems?

Answer: It could cause some problems. But if we make sure that the tunnels that are using pre-shared key are placed above the certificate tunnels in the list of IPsec interfaces it should work fine. If the authentication method is something other than PSK, the IPsec engine can continue the matching process with tunnels that use certificates. Again, it is recommended to position the PSK tunnels above the certificates tunnels.

Question: If we have a lot of IPsec tunnels and all are using all-nets as the remote endpoint, can the process of matching the correct tunnel slow down the system and take up a lot of CPU resources?

Answer: It is a very difficult question to answer. Similar to IP policies the process of matching the correct tunnel must be performed in a specific order. It is similar to IP policies where it starts at the top and moves down in the IPsec interface list. The more IPsec interfaces configured, the more interfaces the IPsec engine must examine in order to find a match. However, the specific scenario of multiple all-nets remote endpoint tunnels would make very little difference compared to having a large amount of IPsec tunnels in general.

Recipe 2.9. Making public internet access go through IPsec tunnel towards HQ

Objectives

The objective of this recipe is to make all users behind a satellite office (London) go through the IPsec tunnel towards HQ (Stockholm) for their Internet access, as shown in figure 2.9.1.

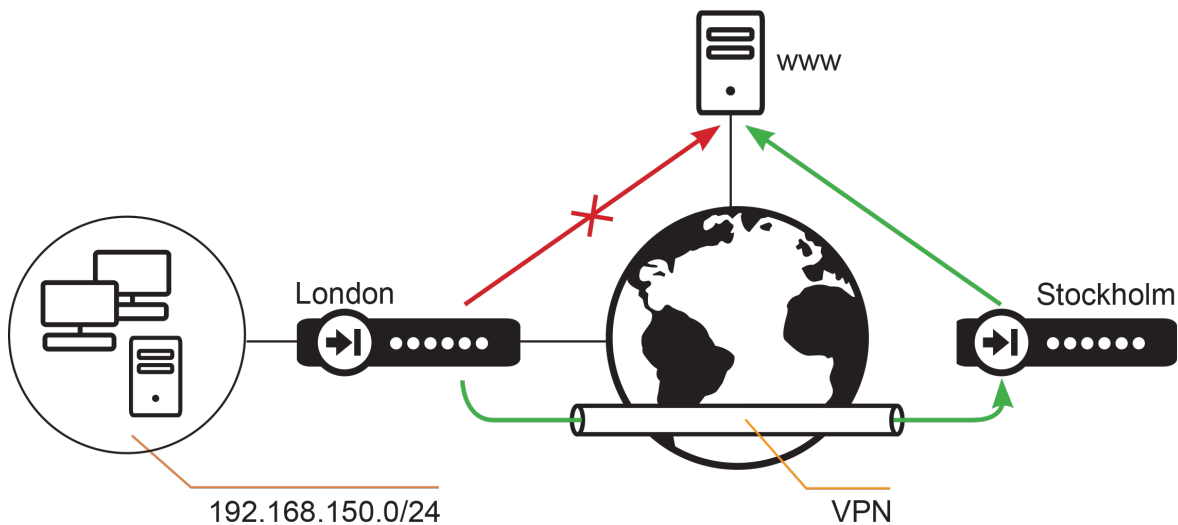


Figure 2.9.1 Forcing all users behind the London Firewall to go through the IPsec tunnel to Stockholm for their public Internet access

An example of when we want to do this is if we have various software services e.g. Anti-Virus, web proxies, and generally more powerful hardware available at HQ to better protect the users in London. By making users in London go through Stockholm-HQ for their Internet access we can re-use existing systems and also simplify the security policies by only having to configure them at Stockholm-HQ.

The drawback will be an increase of latency, dependency on the HQ for internet access and higher bandwidth usage at HQ. So there would need to be a comparison made of the pros and cons whether this particular system should be implemented or not. But for smaller satellite offices it offers great advantages in terms of simplified management once the system is in place.

Detailed Discussion

This recipe will use *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario* as a basis. We will assume that a standard Lan2Lan tunnel is currently configured as shown in figure 2.9.2.

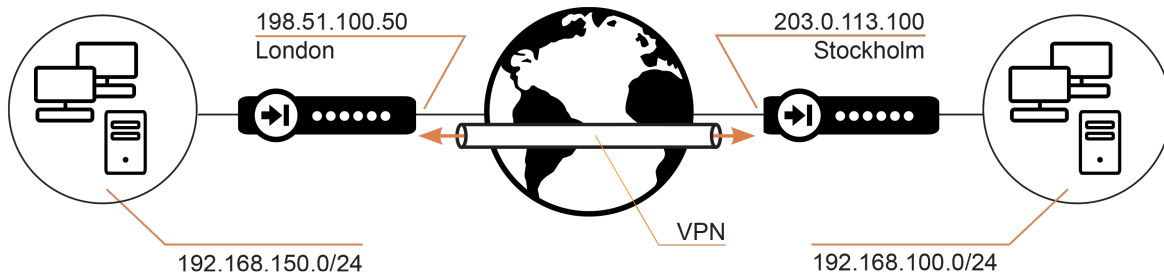


Figure 2.9.2 A standard Lan2Lan tunnel scenario we will use as base for this scenario

Requirements

Currently both London and Stockholm can access the Internet from each side directly.



Note

We have not gone into specifics when it comes to the Internet access rules and routing in the various recipes, but it is assumed those types of rules exist, because a firewall without Internet access would be rather unusual. The first Clavister cookbook as well as the admin guide describes Internet setup in great detail.

For this scenario we have the following requirements:

- Stockholm must be allowed to access the Internet directly
- London must not be allowed to access the Internet directly
- London must be allowed to access the Internet using the IPsec tunnel towards Stockholm
- Stockholm must be configured to protect/restrict users behind both the Stockholm and London Firewalls. For instance using WCF, IDP, Anti-Virus and additional security mechanisms depending on administrator requirements.

Configuring the London Firewall

Before we start we need to examine the current status of the routing table on our London Firewall, we currently have the following routes as shown in figure 2.9.3.

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	Wan	Wan_net	
2	Route IPv4	Wan	all-nets	Wan_gw
3	Route IPv4	Lan	Lan_net	
4	Route IPv4	Stockholm_Tunnel	Stockholm-Net	

Figure 2.9.3 The current routing table status on the London Firewall

As we can see from the routing table status we have configured access only to the network behind the Stockholm tunnel and we have a standard Internet access going through directly towards the Wan interface (route #2) as well as our local network routed behind the Lan interface (route #3).

What we need to do first is to remove route #2 as we want Internet access to go through the tunnel and not directly to the ISP.



Important

Removing route #2 would in this case cause internet access to go down on the London Firewall. Needless to say this kind of operation should be done out of office hours and preferably not by managing the Firewall remotely from the Internet. We will assume that the administrator is managing the London firewall by being able to access it from the local network, Lan_net.

Removing the all-nets route #2 can be done in two ways depending on how the route was added. If the route was added manually it can simply be removed directly by right-clicking it and selecting delete. But if the route was added automatically from the Wan interface we need to open the interface itself and uncheck the checkbox for the option “Automatically add a default route for this interface using the given default gateway” as shown in figure 2.9.4.

Automatic Route Creation

- Automatically add a route for this interface using the given network.
- Automatically add a default route for this interface using the given default gateway.

Figure 2.9.4 Deselecting the interface option to create a default (all-nets) route

Once the route has been removed it is recommended to wait with any configuration deployments until we have completed all the other required changes. If we were to deploy the configuration now, then Internet access would go down, so we will complete the configuration changes before making any deployments.

Without a route towards the Internet ISP, cOS Core will be unable to connect to the remote endpoint (the Stockholm Firewall IP) to establish the IPsec tunnel. In order to solve this problem we must add a new route that informs cOS Core where the remote endpoint for the tunnel is located and how it can be reached.

To accomplish this we create a new route that uses the external interface (Wan) as interface and as the network we will use the (single) public IP of the Stockholm Firewall and lastly as the gateway it will be the ISP gateway in the London office. A summary of the new route is shown in figure 2.9.5.

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	Wan	Stockholm-IP	Wan_gw

Figure 2.9.5 We single-host route the Stockholm Firewall’s public IP towards the London ISP

With this change, the London Firewall does not have any Internet access with one exception, it knows how to reach the public IP of the Stockholm Firewall. This route must exist in order for the IPsec tunnel to be established.

Modifying the IPsec tunnel

Assuming that the IPsec Lan2Lan tunnel between London and Stockholm-HQ is already up and running, we only need to modify the protected network setting as shown in figure 2.9.6.

Protected Networks

Local Network:

Remote Network:

Figure 2.9.6 Specifying that all networks instead of the external Wan are behind the IPsec interface

In order to get this working in Protected Networks, we need to change the Remote Network to be “all-nets” instead of being just the local network in Stockholm. As we want to reach the Internet through the tunnel, we do not know to which IP address the clients will connect. So we must configure the IPsec tunnel to allow requests to any IP address inside the IPsec tunnel. Therefore we select “all-nets” as the remote network.

Once this is done a route must be added to the routing table (or if it is automatically added by the IPsec interface depending on chosen setting). Since we want to use the IPsec tunnel to reach the Internet the all-nets route must now point to the IPsec interface as shown in figure 2.9.7.

#	Type	Interface	Network	Gateway...
1	Route IPv4	Wan	Stockholm-IP	Wan_gw
2	Route IPv4	Stockholm_Tunnel	all-nets	
3	Route IPv4	Wan	Wan_net	
4	Route IPv4	Lan	Lan_net	

Figure 2.9.7 The all-nets route now points to the IPsec interface instead of Wan

The updated routing table will tell cOS Core that in order to reach the Internet, cOS Core must send the request through the IPsec interface.

Take note of the setting on route #1 as well. This is the route exception that we need in order for cOS Core to reach the remote endpoint to establish the tunnel. We must keep in mind the routing principle “smallest route first”, and a single IP is much smaller than all-nets and will be matched first.

Changing the IP Policies

We now move on to the IP Policies. As per previous configuration in *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*, we currently only allow London to communicate with the local network in Stockholm. To change this we only need to change the IP Policy that allows traffic from London to Stockholm to contain unrestricted destination network access as shown in figure 2.9.8.

#	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	Stockholm_To_London	✓	Stockholm_Tunnel	Stockholm-Net	any	Lan_net	all_services
2	London_To_Stockholm	✓	Lan	Lan_net	Stockholm_Tunnel	all-nets	all_services

Figure 2.9.8 Changing the destination network on the outgoing IP Policy towards Stockholm

We do not enable any NAT (network address translation) on the IP Policy as there is currently no need to mask the source IP of the sender towards the IPsec interface.

Not masking the source IP using NAT on the London Firewall can be quite advantageous when it comes to performing troubleshooting or log investigations on the Stockholm-HQ Firewall. If we need to examine the logs of data initiated from the London local network we can see the original IP address in the logs at Stockholm, which would reduce the time needed to trace data or packets back to the originator by avoiding having to connect to the London Firewall as well.

Configuring the Stockholm Firewall

Similarly as on the London Firewall we start by examining the status of the routing table on the Stockholm Firewall. The current status is shown in figure 2.9.9.

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	London_Tunnel	LondonNet	
2	Route IPv4	Wan	Wan_net	
3	Route IPv4	Wan	all-nets	Wan_gw
4	Route IPv4	Lan	Lan_net	

Figure 2.9.9 The current status of the routing table in the Stockholm Firewall

As the Stockholm Firewall will be the one used for public Internet access we do not need to do any changes at all to the routing table. We still expect traffic to arrive from the London tunnel from the LondonNet, so no changes are needed here either. The Stockholm Firewall must use its primary Wan interface to reach the Internet.

Modifying the IPsec tunnel

The IPsec tunnel in London has been changed to allow all-nets (all available IPv4 addresses) to be sent inside the IPsec tunnel (see *Figure 2.9.6*). As changes on one side of the tunnel must be done on the other side, we need to configure the Stockholm IPsec tunnel to match the new settings on the London side in order for the IPsec negotiations to complete successfully.

The principle was explained in *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*. To once again highlight the changes needed we can have a look at figure 2.9.10.

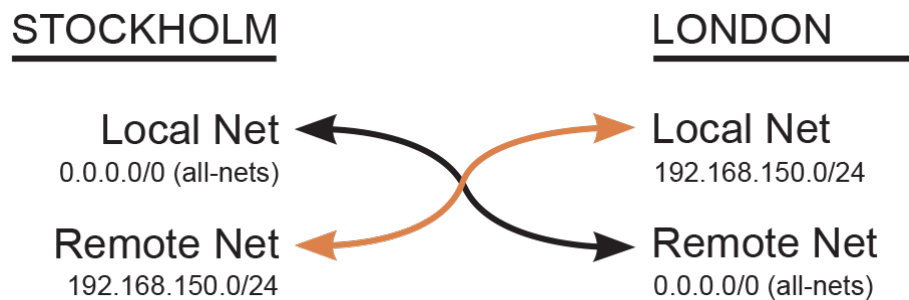


Figure 2.9.10 Changing the Local Network on Stockholm to match the network setting in London

Since we changed the Remote Network in London to be all-nets we need to perform a similar change in Stockholm but here we change the Local Network as shown in figure 2.9.10 and 2.9.11.

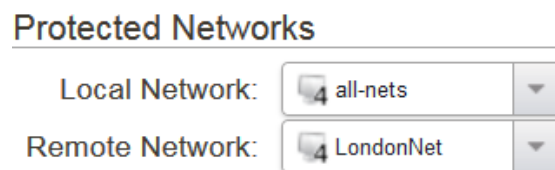


Figure 2.9.11 Changing the Local Network on Stockholm to match the network setting in London

And that is all the changes needed on the IPsec tunnel interface. In principle we have indicated that the Internet is part of the Local Network in Stockholm. Which is true in a sense, because we will forward the packets further out on the Wan interface as per our routing table configuration in Stockholm.

However, from the perspective of the Firewall in London, the Internet is located in Stockholm or behind the Stockholm Firewall depending on how we view it.

Changing the IP Policies

Currently the IP Policies in Stockholm only allow communication from the London internal network to communicate with the Stockholm internal network. This is just fine and we do not want to do any changes here as we still want to allow this communication. But we need to create an additional IP Policy to allow the traffic from London to the Internet coming from the IPsec tunnel interface.

The rule we need to create is shown in figure 2.9.12.

# ^	Name	Log	Src If	Src Net	Dest If	Dest Net	Service	Address Translation
1	London_To_Internet	✓	London_Tunnel	LondonNet	Wan	all-nets	all_services	SRC:NAT

Figure 2.9.12 Creating the new IP Policy needed to allow users behind London to surf the Internet through Stockholm

In order to allow traffic from the London users towards the Internet we must configure the IP Policy to allow access from the tunnel to the internet located behind the Wan interface.

In our example above we used "all_services" as the service which means all ports and protocols. This should only be done for testing purposes, it is highly recommended to allow only the needed services/ports such as http(80), https(443), dns(53) etc. But it will be up to the administrator to decide what should be allowed or not.

And lastly we need to NAT the traffic towards the Internet as we cannot connect to our ISP's router with a sender IP coming from a private IP address in the London internal network range. That would be dropped immediately. The sender must use the public IP address of the Wan interface in Stockholm, which is why we must choose NAT as the address translation on our rule.

Requirement fulfillment check

That concludes the configuration of this scenario. We will now perform a check of the requirement list we defined at the start of the recipe:

- Stockholm must be allowed to access the Internet directly
 - We made no changes to the routing table or IP policy rules in Stockholm so our Stockholm users still have direct access to the Internet.

- London must not be allowed to access the Internet directly
 - We achieved this by removing the all-nets route pointing to the physical Wan interface from the routing table in London.
- London must be allowed to access the Internet using the IPsec tunnel towards Stockholm
 - We achieved this by changing the all-nets route from the Wan interface to the IPsec interface pointing to Stockholm and also modified the IP policy rule to point to the new "Internet" interface.
- Stockholm must be configured to protect/restrict users behind both the Stockholm and London Firewalls using WCF, IDP, Anti-Virus and additional security mechanisms depending on administrator requirements.
 - This has currently **not** been achieved as it will be up to the administrator to determine which security mechanisms to activate and how to proceed from this point. But everything is ready to further lock down the services and to activate additional security mechanisms on the traffic towards the Internet. This recipe is primarily focused on the IPsec part of the scenario and will not go into details about IDP, AV, Application Control etc. For more information about those functions please see either the first Clavister Cookbook or the administrator guide.

Using a ping simulation to verify rules and routing

A very useful “tool” exists in the CLI which we can use to verify that the rules and routing seems to be correctly configured.

Using the “ping” command in the CLI, we can use subcommands to make it simulate data traffic going to or arriving from a specific interface or source IP. A simple test would be to use this command on the Stockholm Firewall to simulate that traffic is arriving from a host in the London network range that tries to send a ping to a host on the Internet.

```
Stockholm:/> ping 192.36.125.18 -srcif=London_Tunnel -srcip=192.168.150.50 -verbose
Rule and routing information for ping:
    allowed by rule "London_To_Internet"

Sending 1 4-byte ICMP ping to 192.36.125.18 from 203.0.113.100
sent via route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "main"
ICMP Reply from 192.36.125.18  seq=0  time= 10 ms  TTL=58

Ping Results:  Sent: 1, Received:1, Avg RTT: 10.0 ms
```

For more information about ping simulations, please see Appendix-A.

Alternative solution using Virtual Routing in London

Due to the complexity of cOS Core and the large amount of features, functions and the ability to redirect and forward traffic in all manner of directions, most recipes and scenarios in this Cookbook have alternative solutions. This scenario has a fairly easy to implement alternative solution on how to forward all public Internet traffic into the IPsec tunnel on the London Firewall.

Our current solution was to remove the “all-nets” route and add a single-host route that points to the Stockholm Firewall. An alternative solution to this would be to use VR (virtual routing).

If we go back to how the routing table looked on the London Firewall at the start of this recipe, as shown in figure 2.9.13.

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	Wan	Wan_net	
2	Route IPv4	Wan	all-nets	Wan_gw
3	Route IPv4	Lan	Lan_net	
4	Route IPv4	Stockholm_Tunnel	Stockholm-Net	

Figure 2.9.13 The status of the routing table at the start of this recipe

The problem is, when we change the Stockholm_Tunnel to use "all-nets" as its network, then it would conflict with the "all-nets" route that is configured on the Wan interface. The previous solution was to remove the "all-nets" route from Wan and then add a single-host route towards the route endpoint (Stockholm's public IP) as shown in previous Figure 2.9.7.

To use VR to solve the same scenario we first create a new routing table under "Network->Routing->Routing Tables" with ordering "Only" as shown in figure 2.9.14.

# ▲	Name	Ordering
1	main	Default
2	RT_1	Only

Figure 2.9.14 Adding a new routing table with ordering Only

The ordering is not that important as this scenario would work no matter which ordering we have chosen. Using "Only" is however easier to understand, because we know that anything processed within this routing table stays in this routing table. For more information about the different sequence order please see the cOS Core Administration Guide.

Once the new routing table has been created we want to move two routes from the main routing table into this routing table, the two routes we want to move are the two routes towards the Wan interface.

```
Wan Wan_net
Wan all-nets Wan_gw
```

If both of these routes are automatically added by the interface itself the easiest way to move them would be to open the Wan interface properties and then go to the Virtual Routing tab as shown in figure 2.9.15.

Wan

An Ethernet interface represents a logical endpoint for Ethernet traffic.

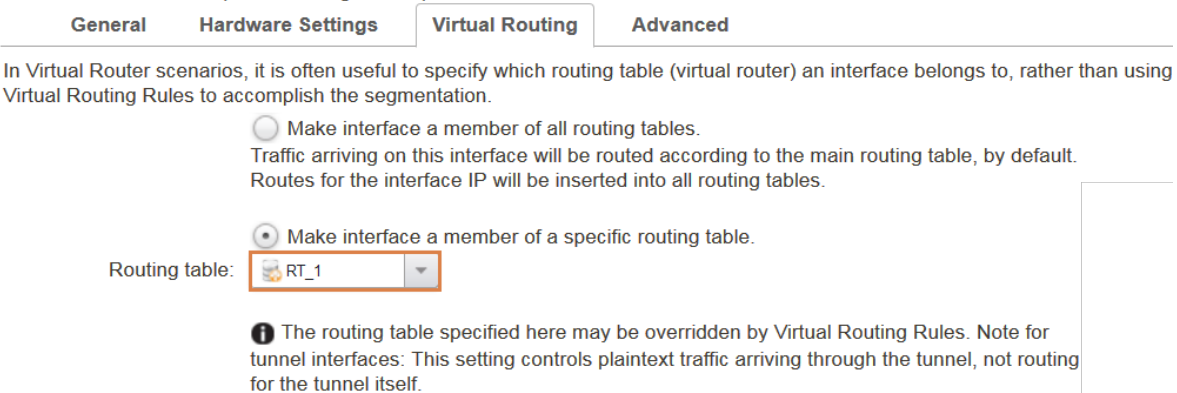


Figure 2.9.15 Changing the routing table membership for the Wan interface

If they are not added automatically by the interface they need to be deleted manually from the main routing table and then recreated in the RT_1 routing table. Once this operation is complete the routing table called "RT_1" should contain two routes as shown in figure 2.9.16.

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	Wan	Wan_net	
2	Route IPv4	Wan	all-nets	Wan_gw


Figure 2.9.16 The contents of the "RT_1" routing table


The last thing we need to do now is to modify the IPsec tunnel. First we need to change the "Remote Network" to be "all-nets" again, as we want Internet access to be beyond the IPsec tunnel. The remote network previously used to be the local network in Stockholm (before the above change).

The second change that needs to be done is to change the Outgoing Routing Table. This setting can be found under the IKE (Phase-1) tab on the IPsec tunnel.

We change the routing table from <Main> to our newly created RT_1 routing table as shown in figure 2.9.17.

IKE Peers Settings

Outgoing Routing Table: 

Local Endpoint: 


Incoming Interface Filter: 

Figure 2.9.17 Setting the RT_1 routing table as the Outgoing Routing Table on the IPsec tunnel in London

The function of the Outgoing Routing Table is to tell the IPsec tunnel which routing table it should use to try to locate the Remote Endpoint towards which the tunnel should establish connection, in short, how to locate the public IP of the Stockholm Firewall.

The alternative configuration is now complete and we will achieve the same results. We only need to do this change on the London Firewall, the Stockholm Firewall should still be configured as described in the *Configuring the Stockholm Firewall* section.

Both London configuration methods work just fine. Which one is used in practice is up to the administrator's decision.

A final check of the main routing table in London, as shown in figure 2.9.18, and it is almost empty.







# ▲	Type	Interface	Network	Gateway
1	 Route IPv4	 Stockholm_Tunnel	 all-nets	
2	 Route IPv4	 Lan	 Lan_net	

Figure 2.9.18 The final state of the <main> routing table in London

The configuration becomes very streamlined when using the VR solution. And that concludes this recipe.

Questions & answers

Question: What about DNS queries? Should we use the local ISP DNS in London or the DNS server of the ISP in the Stockholm office? Or perhaps a local DNS server in Stockholm?

Answer: This will basically be up to the administrator to decide. The method that requires no changes to the current setup would be to use either the ISP DNS or local DNS server in Stockholm.

Question: What if I want to perform this configuration change remotely from the internet. Is it not possible?

Answer: It is possible but it requires that there exist routes using the Wan interface towards the management host. There are multiple ways to solve this depending on the exact scenario; static single-host routes, VR or Policy Based Routing. That particular scenario will not be described in this book, refer to the Administrator's Guide.

Question: What if I want to NAT from Stockholm to London? Which IP would be the sender if we have "all-nets" as the local network?

Answer: If "all-nets" is used as the Local Network, cOS Core will be unable to determine which IP it should use as sender, because "all-nets" means all possible IPv4 addresses. In order to get this to work we must tell the IPsec tunnel which IP it should use a sender. This is configured on the IPsec tunnels "Advanced" tab. The option is shown in figure 2.7.19.

IP Addresses

Automatically pick the address of a local interface that corresponds to the local net.
 Specify address manually:

IP Address:

HA IP Address:

Figure 2.9.19 Manually setting an IP address on the IPsec tunnel which it can use in e.g. NAT scenarios or when traffic is initiated from the Firewall (Core) itself

This problem is fairly easy to spot by using the CLI and attempt to ping something on the other side of the tunnel, by using the `-verbose` flag on the ping command cOS Core will print out the sender IP.

```
Stockholm:/> ping 192.168.150.1 -verbose  
Sending 1 4-byte ICMP ping to 192.168.150.1 from 127.0.0.1
```

And as we can see in the above output, the sender would be 127.0.0.1. To solve this problem we must tell the IPsec tunnel which IP it should use as a sender in this particular scenario.

Chapter 3: L2TPv3

3.1. An introduction to L2TPv3

In this chapter we will go through some of the basics of L2TPv3 as well as recipes of scenarios where L2TPv3 can be used.

L2TP Version 3 (L2TPv3) is a tunneling protocol that is an alternative to standard L2TP (standard L2TP is also referred to as L2TPv2). L2TPv2 can only tunnel PPP traffic, whereas L2TPv3 has the key advantage being able to process OSI layer two traffic (such as ARP). This is sometimes referred to as Layer two Tunneling or as a pseudowire. This means L2TPv3 can carry Ethernet frames over an IP network, allowing one or more Ethernet LANs to be joined together across the public Internet. cOS Core L2TPv3 can tunnel both Ethernet as well as VLANs.

Further giving an example of pseudowire, it is simply an emulated circuit. By using L2TPv3, it is possible to extend a number of layer two circuit types over an IP network.

The most basic of scenarios where L2TPv3 can be used is when we want to extend a local network to also exist beyond an encrypted tunnel using layer two. Meaning the same network exist on both sides of the tunnel as if the clients was sitting locally as shown in figure 3.1.1. The clients themselves will be unaware that part of its own network segment is located beyond a VPN tunnel.

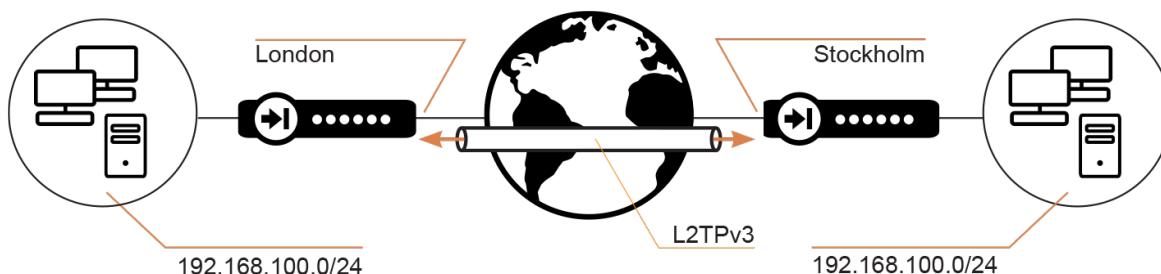


Figure 3.1.1 The basic L2TPv3 scenario where hosts on either side of the tunnel will believe the other machine is on the same local network segment.

Points of interest when implementing L2TPv3

A few points of interest when considering implementing L2TPv3:

- Unlike standard L2TP, L2TPv3 does not provide encryption of transmitted data. If the L2TPv3 tunnel is to be secure, it should be used over another tunneling protocol such as IPsec.
- L2TPv3 support in cOS Core allows the Firewall to act as either an L2TPv3 server or a client.
 - This will be described in the upcoming recipes.
- L2TPv3 does not support HA cluster. L2TPv3 is using transparent mode / switch route solutions that is not possible to configure/use on an HA cluster due to the lack of loop avoidance.
- cOS Core L2TPv3 can only be used with IPv4. IPv6 is not supported by cOS Core at this time. However, IPv6 can be allowed to be transmitted, as described next.

Passthrough options

Both the cOS Core L2TPv3 client and server objects have two pass through properties associated with them:

1. **DHCP Passthrough**

This allows DHCP protocol traffic to pass through the L2TPv3 client/server interface(s).

2. **Non-IP Protocol Passthrough**

This allows non-IPv4 protocol traffic to flow. IPv6 traffic is an example of traffic which will be allowed when this property is enabled. However, the IPv6 traffic will not be subject to any configuration rules or policies.

Please note that these options are disabled by default and when enabled, the traffic that they allow will not be subject to any rules or policies in the cOS Core configuration.

Recipe 3.2. Bridging a layer 2 network using L2TPv3 & IPsec

Objective

The objective of this recipe is to implement a Layer 2 Tunneling Protocol version 3 (L2TPv3) server/client solution to solve a scenario where we have the same local network on two locations as shown in figure 3.2.1.

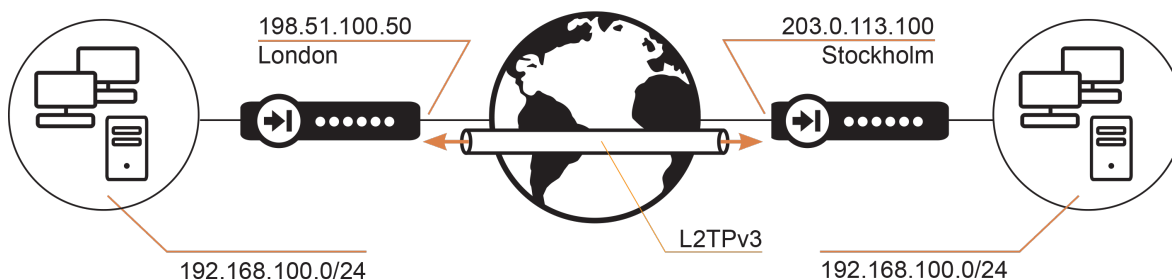


Figure 3.2.1 A situation where the network in London and Stockholm is the same

In this scenario the local networks in London and Stockholm are the same. One solution to this scenario would be to change the network on either side to avoid the conflict.

But let us assume it is not possible to make changes due to one reason or another. An alternative solution would be to connect the two networks together using a Layer 2 VPN tunnel. Meaning that a host in London can communicate with a host in Stockholm believing it is part of its own local network segment.

There are some things to consider, however such as the possibility of IP conflicts of statically configured machines on either side. But to keep the scenario simple we will assume that there are no IP conflicts on either side and that any such problems have already been sorted out.

Detailed discussion

This part will contain some background information regarding L2TPv3 and transparent mode. To jump directly to the scenario configuration, go to the Requirements section further into this recipe.

L2TPv3

L2TP Version 3 (L2TPv3) is a tunneling protocol that is an alternative to traditional L2TP (traditional L2TP is also referred to as L2TPv2). L2TPv2 can only tunnel PPP traffic, whereas L2TPv3 has the key advantage of emulating the properties of an OSI layer 2 service. This is sometimes referred to as Layer 2 Tunneling or as a pseudowire. This means L2TPv3 can carry Ethernet frames over an IP network, allowing one or more Ethernet LANs to be joined together across the public internet.

In cOS Core, L2TPv3 can tunnel both Ethernet as well as VLANs.

A summary of the advantages of using L2TPv3 over L2TPv2:

- Can be carried directly over IP without UDP. L2TPv2 requires UDP.
- Offers better security against man-in-the-middle or packet-insertion attacks.
- Has support for many more tunnels or many more sessions within one tunnel.
- Can be manually configured with static parameters and does not require a control channel.

Other important considerations with L2TPv3 are:

- Like standard L2TP, L2TPv3 does not provide encryption of transmitted data. If the L2TPv3 tunnel is to be secure, it should be used with IPsec or PPPoE.
- cOS Core L2TPv3 can only be used with IPv4. IPv6 is not supported by cOS Core at this time.
- L2TPv3 support in cOS Core allows the Clavister Security Gateway to act as either an L2TPv3 server or a client. Next is described how to set up these two functions.

L2TPv3 can run over IP or UDP, and covers - when running over UDP - a mechanism to automatically fallback to L2TPv2 should the other end not support L2TPv3. When running over UDP, L2TPv3 uses the same port as L2TPv2 (1701).

But perhaps the most interesting aspect of L2TPv3 is the ability for ARP to traverse an encrypted connection. No other VPN type (IPsec, GRE, PPTP etc) have support for this except L2TPv3. There are of course some alternative ways to get ARP to “partially” work using e.g. ProxyARP but then the ARP request itself never traverses the VPN tunnel. ARP traversal is achieved by using transparent mode, also known as switch routes.

Transparent mode

The cOS Core Transparent Mode feature allows a Clavister firewall to be placed at a point in a network without any reconfiguration of the network and without hosts being aware of its presence. All cOS Core features can then be used to monitor and manage traffic flowing through that point.

cOS Core can allow or deny access to different types of services (for example HTTP) and in specified directions. As long as users are accessing the services permitted, they will not be aware of the Clavister Security Gateway's presence. Network security and control can therefore be significantly enhanced with cOS Core operating in transparent mode but while disturbance to existing users and hosts is kept to a minimum.



Note

Transparent mode/switch routes is not supported in HA clusters due to the lack of loop avoidance functionality. This also means that using L2TPv3 in an HA is not viable.

Switch routes

Transparent mode is enabled by specifying a switch route instead of a standard Route in routing tables. With switch routes, cOS Core uses ARP message exchanges over the connected Ethernet network to identify and keep track of which host IP addresses are located behind which interface.

It is recommended not to specify normal non-switch routes for the same interface that is part of the transparent mode switch routes in order to avoid mixing layer 2 and 3 network traffic (it is possible, just not recommended due to various problems that can arise).



Limitation Note

Even though ARP can traverse L2TPv3 it is only partially possible to get broadcasts to traverse L2TPv3. The reason for this limitation is due to the L2TPv3 interface (client and server) not have the option "enable broadcast forwarding". There exist an alternative way to configure this and it will be described later in this recipe in the "Enabling broadcast forwarding on L2TPv3 interface(s)" section.

Requirements

Moving past the background information we now come to the requirements we have for this scenario. We have three requirements:

- Clients in London should be able to locate hosts in Stockholm using ARP and ICMP.
- Clients in Stockholm should be able to locate hosts in London using ARP and ICMP.
- The communication between the two endpoints must be encrypted.

Configuring the Stockholm Firewall (server)

We will start the configuration on the Stockholm Firewall because it will act as the server. By default the traffic encapsulated by L2TPv3 is not encrypted so in order to add a layer of security we will encapsulate the L2TPv3 traffic inside an IPsec tunnel as shown in figure 3.2.2.



Figure 3.2.2 In order to protect the data we encapsulate the L2TPv3 traffic inside an encrypted IPsec tunnel

Configuring the IPsec tunnel

For our IPsec tunnel we will configure it using Transport mode. The reason for this is because transport mode is designed for communication between individual IP addresses and the only communication inside the tunnel will be between the L2TPv3 client and server IPs. The communication between our users in Stockholm and London networks will take place inside an L2TPv3 tunnel which in turn is inside the IPsec tunnel.

Once the IPsec interface is created we change the Encapsulation Mode (1) to be Transport mode as shown in figure 3.2.3.

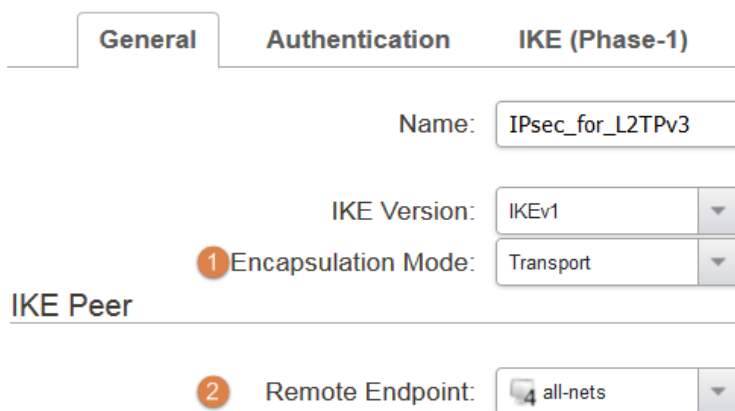


Figure 3.2.3 The General settings on the IPsec interface to be used by L2PTv3

For the Remote Endpoint (2) we have in this case chosen to use “all-nets”. This is however up to the administrator as an alternative would be to input the public IP of the London Firewall. The reason we use “all-nets”, is in case we want additional remote offices connected using L2TPv3, in other words, bridge the same local network over more than two offices.

For Authentication we can use either PSK or certificates.

There are two more settings we need to change on the IPsec tunnel before we move on to the next configuration object, the first setting is the Local Endpoint as shown in figure 3.2.4

The screenshot shows the 'IKE Peers Settings' configuration page. It features three dropdown menus: 'Outgoing Routing Table' set to 'main', 'Local Endpoint' set to 'Wan_ip' (highlighted with an orange border), and 'Incoming Interface Filter' set to 'any'.

Figure 3.2.4 The Local Endpoint on the IPsec tunnel must be set to be the external IP address of the Firewall

The Local Endpoint must be set to be the external IP address of the Firewall. The reason for this is because we are using Transport Mode for an Endpoint-to-Endpoint solution. We must be able to tell cOS Core specifically which IP address it should use as the sender for the tunnel negotiation (even if we only have one public IP address assigned to the Firewall). This applies both for the server and client.

The second setting we need to change on the IPsec tunnel is the automatic route creation as shown in figure 3.2.5.

The screenshot shows the 'Routing' configuration page. It contains three settings: 'Add route dynamically' (unchecked), 'Add route statically' (unchecked and highlighted with an orange border), and 'Plaintext MTU' set to '1420'.

Figure 3.2.5 The automatic route creation must be disabled

The “Add Route Statically” must be disabled as all routes used by IPsec for this particular scenario are handled automatically by the IPsec engine itself, the same goes for the dynamic route option.

Configuring the L2TPv3 server

We now move on to configuring the L2TPv3 server. Adding a L2TPv3 server is done in the same area as IPsec under *Network->Interfaces and VPN->VPN and Tunnels->L2TPv3 server*. Once the server interface is created we are met with the options shown in figure 3.2.6.

The screenshot shows the configuration window for an L2TPv3 server interface. The 'General' tab is selected, and the settings are as follows:

- Name:** L2TPv3_Srv
- 1 Inner IP Address:** Lan_ip
- 2 Local Network:** SwitchNet
- 3 Protocol:** UDP
- 4 Outer Interface Filter:** IPsec_for_L2TPv3
- 5 Server IP:** Wan_ip

Below the main settings, there is a section for **Transparent Mode** with the following options:

- 6 DHCP Passthrough:**
- 7 L2 Passthrough for Non-IP Protocols:**

Figure 3.2.6 The various general options for the L2TPv3 server interface

First, we need to specify the Inner IP Address (**1**), this is the IP address of the L2TPv3 server interface itself. We set it to be the Local Lan interface IP address (which is 192.168.100.1 in this case). We must keep in mind however that in a complete transparent mode installation there may not be any IP addresses set on the local interface(s), so the Inner IP address will be set to localhost (127.0.0.1) in many scenarios.

The Local Network (**2**) will be an object we call "SwitchNet". This object is the same as "Lan_Net" (192.168.100.0/24). We have chosen to call it SwitchNet in order to give a clear indication that the object is meant to be used by a switched network segment in the Firewall.

The Protocol (**3**) determines which protocol that the L2TPv3 server/client should use to transport the data. There are two options:

UDP:

Using UDP as the lower level transport protocol is the default setting for this property and is recommended. It ensures that communication is able to traverse most network equipment, particularly if NAT is being employed in the path through the network.

IP:

Using IP as the transport protocol allows packet processing to be optimized and therefore provides a means to transport data using less processing resources. However, some network equipment may not allow traversal and problems can occur where NAT is employed in the path through the network. Such problems can be solved by using UDP instead.

In our scenario we are encapsulating L2TPv3 inside IPsec, so it does not matter much which one we choose, so we will go for the default UDP. But for scenarios not involving IPsec, this option may have to be considered.

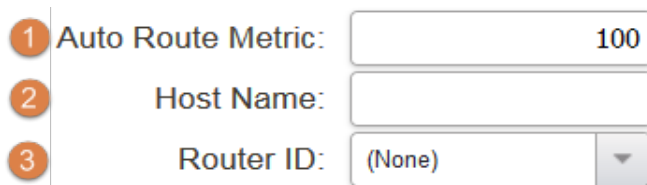
The Outer Interface Filter (**4**) determines which interface the L2TPv3 server should listen on for incoming connections. Due to the IPsec encapsulation we instruct the L2TPv3 server to listen on the IPsec interface we created earlier, since we expect incoming connections to arrive from the IPsec interface/tunnel.

The server IP (**5**), this is the IP address the L2TPv3 server should listen on for incoming connections. Using the external IP of the Firewall may seem strange here but since we are using encapsulation mode for the IPsec tunnel it will use the external IP inside the tunnel, and this is the IP we expect incoming connections on (even though it is inside the IPsec tunnel).

There are two additional options under the Transparent mode, the first option is DHCP Passthrough (**6**) which means that it is possible to get DHCP requests/answers to pass through on layer 2 through the L2TPv3 tunnel. In this scenario we have not configured any DHCP server on either side but if that is the case and we want for instance one DHCP server to hand out IP leases to both offices, this checkbox should be enabled.

The last option under the General options tab is called L2 Passthrough for Non-IP protocols is a more unusual setting that allows traffic / data that is not an IP protocol to pass through the L2TPv3 interface. An example of a non-IP protocol is IPX.

On the L2TPv3 server's Advanced tab we have some options that, in this scenario, we will not change but we will go through them nonetheless. The options are shown in figure 3.2.7.



The image shows a configuration interface with three numbered items:

- 1 Auto Route Metric: 100
- 2 Host Name: (empty field)
- 3 Router ID: (None) [dropdown arrow]

Figure 3.2.7 The options under the Advanced tab on the L2TPv3 server

The first option called Auto Route Metric (1) determines the metric that should be set for the switch route that will be created once the L2TPv3 server is activated. We will discuss more about switch routes in the subsequent heading in this recipe.

Host Name (2) and Router ID (3) may have to be set as some L2TPv3 clients may require/request these settings. If the Host Name is left blank the tunnel’s Inner IP address will be used. Since our scenario is an L2TPv3 between two units running cOS Core, we leave these settings at their default.

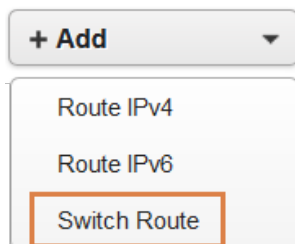
Creating the switch route

By default the L2TPv3 interface automatically creates a switch route on itself based on the network defined on the L2TPv3 interface (as shown in Figure 3.2.6).

This switch route tells cOS Core that it should keep track of all ARP related messages arriving on this interface in order to build a database of which MAC addresses exists behind the L2TPv3 interface.

But this is not enough. We must also inform cOS Core of all other interfaces that is part of the 192.168.100.0/24 switched network in order for it to start building a complete database of who is where, in order to be able to route packets via the correct interface.

To accomplish this we add a new route in the main routing table of the type “Switch Route” as shown in figure 3.2.8.



The image shows a dropdown menu with the following options:

- + Add
- Route IPv4
- Route IPv6
- Switch Route (highlighted with a red box)

Figure 3.2.8 Adding a new route in the routing table of the type “Switch Route”

Once the route has been created it looks very similar when creating a normal route and is shown in figure 3.2.9.



Note

It is also possible to add the switch route by enabling the "Transparent mode" option on e.g. the Lan interface itself. There is no difference between the two methods and either will work just fine. In this example we will create a manual switch route.

General	Proxy ARP
1	Switched Interfaces: Lan
2	Network: SwitchNet
3	Metric: 100
4	Forward Broadcast Traffic: <input checked="" type="checkbox"/>

Figure 3.2.9 The properties of a Switch Route in the routing table

The first option called Switch Interfaces (**1**) determines what interfaces that should be used for the switch route. An interface group object can be used here in case we have more than one internal interface that we want part of the switched network (**2**).

There are two methods to make multiple interfaces part of the switched network. One is to use an interface group on the Switched Interfaces (1) setting. The second method is to create a second switch route for the other interface(s) that should be part of the switched network. Which method to use will be up to the administrator as there are no technical or behavioral differences.

The metric setting (**3**) is the same as for normal routes. We will leave the metric setting at its default 100.

Next is the Forward Broadcast Traffic (**4**) setting. As previously mentioned Broadcast Forwarding is not possible on the L2TPv3 interface but it is possible to use on physical/VLAN interface. An example would be if we have more than one interface that is part of the transparent mode scenario, so instead of Lan we have Lan + DMZ. In that scenario we most likely intend to have the Broadcast Forwarding option enabled in order for hosts to communicate using broadcasts

between Lan and DMZ. Even though it is not possible to do this to/from the L2TPv3 interface (without the alternative solution later in the recipe) there is no reason why we should not allow it between interfaces that can. In order to be prepared for future extensions/expansions of the transparent mode scenario we enable this option.

A final check of the switch routes using the CLI command "routes" will look like this on Stockholm (and also London):

```

Network                Iface                Metric
-----
192.168.100.0/24      switched             100
192.168.100.0/24      switched             100
    
```

The interface is listed twice as "switched" because this indicates that the network 192.168.100.0/24 belongs to a switch route between two interfaces.

Adding needed IP Policy rules

The next step of the configuration is the IP Policy rules needed to allow the traffic to/from the L2TPv3 interface.

We need a total of two rules, one that allows traffic from the local Lan interface to the L2TPv3 interface and vice versa another rule that allows traffic from the L2TPv3 interface to the Lan interface, as shown in figure 3.2.10.

# ▲	Name	Log	Src If	Src Net	Dest If	Dest Net	Service
1	▶ To_Switchroute	✓	Lan	SwitchNet	L2TPv3_Srv	SwitchNet	all_services
2	▶ From_Switchroute	✓	L2TPv3_Srv	SwitchNet	Lan	SwitchNet	all_services

Figure 3.2.10 The two IP policy rules needed to allow traffic to/from the L2TPv3 interface

This implies one rule for each direction. We want to allow connections to be initiated from both the Lan interface and from the other side of the L2TPv3 interface.

The source and destination network will be the same object as we will allow traffic to/from only one network as they are part of a switched network (192.168.100.0/24). The switch routes and CAM (Content Addressable Memory) table will keep track of which host is behind which interface in order for cOS Core to know to/from which interface it should forward the various requests.

More information about the CAM table can be found in the "*Additional information on how transparent mode works, details about the CAM table (Content Addressable Memory)*" section later in this recipe.

Configuring the London Firewall (client)

We will now move on to the London firewall where we will configure and use the L2TPv3 client.

Configuring the IPsec tunnel

We will start by configuring the IPsec interface. The process is almost identical to the IPsec interface used by the L2TPv3 server. We only need to configure three settings. First select encapsulation mode to be "Transport mode" then add the Stockholm Firewall as the Remote Endpoint and the external Wan_ip as the Local Endpoint as shown on the IPsec tunnel summary view in figure 3.2.11.




Name	Remote Endpoint	Local Endpoint
 IPsec_For_L2TPv3	 Stockholm-IP	 Wan_ip

Figure 3.2.11 The IPsec tunnel used by the L2TPv3 client

Configuring the L2TPv3 client

Configuring the L2TPv3 client is fairly similar to the L2TPv3 server but with some differences. The various options we have for the L2TPv3 client is shown in figure 3.2.12.

The screenshot shows the configuration interface for an L2TPv3 client, divided into three tabs: General, Virtual Routing, and Advanced. The General tab is active and contains the following settings:

- Name:** L2TPv3_Client
- Inner IP Address:** Lan_ip (callout 1)
- Local Network:** SwitchNet (callout 2)
- Pseudowire Type:** Ethernet (callout 3)
- Protocol:** UDP (callout 4)
- Remote Endpoint:** Stockholm-IP (callout 5)
- Transparent Mode:**
 - DHCP Passthrough:
 - L2 Passthrough for Non-IP Protocols:
- IPsec:**
 - IPsec Interface: IPsec_For_L2TPv (callout 7)

Figure 3.2.12 Configuring the L2TPv3 client

We will go through the options briefly as the majority of the options were already discussed when configuring the server in Stockholm and the server and client share many similarities in the options.

The Inner IP Address (**1**) will be the IP address of the internal/local interface.

The Local Network (**2**) will be the network used by the transparent mode / switch routes, which in our case is SwitchNet (192.168.100.0/24).

The Pseudowire Type (**3**) determines if the L2TPv3 interface should tunnel Ethernet or VLAN tagged Ethernet frames. We will use Ethernet as we will not use VLANs in this scenario. We will discuss and use VLANs for L2TPv3 in the next recipe.

Protocol (**4**) will use UDP, same as that is configured on the L2TPv3 server in Stockholm.

Remote Endpoint (**5**) will be the public IP address of the Stockholm Firewall.

The Transparent Mode (**6**) options will not be enabled in this scenario. Depending on the scenario it will not be too uncommon to have both options enabled, at least in order to allow DHCP.

For the IPsec Interface (**7**) we select the IPsec interface that we created earlier in order to encapsulate L2TPv3 inside an IPsec tunnel when the L2TPv3 client makes a connection attempt to the Stockholm L2TPv3 server.

Creating the switch route

Similarly as on the L2TPv3 server we need to create another switch route in order to inform cOS Core which interface(s) form part of the switched network. So far we only have a switch route for the L2TPv3 client interface that was created automatically but we must also add a switch route that points to the local network.

For the L2TPv3 server we created/used a manual switch route and we will create an identical switch route for the client as well as, shown in figure 3.2.13.

The screenshot shows a configuration window with two tabs: 'General' and 'Proxy ARP'. The 'General' tab is active. It contains the following fields:

- Switched Interfaces:** A dropdown menu with 'Lan' selected.
- Network:** A dropdown menu with '4 SwitchNet' selected.
- Metric:** A text input field containing the value '100'.
- Forward Broadcast Traffic:** A checkbox that is checked.

Figure 3.2.13 The switch route created for the local Lan interface

Adding needed IP Policy rules

For the client the rules are identical to the server. We allow traffic both to and from the L2TPv3 interface and to/from the local Lan interface as shown in figure 3.2.14.

Name	Log	Src If	Src Net	Dest If	Dest Net	Service
▶ To_Switchroute	✓	Lan	SwitchNet	L2TPv3_Client	SwitchNet	all_services
▶ From_Switchroute	✓	L2TPv3_Client	SwitchNet	Lan	SwitchNet	all_services

Figure 3.2.14 The two rules that allow traffic to be initiated from either the L2TPv3 or the local Lan interface

And that concludes the recipe on how to configure an L2TPv3 server & client solution to bridge one network together using layer 2 even if the two sites are physically separated by long distances.

Alternative to Transport mode, tunnel mode with SA Per Host

In this recipe we used Transport Mode to setup an encrypted tunnel between two endpoints. Transport mode however is an IKEv1 encapsulation mode only that does not exist in IKEv2. An alternative solution to using Transport mode and IKEv1 would be to change to use tunnel mode for either IKEv1 or IKEv2 and change the Phase-2 setting "Setup SA Per" to use Host instead of Network. The setting can be found on the IPsec tunnels phase-2 tab and is shown in figure 3.2.15.

Protected Networks Settings

Setup SA per: Network

Config Mode Pool:

Name	Description
Network	Create a tunnel between every combination of the networks given in local network and remote network
Host	Create a new tunnel for every two hosts communicating through this IPsec tunnel
Port	Create a new tunnel for each new pair of source and destination ports

Miscellaneous

DS Field: Port

Figure 3.2.15 How to change from "SA per Network" to "SA per Host"

For the L2TPv3 client and server scenario there will always be only two IP addresses that communicate inside the tunnel, which are the IPs used by the L2TPv3 client and server.

By defining the Local Endpoint on the IPsec tunnel we effectively tell each side (server/client) which IP address it should use as the sender for the L2TPv3 tunnel, and in our scenario the IP would be the public IP address of the Firewall.

This will not cause any conflicts as the IP is already owned by cOS Core, we will simply use it on another location than just the external interface IP. If tunnel mode is used we also must define a Local and Remote Network, the easiest would be to use all-nets on both. For the remote network we do not know from where the client(s) will connect in case the client is behind dynamic IP or that we have multiple sites that should connect to the same server.

Additional information on how transparent mode works, details about the CAM table (Content Addressable Memory)

In transparent mode, cOS Core allows ARP transactions to pass through cOS Core, and determines from this ARP traffic the relationship between IP addresses, physical addresses and interfaces. cOS Core remembers this address information in order to relay IP packets to the correct receiver. During the ARP transactions, neither of the endpoints will be aware of the Clavister Security Gateway.

When beginning communication, a host will locate the target host's physical address by broadcasting an ARP request. This request is intercepted by cOS Core, which sets up an internal ARP Transaction State entry and broadcasts the ARP request to all the other switch-route interfaces except the interface that the ARP request was received on. If cOS Core receives an ARP reply from the destination within a configurable timeout period, it will relay the reply back to the sender of the request, using the information previously stored in the ARP Transaction State entry.

During the ARP transaction, cOS Core learns the source address information for both ends from the request and reply. cOS Core maintains two tables, in which to store this information: the Content Addressable Memory (CAM) and Layer 3 Cache. The CAM table tracks the MAC addresses available on a given interface and the Layer 3 cache maps an IP address to both a MAC address and an interface. As the Layer 3 Cache is only used for IP traffic, Layer 3 Cache entries are stored as single host entries in the routing table.

For each IP packet that passes through the Clavister Security Gateway, a route lookup for the destination is done. If the route of the packet matches a Switch Route or a Layer 3 Cache entry in the routing table, cOS Core knows that it should handle this packet in a transparent manner. If a destination interface and MAC address is found in the route, cOS Core has the information

required to forward the packet to the destination. If the route was a **Switch Route**, no specific information about the destination is available and the security gateway will have to discover where the destination is located in the network.

Discovery is done by cOS Core sending out ARP as well as ICMP (ping) requests, acting as the initiating sender of the original IP packet for the destination on the interfaces specified in the **Switch Route**. If an ARP reply is received, cOS Core will update the CAM table and Layer 3 Cache and forward the packet to the destination.

If the CAM table or the Layer 3 Cache is full, the tables are partially flushed automatically. Using the discovery mechanism of sending ARP and ICMP requests, cOS Core will rediscover destinations that may have been flushed.

The contents of the CAM table can be shown using the "CAM" CLI command. The CAM table can also be manually flushed using the "CAM -flush" subcommand. An example on how the CAM table can look is shown below:

```
CAM Table #1
Interfaces: Lan L2TPv3_Srv
MAC Address      Iface
-----
40-85-93-11-31-3f  Lan
00-51-56-83-b8-2a  L2TPv3_Srv
10-00-00-0a-00-02  Lan
94-13-82-bc-de-e1  Lan
41-5A-50-83-b8-2b  L2TPv3_Srv
```

Enabling broadcast forwarding on L2TPv3 interface(s)

Even though the L2TPv3 interface does not have the broadcast forwarding option, it is possible to achieve forwarding of broadcast by configuring the switch routes for the L2TPv3 interface(s) manually.

When configuring the L2TPv3 client and server as we have done in this recipe, the switch route towards the L2TPv3 interface (client and server) is automatically created. The problem is that the option to forward broadcast traffic does not exist on the L2TPv3 interface.

The solution is to add a manual switch route towards the L2TPv3 interface (client and server) that contained the switched network but with the Forward Broadcast Traffic option enabled as shown in figure 3.2.16.

The screenshot shows the configuration for a manual route in the Cisco IOS interface. The 'General' tab is active. The configuration parameters are as follows:

Interface:	L2TPv3_Srv
Network:	SwitchNet
Gateway:	(None)
Local IP address:	(None)
Metric:	100
Forward Broadcast Traffic:	<input checked="" type="checkbox"/>

Figure 3.2.16 Creating a manual route for the SwitchNet towards the L2TPv3 interface with the forward broadcast traffic option enabled

The forward broadcast traffic option needs to be enabled on all interfaces that is part of the switch route. In this recipe we already had it enabled as shown in *Figure 3.2.13*.

Question: Will this not cause any problems in case the L2TPv3 interfaces have already created a switch route for this particular network?

Answer: No, cOS Core will process it as one switch route but with the switch route options merged.

Requirement fulfillment check

That concludes this recipe. We will now perform a check of the requirement list we defined at the start of the recipe:

- Clients in London should be able to locate hosts in Stockholm using ARP and ICMP.
 - We created switch routes for the L2TPv3 interface and the LAN interface on both London and Stockholm sites to allow ARP to traverse between the two endpoints. We also added the necessary IP Policies to allow ICMP from London to Stockholm.

Status: Fulfilled.

- Clients in Stockholm should be able to locate hosts in London using ARP and ICMP.
 - We created switch routes for the L2TPv3 interface and the LAN interface on both London and Stockholm sides to allow ARP to traverse between the two sites. We also added the necessary IP Policies to allow ICMP from Stockholm to London.

Status: Fulfilled.

- The communication between the two endpoints must be encrypted.
 - All traffic that gets forwarded to and from the L2TPv3 interface will be encrypted and encapsulated by the IPsec tunnel defined on the L2TPv3 interface.

Status: Fulfilled.

Recipe 3.3. Bridging a layer 2 VLAN network using L2TPv3 & IPsec

Objectives

The objective of this recipe is to implement a layer 2 transparent solution involving VLANs. We will base this recipe on the previous recipe and will assume that an L2TPv3 client/server solution is already up and running. We will extend the previous scenario with support for VLANs as shown in figure 3.3.1.

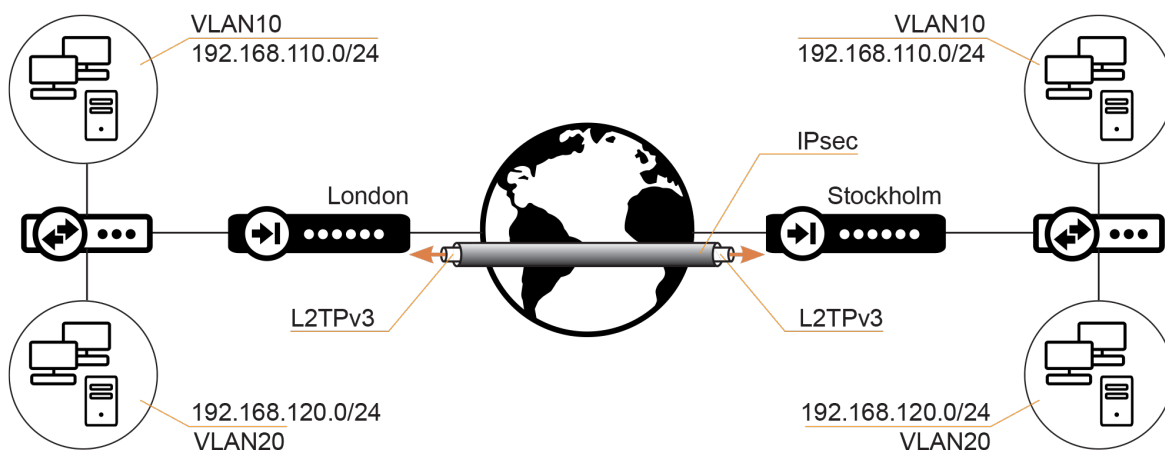


Figure 3.3.1 A situation where the same VLAN exists on both sides of the L2TPv3 tunnel

We have two VLANs, Vlan 10 and Vlan 20, that exist on both sides of the L2TPv3 tunnel. We want users behind each VLAN to be able to communicate with their peers through the tunnel as if they were located locally, just as in the previous recipe, but with the VLAN tag intact over the L2TPv3 interface/tunnel.

Detailed Discussion

Normally VLANs can only be configured/attached on physical interfaces and other VLANs (Service VLANs also known as Q-in-Q) but the exception is the L2TPv3 interface. If an L2TPv3 interface is already configured we have the option to attach a VLAN directly to the L2TPv3 interface and that is what we will do to accomplish this scenario.

For VLAN10 we have chosen the network 192.168.110.0/24 and for VLAN20 192.168.120.0/24 as shown in figure 3.3.1.

Requirements

For this scenario we have two requirements.

1. Clients on each side should be able to communicate with their peers as if they were located locally. Meaning a client located behind VLAN 10 in London should be able to communicate with VLAN 10 in Stockholm.
 - a. The network involved must be the same. Just as in the previous recipe. Meaning for instance that the VLAN 10 network in London & Stockholm must be 192.168.110.0/24.
2. Intercommunication between the two VLAN networks should not be allowed. Only VLAN10 should be able to talk to VLAN10 either locally or through the L2TPv3 tunnel, and the same with VLAN20.
3. Clients on each side should have internet access that uses their own firewalls. Clients in London should use the London firewall's ISP to surf the Internet, same in Stockholm. Meaning that Internet access will not be forwarded or sent through the tunnel.

Configuring the Stockholm Firewall (server)

We will start the configuration on the server side on the Stockholm firewall. It is recommended to make all interfaces a member of a specific routing table (except the Wan interface) to avoid a warning that cOS Core will generate when using transparent mode on an interface where the Virtual Routing membership is not set. The warning says it will use the <main> routing table if not specified, but by specifying which routing table to use we can get rid of the warning that will otherwise be displayed every time a configuration change is made.

We will go into more details about the routing table membership as we progress in this recipe.

Creating the routing tables

The first items we need to create are two new routing tables. We need one routing table for each VLAN ID we want to use in transparent mode.

To create a new routing table we go to Network->Routing->Static Routes->Routing Tables and add new routing tables as shown in figure 3.3.2.

Network » Routing » Static Routes » Routing Tables

Routing Tables

Configure the routing tables of the system.



Figure 3.3.2 Adding a new routing table to be used by each VLAN

Once the new routing table is created we are greeted by the options shown in figure 3.3.3.

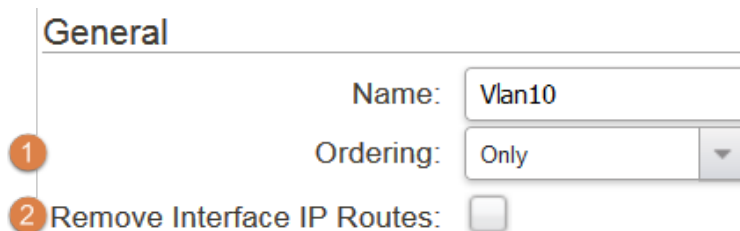


Figure 3.3.3 The routing table properties

The Ordering (**1**) option determines how cOS Core should handle traffic that is received/sent to this routing table. There are three ordering options:

1. **Default**

The default behavior is to first look up the route in the main table. If no matching route is found, or a default route is found (a route with the destination "all-nets"), a lookup for a matching route in the alternate routing table (the one called Vlan10 shown in figure 3.3.3) is performed. If no match is found in the alternate table then the default route in the main table will be used.

2. **First**

This behavior is to first look up the connection's route in the alternate table. If no matching route is found there then the main table is used for the lookup. If the alternative table has an "all-nets" route, it will be counted as a match.

3. **Only**

This option ignores the existence of any other table except the alternate table so that is the only one used for the lookup. One application of this option is to give the administrator a way to dedicate a single routing table to one set of interfaces. The Only option should be used when creating virtual systems since it can dedicate a routing table to a set of interfaces.

For this scenario we will use ordering "Only" as once traffic arrives in this routing table, it should stay in this routing table without any fallbacks to the <main> routing table or otherwise.

The Remove Interface IP Routes (**2**) option is an option that enables us to remove any Core routes that is automatically created by e.g. a physical interface in this routing table. It can be useful for some transparent mode scenarios where we want the Firewall to be totally transparent. But in this case we have need for at least one of the Core routes so we will leave the option unchecked.



Note

We only touched Virtual Routing (VR) very briefly in this recipe. VR and Policy Based Routing (PBR) is one of the most powerful and versatile features in cOS Core. It enables us to change direction, modify and change all aspects of routing and packet flows. The amount of complex routing scenarios that can be achieved/solved using PBR and VR is almost unlimited, VR and PBR is grounds for a book on its own so we will not go into any great detail about VR in this book.

Creating the VLANs

To create the VLAN interfaces we go to Network->Interfaces->VLAN and create the first of our four VLANs as shown in figure 3.3.4. We will discuss why it will be four VLANs and not two further on.

The screenshot shows the configuration for a VLAN interface. The 'General' tab is active, and the interface is named 'Vlan10_Lan'. The 'Base Interface' is set to 'Lan'. The 'VLAN ID' is 10, and the 'Type' is 0x8100. Under 'Address Settings IPv4', the IP address is 'Vlan10_ip', the network is 'Vlan10_net', and the default gateway is '(None)'. The 'Enable DHCP Client' checkbox is unchecked. Under 'Address Settings IPv6', the 'Enable IPv6' checkbox is unchecked. Under 'Transparent Mode', the 'Enable Transparent Mode' checkbox is checked, while 'DHCP Passthrough', 'L2 Passthrough for Non-IP Protocols', and 'Forward Broadcast Traffic' are all unchecked.

Figure 3.3.4 The general VLAN interface properties

The Base Interface (1) is the interface we want to attach the VLAN to. It can be either a physical interface, another VLAN (Service VLAN / Q-in-Q) or an L2TPv3 server. In this scenario the VLAN will be attached to the physical Lan interface.



Note

It is possible to select another non-service VLAN as the base interface, this is a small bug in the GUI as in practice it is not possible to attach a normal VLAN on another normal VLAN (developer ID COP-19796).

The VLAN ID (**2**) determines which ID the VLAN should have, for a normal VLAN it can be between 1 and 4094 (although it is considered good practice not to put user data on VLAN 1).

The Type (**3**) determines what kind of VLAN it is. We have a total of 5 choices:

1. 0x8100 IEEE 802.1Q VLAN (the default)
2. 0x88a8 IEEE diffserv Service VLAN (Q-In-Q)
3. 0x9100 0x9100 VLAN
4. 0x9200 0x9200 VLAN
5. 0x9300 0x9300 VLAN

The first choice is a normal VLAN, the second is a Service VLAN also known as Q-In-Q and the last three entries in the list may be needed to provide interoperability with external equipment from some manufacturers. We will not go into any more details about the various types as we will stick with the standard 802.1Q VLAN in this scenario, so the first VLAN type will be used for all our VLANs.

IP address (**4**) is like any other interface. Depending on the scenario we may not want to set an IP address on the VLAN itself. In our scenario we must allocate an IP address to the VLAN interface, in order for the clients on VLAN10 behind the physical Lan interface to connect to the interface using the VLAN10 IP as their default gateway. As an example, 192.168.110.1 would be an appropriate IP address to assign as VLAN10 IP. We will discuss Internet access further as we progress in this recipe.

For the Network (**5**) we choose the network we have designated for VLAN10 as shown in figure 3.3.1 at the start of this recipe, 192.168.110.0/24.

We will not use any Default Gateway (6), DHCP client (7) or IPv6 (8) as these settings are not applicable for this scenario. IPv6 is not supported on switch routes.

For the last option we enable Transparent mode (9). To keep the scenario fairly simple we will not use any of the other transparent mode options but it will be the decision of the administrator to determine what should be used/allowed or not.

Once the transparent mode option has been enabled the Advanced Setting "Automatically add a route for this virtual LAN interface using the given network" will be automatically enabled. An alternative way to configure would be to use LocalHost as the network, to not check the transparent mode checkbox and then to manually create a switch route in the correct routing table. Both methods work just fine as there are no differences in functionality between them. Both will achieve the same goal.

The next step for our newly created VLAN10 can be found under the Virtual Routing tab on the VLAN interface as shown in figure 3.3.5.

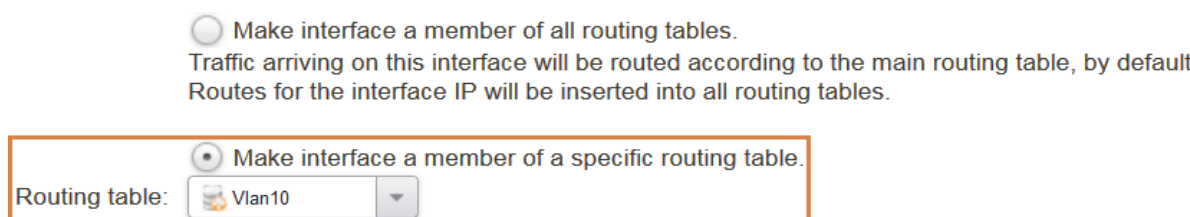


Figure 3.3.5 Making VLAN10 a specific member of the Vlan10 routing table

The task here is to make the VLAN a member of our previously created Vlan10 routing table. This operation has to be repeated for all VLANs. Each VLAN must be placed in its own routing table. So for VLAN20 we have to repeat the process, starting from the section.

The reason that we need to create and place each intended VLAN as part of the switch route in its own routing table, is based on how transparent mode works. Packets received on one VLAN will be automatically propagated on all other VLANs that are part of the switch route in that particular routing table. If we for instance, would have had both VLAN10 and VLAN20 in the same routing table, it means that packets arriving on VLAN10 will be sent out on VLAN20 as well. That would cause a network/routing issue as we do not have the network configured on VLAN10 on VLAN 20.

By placing the involved VLANs only in their own routing table, there is no risk of "leakage" and packet propagation will only be sent out on the VLANs with the correct VLAN tag.

Now there is only one more step that must be done for the VLAN part and if we take VLAN10 as the example we must now create VLAN10 again, but this time as shown in figure 3.3.6.

Name: Vlan10_L2TPv3

1 Base Interface: L2TPv3_Srv

VLAN ID: 10

Type: 0x8100

Address Settings IPv4

2 IP address: localhost

3 Network: Vlan10_net

Default Gateway: (None)

Enable DHCP Client:

Address Settings IPv6

Enable IPv6:

Transparent Mode

Enable Transparent Mode:

DHCP Passthrough:

L2 Passthrough for Non-IP Protocols:

Forward Broadcast Traffic:

Figure 3.3.6 Creating another VLAN10 but this time with the Base interface set to the L2TPv3 server

The difference between this VLAN and the first one is that the Base Interface (1) is now the L2TPv3 server and not a physical interface. The reason for this is because we need to inform cOS Core that the Network (3) allocated on VLAN10 exists on more than one interface as part of a switch route.

Since we will not use internet access through the L2TPV3 interface/tunnel there is no need to allocate an IP address on the VLAN towards the L2TPv3 server, so we simply select LocalHost as the IP address (2).

We repeat the VLAN creation process for VLAN20 as well and make the VLAN20 a member of its own routing table, same as we did for as for VLAN10, and also create an L2TPv3 VLAN.

Creating the needed switch and normal routes

Once all VLANs and Routing Tables have been created we should end up with a routing table summary as shown in figure 3.3.7.




Name	Ordering	Remove interface IP routes
 main	Default	Yes
 Vlan10	Only	No
 Vlan20	Only	No

Figure 3.3.7 The summary of the 3 routing tables

One routing table for each VLAN. We will now go into the Vlan10 routing table as shown in figure 3.3.8 to verify that we have all the required routes for our scenario.



Note

Some may notice that the main routing table is set to ordering "Default" and has the "remove interface IP routes" is set to Yes. This is basically false. The Main routing table is not affected by these options. It will make more sense if we consider these two options for the main routing table to be either blank or that the text "n/a" or "Not Applicable" would be present instead.











Type	Interface	Network	Gateway
 Route IPv4	 Wan	 all-nets	 wan_gw
 Switch Route	 Vlan10_L2TPv3	 Vlan10_net	
 Switch Route	 Vlan10_Lan	 Vlan10_net	

Figure 3.3.8 The contents of the Vlan10 routing table

For our Vlan10 routing table we have a total of 3 routes. The two switch routes are either automatically created by the VLAN interface themselves, if the option “Enable Transparent Mode” is enabled, or created manually by the administrator. In our scenario they are automatically added by the interface as we enabled the transparent mode option.

The route at the top however is a bit special. This route we have added manually into this routing table. The reason why we added this route is because want users located behind the Stockholm Firewall to be able to access the Internet using the Stockholm Firewall’s normal Internet connection. If this route were missing, it would mean that there would be no path in this routing table to the Internet and the Vlan10 users would only be able to talk to the Vlan10 network and nothing else.

When it comes to the external Wan interface it has not been made a member of a specific routing table on the Virtual Routing’s tab on the interface but is left at its default value as shown in figure 3.3.9.

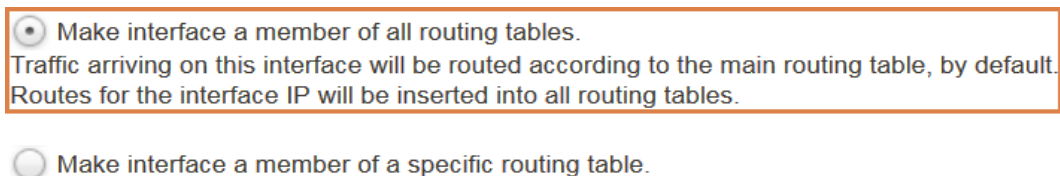


Figure 3.3.9 The Wan interface is not a specific member of any routing table

The reason for this is because the Wan interface has to be present in all the Vlan routing tables in order to tell each routing table and VLAN how to gain access to the Internet.

This of course means we are mixing layer 2 and 3 routing, which may not always be advisable, but for this scenario it will work just fine.

We repeat the route verification for the Vlan20 routing table as well and add the “all-nets” route for the Wan interface as shown in figure 3.3.10.

Type	Interface	Network	Gateway
Route IPv4	Wan	all-nets	wan_gw
Switch Route	Vlan20_L2TPv3	Vlan20_net	
Switch Route	Vlan20_Lan	Vlan20_net	

Figure 3.3.10 The contents of the Vlan20 routing table

Creating the required IP policy rules

The last thing we need to do is to create the required IP Policies to allow the traffic to/from the VLAN on either side of the L2TPv3 interface as well as external Internet access. The rules needed for VLAN10 are shown in figure 3.3.11.

# ^	Name	Log	Src If	Src Net	Dest If	Dest Net	Service	Address Translation
1	▶ NAT_Internet	✓	Vlan10_Lan	Vlan10_net	Wan	all-nets	all_services	SRC:NAT
2	▶ From_Vlan10_L2TPv3	✓	Vlan10_L2TPv3	Vlan10_net	Vlan10_Lan	Vlan10_net	all_services	
3	▶ To_Vlan10_L2TPv3	✓	Vlan10_Lan	Vlan10_net	Vlan10_L2TPv3	Vlan10_net	all_services	

Figure 3.3.11 The IP policy's needed for internal and external access for the VLAN10 interface

These rules allows users behind VLAN10 either locally or beyond the L2TPv3 server access to their own network segment as well as Internet using the Wan interface. We are also using "all_services" which should only be used for testing purposes. For live environments it is recommended to use only the ports/services that should be allowed to/from the various interfaces and networks.

A similar set of IP Policy rules must also be added for VLAN20.

The reason why we do not create IP policy rules for access between VLAN10 and VLAN20 is due to our requirement stated earlier, that communication between separate VLANs should not be allowed in this scenario.

Configuring the London Firewall (client)

Configuring the London Firewall is pretty much identical to configuring the Stockholm Firewall. The only difference is that the L2TPv3 client is used instead of the server when attaching the VLANs. All the details on the how the client should be configured can be found in *Recipe 3.2. Bridging a layer 2 network using L2TPv3 & IPsec*. There will be no changes on how the client is configured.

In order to avoid repeating the same description again we will perform a quick summary of the configuration of the London Firewall.

Creating the routing tables

We create two new routing tables. The routing tables will use Ordering “Only” and will be called Vlan10 and Vlan20.

Creating the VLANs

We create four new VLANs with the properties shown in Table 3.3.12.

Name	Base Interface	Network	Routing Table	Transparent mode	AutoAdd Route
Vlan10_lan	Lan	Vlan10_net	Vlan10	Enabled	Enabled
Vlan10_L2TPv3	L2TPv3_Client	Vlan10_net	Vlan10	Enabled	Enabled
Vlan20_lan	Lan	Vlan20_net	Vlan20	Enabled	Enabled
Vlan20_L2TPv3	L2TPv3_Client	Vlan20_net	Vlan20	Enabled	Enabled

Table 3.3.12

Creating the needed switch and normal routes

As the routes for the switch route has been created automatically by the various interfaces in the previous step, we need to add only the “all-nets” route to each routing table on order for users behind VLAN 10 and 20 to reach the internet using the ISP in London as shown in tables 3.3.13 and 3.3.14.

Routing table Vlan10:

Type	Interface	Network	Gateway
Route	Wan	All-nets	wan_gw
Switch Route	Vlan10_L2TPv3	Vlan10_net	
Switch Route	Vlan10_Lan	Vlan10_net	

Table 3.3.13

Routing table Vlan20:

Type	Interface	Network	Gateway
Route	Wan	All-nets	wan_gw
Switch Route	Vlan20_L2TPv3	Vlan20_net	
Switch Route	Vlan20_Lan	Vlan20_net	

Table 3.3.14

Creating the required IP policy rules

The IP Policies needed are the same as on the Stockholm Firewall and are shown in the below table:

Src iface	Src net	Dest iface	Dest net	Service	Address Translation
Vlan10_Lan	Vlan10_Net	Vlan10_L2TPv3	Vlan10_Net	All_Services	
Vlan10_L2TPv3	Vlan10_Net	Vlan10_Lan	Vlan10_Net	All_Services	
Vlan10_Lan	Vlan10_Net	Wan	All-nets	All_Services	SRC:NAT
Vlan20_Lan	Vlan20_Net	Vlan20_L2TPv3	Vlan20_Net	All_Services	
Vlan20_L2TPv3	Vlan20_Net	Vlan20_Lan	Vlan20_Net	All_Services	
Vlan20_Lan	Vlan20_Net	Wan	All-nets	All_Services	SRC:NAT

Table 3.3.15

Using "All_Services" should only be used for testing purposes and should be restricted once everything is up and running according to specification. Locking down the amount of open ports and protocols is always recommended.

Requirement check

This concludes this recipe. As a last step we will look through the initial requirements to see if they can now be met. We will go through them one by one.

- Clients on each side should be able to communicate with their peers as if they were located locally. Meaning a client located behind VLAN 10 in London should be able to communicate with VLAN 10 in Stockholm.
 - By configuring the switch routes, we tell cOS Core on both London and Stockholm that the network(s) defined on the switch routes exist behind two interfaces, one behind a local physical interface and the other past the L2TPv3 client/server. cOS Core will keep track of the locations of the various host using the CAM table (see *Recipe 3.2. Bridging a layer 2 network using L2TPv3 & IPsec* for more information about the CAM table) and if the host cannot be found locally it will forward the query through the L2TPv3 interface and (hopefully) locate the target host there. cOS Core will then add an entry in the CAM table of where the target host MAC address is located.

We also make sure that the routing is correct (e.g. *Figure 3.3.10*) and our rule policy's (e.g. *Figure 3.3.11*) to allow only the traffic that we want between the transparent group we have defined (the local VLAN and the L2TPv3 client & server interfaces).

Status: Fulfilled.

- Intercommunication between the two VLAN networks should not be allowed, only VLAN10 should be able to talk to VLAN10 either locally or through the L2TPv3 tunnel, same with VLAN20.
 - There are two ways we have restricted communication between VLAN10 and VLAN20. The first restriction was made when the routing tables Vlan10 and Vlan20 (see *Figure 3.3.8* and *Figure 3.3.10*) contain only the routes related to their own VLANs.

The second restriction was in our IP policy rules that allow communication only to/from the VLAN10 network to the VLAN10 network or towards the Internet (see

Figure 3.3.11 and Table 3.3.15)

Status: Fulfilled.

- Clients on each side should have internet access that uses their own firewalls. Clients in London should use the London firewall's ISP to surf the Internet, same in Stockholm. Meaning that Internet access will **not** be forwarded or sent through the tunnel.
 - This is primarily achieved by having the route towards the Internet (the all-nets route) in each VLAN routing table on each site point to the external/Wan interface of the Firewall and not e.g. the L2TPv3 interface (see e.g. *Figure 3.3.8* and *Figure 3.3.10*). The IP policy rules are also enforcing Internet access to only be possible using the external/Wan interface (see e.g. *Figure 3.3.11*).

Status: Fulfilled.

Appendix A - Troubleshooting rules and route problems using Ping Simulations.

Purpose

The purpose of this appendix is to discuss some of the common issues as well as tips and tricks when trying to locate problems related to rules or routing.

We will not describe any specific scenario, except to use a basic setup as base for the examples, but we will list some of the common problems customers have run into as well as a small question and answers section.

We will only be using the CLI as ping simulations are only possible to perform using the CLI.

Detailed Discussion

Ping simulations, what is it?

A ping simulation is basically a rule or route tester. Using the CLI command “ping” with various options we can simulate that a packet goes through rules and routing and then print the result.

This is an extremely powerful “tool” to help determine whether a rule or route triggers as it should or when performing tests to see whether we have any potential “leaks” in the network where we want to make sure that access to a specific network segment is only possible from the desired source.

Scenario environment

In order to make a few examples on how the ping simulation can be used we will use the scenario described in *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario*. Figure A-1 shows a slightly modified schematic from *Recipe 2.3. A basic IPsec Lan to Lan tunnel scenario* scenario but with some additional network details to make the examples easier to follow.

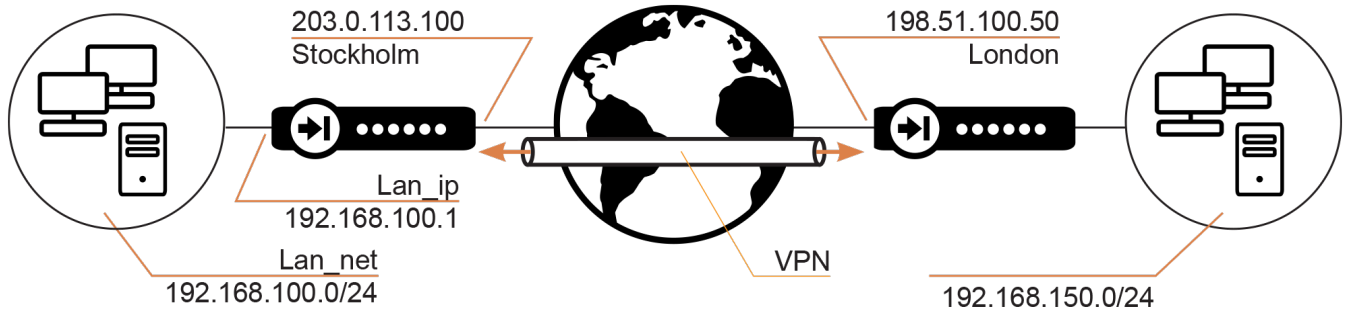


Figure A.1 The environment which we will use for the various ping simulation examples

Rules and routes in Stockholm

We will base the examples of using the CLI in Stockholm. The current rule policies and routing configuration in Stockholm are shown in figure A-2 and A-3.

Rules:

#	Name	Log	Src If	Src Net	Dest If	Dest Net	Service	Address Translation
Rule that allows hosts on Lan to ping all the IP addresses of the Firewall								
1	Ping_Core	✓	Lan	Lan_net	core	all-nets	all_icmp	
Rules that allows traffic towards the Internet								
2	Ping_Internet	✓	Lan	Lan_net	Wan	all-nets	all_icmp	SRC:NAT
3	NAT_To_Internet_Http_all	✓	Lan	Lan_net	Wan	all-nets	http-all	SRC:NAT
4	NAT_To_Internet_Dns	✓	Lan	Lan_net	Wan	all-nets	dns-all	SRC:NAT
Rules related to the IPsec tunnel								
5	Stockholm_To_London	✓	Lan	Lan_net	London_Tunnel	LondonNet	all_services	
6	London_To_Stockholm	✓	London_Tunnel	LondonNet	any	Lan_net	all_services	

Figure A.2 The rule setup on the Stockholm Firewall.

Routes:

# ▲	Type	Interface	Network	Gateway
1	Route IPv4	London_Tunnel	LondonNet	
2	Route IPv4	Wan	Wan_net	
3	Route IPv4	Wan	all-nets	Wan_gw
4	Route IPv4	Lan	Lan_net	

Figure A.3 The current state of the routing table on the Stockholm Firewall

CLI output of the above routes:

```
Stockholm: /> routes
Flags Network          Iface          Gateway          Local IP          Metric
-----
      192.168.150.0/24   London_Tunnel
      203.0.113.0/24    Wan
      0.0.0.0/0         Wan             203.0.113.1
      192.168.100.0/24  Lan
```

Ping simulation parameter list

The CLI command ping has quite an extensive list of options, we will go through them one by one and what they mean. A summary of all the options / parameters is listed below:

1. `-6` Force IPv6.
2. `-count=<1...10>` Number of packets to send. (Default: 1)
3. `-length=<2...8192>` Packet size. (Default: 4)
4. `-pbr=<table>` Route using PBR Table.
5. `-port=<0...65535>` Destination port of UDP or TCP ping.
6. `-srcif=<interface>` Pass packet through the rule set, simulating that the packet was received by `<srcif>`.
7. `-srcip=<ip address>` Use this source IP.
8. `-tcp` Send TCP ping.
9. `-tos=<0...255>` Type of service.
10. `-udp` Send UDP ping.
11. `-verbose` Verbose (more information).
12. `<String>` IP address or URL of host to ping.

1. The “-6” parameter means that we tell the Firewall to treat the ping request as if the network only consists of IPv6. Any IPv4 related route will not be consulted.

2. The “-COUNT” parameter defines the number of packets we want to send. The default is sending one packet but we can send up to 10 in one request. Can be useful in case we are trying to track a packet loss problem in the network.

3. The “-LENGTH” parameter defines the length & size of the packet. If we send a ping without this parameter it will be a 4 byte ICMP packet. The command can be useful in cases where there is an MTU or MSS (segment size) problem in the network and we want to try various packet sizes to try to narrow down the problem.

4. The “-PBR” parameter can be used if we want the route lookup to be done in a specific routing table.



Note

Cannot be combined with the “-SRCIF” parameter because the source interface routing table membership is already determined when we define the routing table.

5. The “-PORT” parameter determines which destination port that should be used for the simulation. This can be very useful when combined with the “-TCP” flag to simulate a packet being sent on e.g. HTTP (TCP port 80). More examples on this further down.
6. The “-SRCIF” parameter is probably the most important parameter. When using this parameter we can get detailed information about which rule and routing the specified traffic passes (or is dropped by). We will go into examples on this further down.
7. The “-SRCIP” is commonly combined with the “-SRCIF” flag to make the ping simulation use a specific IP. This is extremely useful if troubleshooting problems related to a specific machine. If not specified, the default IP will be the interface IP of the sending interface. We will go into examples on this further down.
8. The “-TCP” parameter makes it possible to use “ping” to simulate TCP traffic. Must be combined with the –PORT option. We will go into examples on this further down.
9. The “-TOS” parameter is a way to set the “Type Of Service” flag in a packet. It can be used to send simulation packets with a specific TOS value set in the IP header. It can for instance be used if we want to test systems that can map IP DSCP for traffic prioritization. DSCP is a subset of the DiffServ architecture where the Type of Service (ToS) bits are included in the IP packet header. In cOS Core this is possible to configure using Traffic Shaping Pipe Rules.
10. The –UDP” parameter is similar to “-TCP”, must also be combined with the –PORT option.
11. Using the –VERBOSE” parameter will make cOS Core print additional details regarding the resulting command output. We will be using the “-VERBOSE” flag in all our scenarios/examples below.
12. The “STRING” means basically the target host we want to reach using our simulation. It can be placed pretty much anywhere in the ping syntax and is not order dependent. It can be both IP or an FQDN/DNS name.

Ping simulation example 1

Objective:

Ping a host on the internet using the verbose flag and ICMP.

Ping simulation syntax:

```
Ping 198.51.100.50 -verbose
```

Output:

```
Sending 1 4-byte ICMP ping to 198.51.100.50 from 203.0.113.100  
... using route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "main"  
  
ICMP Reply from 198.51.100.50 seq=0 time= 47 ms TTL=54  
Ping Results: Sent: 1, Received:1, Avg RTT: 47.0 ms
```

Result analyzation:

This simulation is the simplest of them all, we tell the Firewall to send a ping to the target host on the Internet using the verbose flag. The Firewall will perform a route lookup to find the target host and since this host is a public IP address not routed on any of the local interfaces (see figure A-3), it will match the all-nets route (0.0.0.0/0). This route has a default gateway defined so it will also use the Gateway address 203.0.113.1 to reach the target host. Since we do not have any other PBR tables, PBR rules or otherwise the PBR table will in all our examples be "main".

The last two lines contains information regarding the reply (if any) and the time it took for the reply to arrive. In this case the request and reply (put together) took 47 milliseconds to return and the average round-trip time (RTT) was 47.0 ms (as we only sent one packet).

Ping simulation example 2

Objective:

Ping a host on the Internet using the verbose flag, simulate that the traffic was initiated from a secondary IP address on the Firewall.

Ping simulation syntax:

```
Ping 198.51.100.50 -srcip=203.0.113.101 -verbose
```

Output:

```
Sending 1 4-byte ICMP ping to 198.51.100.50 from 203.0.113.100
    ... using route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "ma
ICMP Reply from 198.51.100.50  seq=0  time= 43 ms  TTL=56
    Ping Results:  Sent: 1, Received:1, Avg RTT: 43.0 ms
```

Result analyzation:

Similar to the first ping simulation we send a ping request to the target host, the difference with this simulation is that we are using a different IP sender by specifying the `-SRCIP` parameter. Please note that this only works if the Firewall owns the IP address in question (and replies to ARP etc.). It will not work if you specify a sender that the Firewall cannot use or respond to.

Ping simulation example 3

Objective:

Ping a host on the Internet using the verbose flag, simulate that the traffic was initiated from a host behind the LAN interface.

Ping simulation syntax:

```
Ping 198.51.100.50 -srcif=Lan -srcip=192.168.100.50 -verbose
```

Output:

```
Rule and routing information for ping:
PBR selected by rule "iface_member_main" - PBR table "main"
    allowed by rule "Ping_Internet"

Sending 1 4-byte ICMP ping to 198.51.100.50 from 203.0.113.100
    sent via route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "main"

ICMP Reply from 198.51.100.50  seq=0  time= 10 ms  TTL=56
Ping Results:  Sent: 1, Received:1, Avg RTT: 10.0 ms
```

Result analyzation:

This ping simulation is probably the most common simulation used when troubleshooting. When we combine the `-SRCIF` and `-SRCIP` parameter we can get cOS Core to make a complete rule and route simulation. cOS Core will simulate that a packet is received on a specific interface with a specific sender and even send the packet itself to the target host to see if it will reply (198.51.100.50 in this example), then provide detailed output to the administrator about which rule that triggered. In this case, the "Ping_Internet" rule was the involved rule and since the host is on the Internet, it used the All-nets route and the Wan interface.



Note

In the above simulation we specify a source IP address. The Firewall will pretend that it is this IP address for the simulation. When the return traffic (if the server host replies) arrives from the server to the Firewall, the Firewall pretends to be the specified source IP for simulation purposes. The source IP address host will never be consulted or contacted (which in this example is 192.168.100.50).

Addendum:

Ping simulations show traffic sent through the Firewall. We use the term receive when we talk about the traffic that arrives to the Firewall, rather than the traffic that arrives at the ultimate destination.

Ping simulation example 4

Objective:

Ping a host on the Internet using the verbose flag, simulate that the traffic was initiated from a host behind the LAN interface and that the target service was a HTTP server using port 80 and TCP.

Ping simulation syntax:

```
ping 198.51.100.50 -srcif=Lan -srcip=192.168.100.50 -verbose -tcp -port=80
```

Output:

```
Stockholm:/> Rule and routing information for ping:
                TCP: 192.168.100.50:62265 -> 198.51.100.50:80 PBR selected k
                "iface_member_main" - PBR table "main"
                TCP: 192.168.100.50:62265 -> 198.51.100.50:80
                allowed by rule "NAT_To_Internet_Http_all"

Sending 0-byte TCP ping to 198.51.100.50:80 from 203.0.113.100:62265
  sent via route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "main"

TCP Reply from 198.51.100.50:80 to 192.168.100.50:62265 seq=0  SYN+ACK
time= 10 ms  TTL=53
TCP Reply from 198.51.100.50:80 to 192.168.100.50:62265 seq=0  FIN
time= 20 ms  TTL=53
                TCP Ping Results:  Sent: 1, RST/ACKs Received:1, Loss: 0%, Avg RTT:
ms
```

Result analyzation:

This simulation is an extension of the previous simulation where we add the use of `-TCP` and `-PORT` as well to make a more “life-like” simulation as if a user attempted to surf to a particular host using HTTP port 80. The `-TCP` flag specifies that the protocol should be TCP and the `-PORT` which destination port we want to use.

Since a regular Ping is ICMP it’s not really a ping in this case but cOS Core will go through all the rule and routing information as if it was an HTTP connection from a client located behind the LAN interface. The result is that the ping simulation will verify whether it works or not by trying to complete the 3-way TCP handshake.

It will not be able to make an actual HTTP connection but it can complete the first part of the connection establishment and give the administrator valuable information in the process.

In our example we got a reply from the server where the handshake was partially completed, long enough for the simulation to conclude it is working. The exact handshake process looks like this:

1. Firewall: SYN
2. Server: SYN+ACK
3. Firewall: FIN+ACK
4. Server: FIN+ACK
5. 5. Server: ACK

Since we do not want to complete the handshake we send a FIN+ACK on the third packet to the server to tell the server it can kill the connection. Since the connection is only used for testing there is no reason why the server should keep this connection attempt open so we request that the server close it. Basically a nice way to tell the server to close a connection that is no longer needed.

Ping simulation example 5

Objective:

Verify that traffic initiated from behind the Lan interface can be sent into the IPsec tunnel and that we get a reply.

Ping simulation syntax:

```
ping 192.168.150.50 -srcif=Lan -srcip=192.168.100.100 -verbose
```

Output:

```
Rule and routing information for ping:
      PBR selected by rule "iface_member_main" - PBR table "main"
      allowed by rule "Stockholm_To_London"

Sending 1 4-byte ICMP ping to 192.168.150.50 from 192.168.100.100
  sent via route "192.168.150.0/24 via London_Tunnel, no gw" in PBR table
"main"

ICMP Reply from 192.168.150.50  seq=0  time= 23 ms  TTL=5
      Ping Results:  Sent: 1, Received:1, Avg RTT: 23.0 ms
```

Result analyzation:

In the above syntax we pretend that a client with IP 192.168.100.100 located behind the Lan interface wants to reach 192.168.150.50 which is routed behind the IPsec tunnel interface. The output shows that the IP policy rule "Stockholm_To_London" triggers and allows the traffic as well as the route triggered is using the interface called "London_Tunnel". This is the name of the IPsec interface used to connect the Stockholm to London networks together. The "no gw" means that there is "no gateway" used for this communication as there is no need for a gateway towards a VPN tunnel interface.

And the last entry shows that we got a reply to our ping request, so everything seems to be working fine.

Ping simulation example 6

Objective:

Simulation is similar to the previous syntax in example 5 but with traffic initiated in the other direction. We now want to verify that traffic initiated from beyond the IPsec tunnel can be sent to the Lan interface and that we get a reply.

Ping simulation syntax:

```
ping 192.168.100.100 -srcif=London_Tunnel -srcip=192.168.150.50 -verbose
```

Output:

```
Rule and routing information for ping:
```

```
    allowed by rule "London_To_Stockholm"
```

```
Sending 1 4-byte ICMP ping to 192.168.100.100 from 192.168.150.50
```

```
    sent via route "192.168.100.0/24 via Lan, no gw" in PBR table "main"
```

```
ICMP Reply from 192.168.100.100  seq=0  time=<10 ms  TTL=54
```

```
Ping Results:  Sent: 1, Received:1, Avg RTT: 10.0 ms
```

Result analyzation:

This simulation is example-5 but in reverse, the initiator client is located behind the IPsec interface.

A client with IP 192.168.150.50 located behind the IPsec interface wants to reach 192.168.100.100 which is routed behind the IPsec tunnel interface. The output shows that the IP policy rule "London_To_Stockholm" triggers and allows the traffic as well as the route triggered is using the local interface called "Lan". There is again "no gw" printed because the network we are trying to reach is routed directly behind the Lan interface without going through any gateway.

Ping simulation example 7

Objective:

So far we have only run simulations that works as expected, but the most common thing we want to catch in a simulation is problems. So let's start with a very common problem when working with networking, a typo.

Ping simulation syntax:

```
ping 192.167.100.100 -verbose
```

Output:

```
Sending 1 4-byte ICMP ping to 192.167.100.100 from 203.0.113.100  
... using route "0.0.0.0/0 via Wan, gw 203.0.113.1" in PBR table "main"  
Ping Results: Sent: 1, Received:0, Loss: 100%
```

Result analyzation:

This is a bit odd, why would traffic to the internal network go out on the internet instead of the LAN interface? The problem is a typo in the ping simulation itself, we used 192.167.x instead of 192.168.

The result is that the cOS Core was unable to find this network on either the LAN interface or the IPsec tunnel and then the all-nets route matched and cOS Core forwarded the request to its default gateway. If we are unlucky we may get a reply in the event that the IP address that we used by mistake, is used by something that actually responds and that would cause confusion if we were not observant.

We would also have received a similar result had the typo been on the route itself. Meaning even if we were to enter the correct IP address on the ping command, the ping would be sent to the Internet route as the Firewall was unable to find this network behind any of the local or IPsec interfaces.

Ping simulation example 8

Objective:

A log entry everyone will run into sooner or later is a log that contains "Default_Access_Rule". For this example we will give a sample on how we can get this entry and what it means. For easier understanding please have a look at the CLI output under figure A-3 as reference.

Ping simulation syntax:

```
ping 192.168.150.1 -srcif=Lan -srcip=192.168.101.10 -verbose
```

Output:

```
Rule and routing information for ping:  
PBR selected by rule "iface_member_main" - PBR table "main"  
    DROPPED by rule "Default_Access_Rule"
```

Result analyzation:

If we first look at the syntax we see that for this sample we introduced a mistake in the ping simulation. We used the source IP 192.168.101.10 instead of 192.168.100.10 which is the network routed behind the Lan interface.

We tell the Firewall to simulate that traffic is received on the Lan interface with the source IP 192.168.101.10 and it gives us the output of what would happen if that were the case. And it tells us that the result is that it is dropped by the "Default_Access_Rule". This is actually a really good log entry to get because it is **always** a routing problem and it is very easy to explain why this is happening.

It means that the packet received on the receiving interface (in this case the Lan interface) is **not** routed on this interface. When examining logs, always look for the source/receiving interface where the packet was received, it always means that the Firewall did not expect this packet to arrive on this interface based on the routing and drops it due to the "Default_Access_Rule".

If we look at the output result and match it against the routing CLI output in figure A-3, we see that the Firewall expects that the source interface should be Wan and not Lan.

Addendum:

Ping simulations show traffic sent through the Firewall. We use the term "receive" when we talk about the traffic that arrives at the Firewall, rather than the traffic that arrives at the ultimate destination.

Ping simulation example 9

Objective:

A second log message that is also very common is called "Default_Rule" or "Default_Drop_Rule". This example will trigger this message and we will go through what it means.

Ping simulation syntax:

```
ping 198.51.100.50 -srcif=Lan -srcip=192.168.100.50 -tcp -port=8080 -verbose
```

Output:

```
Rule and routing information for ping:
TCP: 192.168.100.50:11815 -> 198.51.100.50:8080 PBR selected by rule
      "iface_member_main" - PBR table "main"
      TCP: 192.168.100.50:11815 -> 192.36.125.18:8080 DROPPED by rule "Default_Rule"
```

Result analyzation:

In this simulation we attempt to ping a host on the Internet from the internal machine 192.168.100.50 located behind the Lan interface. We also want to use TCP and port 8080, the resulting output is that this is not allowed but instead dropped by a rule called "Default_Rule".

The "Default_Rule" is a hidden rule that triggers if no IP rule/policy can match the traffic we want to simulate. If we look at the ruleset shown in Figure A-2 we see that from the Lan interface we allow pinging of the Firewall, Ping towards internet, making DNS queries and to surf the web using HTTP and HTTPS but not to connect to the Internet on port 8080. However, had the destination network we wanted to reach been the network routed behind the IPsec tunnel, the traffic would have been allowed by the rule called "Stockholm_To_London".

The problem is that there is no matching IP rule/policy created that allows the client to connect to port 8080 on the target IP address. Some administrators create a specific drop rule at the end of each ruleset that drops all traffic instead of the "Default_Rule" being triggered. A common

name for such a rule is "DropAll". The reason why we would like to create/use such a rule is in situations where we have multiple rulesets and it will be that much easier to separate of we know exactly which of the Drop rules that triggered for troubleshooting purposes. Having a drop on a rule called e.g. "Drop_Vlan100" is a lot easier to track/spot than drops that are done by the "Default_Rule".

Ping simulation example 10

Objective:

The last simulation example output is of the more unusual kind but if we are dealing with a more complicated routing environment, it may be encountered.

Ping simulation syntax:

```
ping 198.51.100.50 -verbose
```

Output:

```
ping: No route to 198.51.100.50
```

Result analyzation:

What happened here is that when cOS Core attempted to find a route where the target IP is located is was unable to find any matching route at all. The easiest example of when we can get this is if we do not have a default route (all-nets) in the routing table the ping simulation is using. As an example we will get this message if we remove route #3 as shown in Figure A-3.

Questions and Answers

Question:

Why do we need to specify source interface for the simulation to get rule details? Is it not enough to just specify the source IP?

Answer:

No, if we only specify the source IP cOS Core will assume this IP address belongs to cOS Core itself. If we specify both source IP and source interface (also known as receiving interface) cOS Core knows it should simulate that a packet was received on the specified interface and then go through both the route and rule process and print the resulting output.

Question:

When I used a ping simulation the resulting output was very strange, it said that the destination interface was Core and the result is always 100% packet loss even though I know the target host is alive.

Answer:

The most likely reason for this output is because the ping simulation goes through an IP rule or Policy that uses an ALG (Application Layer Gateway). Since an ALG acts as a kind of proxy the ping simulation output will be unable to properly display the resulting output. This is a limitation in the ping simulation. An example on how it can look:

```
ping 198.51.100.50 -srcif=Lan -srcip=192.168.100.50 -verbose -tcp -port=80
```

```
Rule and routing information for ping:
```

```
TCP: 192.168.100.50:16491 -> 198.51.100.50:80 PBR selected by rule
      "iface_member_main" - PBR table "main"
```

```
TCP: 192.168.100.50:16491 -> 198.51.100.50:80
      allowed by rule "NAT_To_Internet_Http_all"
```

```
Sending 0-byte TCP ping to 198.51.100.50:80 from 192.168.98.24:16491
sent via route "0.0.0.0 via core, no gw" in PBR table "main"
```

```
TCP Ping Results:  Sent: 1, RST/ACKs Received:0, Loss: 100%
```

The simulation can still be useful as the data regarding which rule that triggers can be of interest but the route information is invalid as the real destination interface would not be Core.

Appendix B – IPsec IKE version 1 tunnel setup flowchart.

Purpose

The purpose of this flowchart is to give an rough understanding of the various phases and steps needed for an IKEv1 tunnel negotiation using a pre-shared key (with additional option examples).

It can be very useful when troubleshooting a tunnel to know what kind of settings and options are verified in each phase. An example would be if the tunnel fails in phase-2 we do not have to bother spending time on verifying if the ID list or IKE proposal list has a match as that part of the negotiation has already been completed before we end up in phase-2.

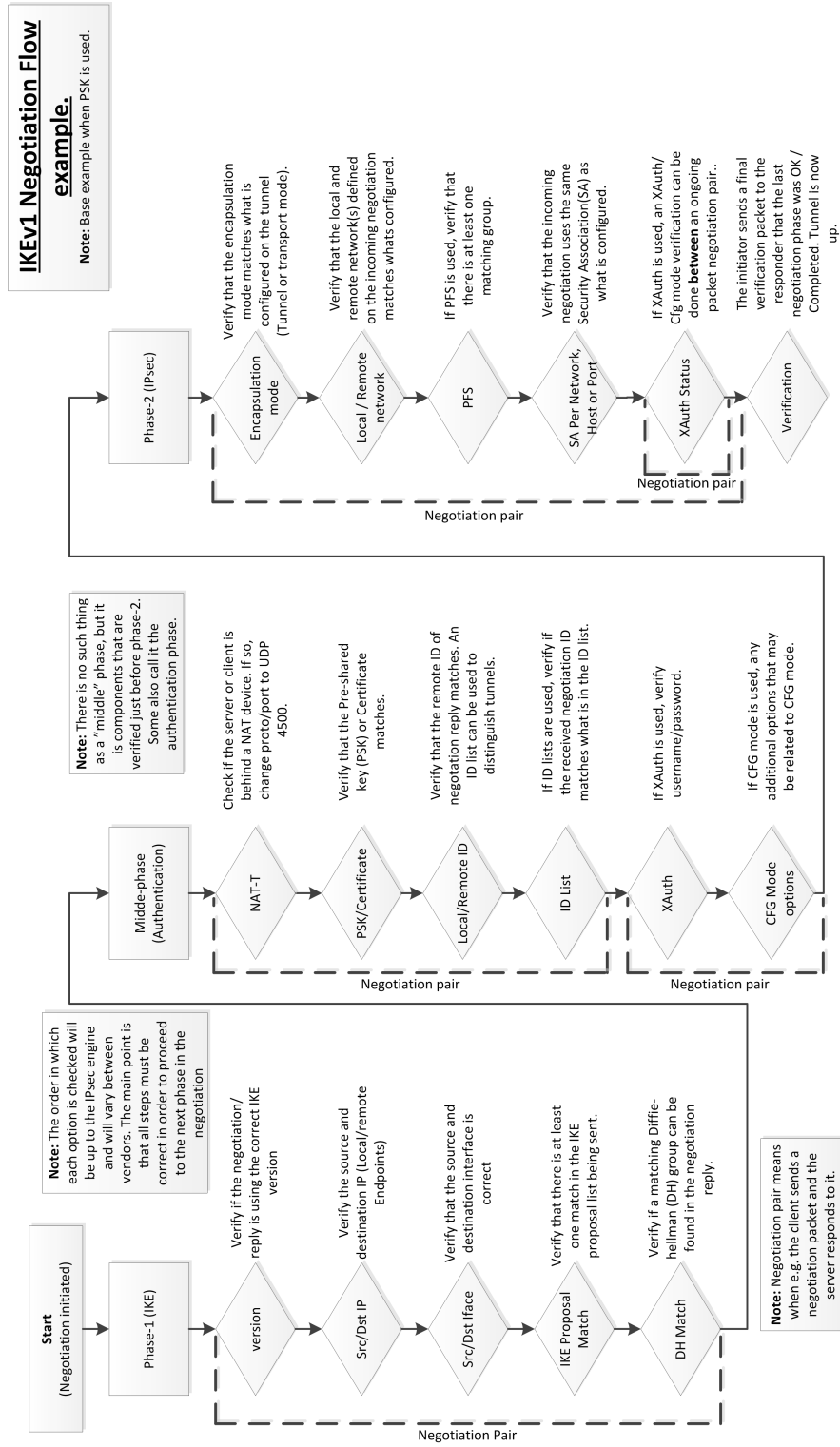


Figure B.1 An example of an IKEv1 tunnel setup using Pre-shared key

Afterword

Thoughts from the Author

We have now reached the end of the second cOS Core Cookbook. I hope that you have found it useful in furthering your knowledge of all things VPN related. The number of scenarios that exists for VPN and in particular IPsec is very vast, and we have only covered some of the more common cases. With the help of the information provided in this Book, I hope that you will be able to adapt the knowledge to even more advanced scenarios.

Some may wonder why we do not cover L2TP, PPTP or SSL VPN in the Book. The first two mentioned protocols are using technologies that are becoming old and obsolete, and some major vendors have already discontinued the use of PPTP. I suspect that L2TP is soon to join that category as well. SSL VPN however, is still a fully valid VPN type and we will consider adding it to a future edition of the Book depending on popular demand.

And last, but not least, don't forget that ping simulations are your friend when troubleshooting.

Best regards,
Peter Nilsson.

Alphabetical Index

A

Appendix A *196, 214*

C

Certificate

CA *112*

Certificate request *128*

Chains *114*

Host Certificate *115*

Importing and Exporting *120*

Local and Remote *120*

Root Certificate *115*

Self-signed *117*

Certificate Authority *112*

Certificates *34, 111*

Config mode pool *73*

Content Addressable Memory *177*

D

Diagrams *8*

E

Encryption *9*

H

HQ setup *93*

I

ID Lists *116, 132*

IKE *17*

IKE negotiation *21*

IKEv2 *66*

Internet through IPsec *142*

Introduction *7*

IPsec *15*

Certificates *34*

- Connection between local and remote net 101
- IKE 17
- IKEv2 66
- Lan to Lan 43
- Local ID 24
- Local network 24
- Phase-1 21, 49
- Phase-2 22, 53
- Properties 24
- Remote Endpoint 28
- Remote ID 24
- Remote network 24
- Roaming 69
- SA 19
- Transport Mode 26
- Tunnel Mode 25
- IPsec explained 16
- IPsec negotiation 22

K

- Key Distribution 11

L

- L2TPv3 159
 - Passthrough options 161
 - Points of interest 160
 - VLAN 181
- Local User Database 77

N

- NAT Traversal 36

P

- Ping simulation 196
 - Parameters 199
- Planning 10

R

- Road Warrior 69

S

- Screenshots 8
- Security Associations 19

Split tunneling 86
Switch routes 164

T

Transparent mode 164

U

User Authentication 78

V

Virtual routing 152
VLAN 185
 VLAN ID 186
VPN introduction 9

The Clavister

cOS Core VPN cookbook

Only the tastiest recipes

The second anticipated release of the Clavister cOS Core Cookbook, as edited by network security expert Peter Nilsson, describes some of the basics of encryption and the use of Virtual Private Networks (VPN) to unlock this critical Clavister solution for customers.

The book's focus is on how VPN can be used in some of the most common scenarios with a strong emphasis on VPN tunnels that use IPsec. It will also describe how Layer 2 tunneling protocol version 3 (L2TPv3) can be used together with transparent mode over an IPsec connection as well as a few tips and tricks on how to troubleshoot rule and routing problems on both physical and VPN interfaces. Taken together, the Cookbook offers structured solutions and improvements to show why Clavister solutions are so well regarded.

CLAVISTER[®]

CONNECT • PROTECT

