CCNP Security SECURE Notes

Private Vlans:

```
vtp mode transparent

vlan 600
private-vlan community

vlan 400
private-vlan isolated

vlan 200
private-vlan primary
private-vlan association 400,600

int gi1/0/13
switchport mode private-vlan host
switchport private-vlan host-association 200 400

int range gi1/0/14 – 15
switchport mode private-vlan host
switchport private-vlan host-association 200 600

int gi1/0/16
switchport mode private-vlan promiscuous
switchport private-vlan mapping 200 400,600
```

Verification Commands:

```
show vlan private-vlan type
show vlan private-vlan
```

PVLAN Edge:

```
int gi1/0/18
switchport protected
```

The PVLAN edge (protected port) is a feature that has only local significance to the switch (unlike Private Vlans), and there is no isolation provided between two protected ports located on different switches. A protected port does not forward any traffic (unicast, multicast, or broadcast) to any other port that is also a protected port in the same switch. Traffic cannot be forwarded between protected ports at L2, all traffic passing between protected ports must be forwarded through a Layer 3 (L3) device.

uRPF (Unicast Reverse Path Forwarding):

Strict uRPF:

When a packet arrives, it extracts the SOURCE IP ADDRESS and searches the FIB to locate the network. If the found location does NOT match the INPUT interface (the interface on which the packet arrived), then the packet is DROPPED.

Loose uRPF:

Like strict, except that the found location can match ANY interface… as long as there is a route in the FIB, it is happy!

```
ip cef

int fa0/0
ip verify unicast source reachable-via [any | rx] [allow-default] [allow-self-ping]
[acl]
```

reachable-via any = Loose uRPF

reachable-via rx = Strict uRPF

allow-default = Use default route for matching. If used in combination with "reachable-via any", it totally invalidates the configuration and is pointless.

acl = Specify ACL name or number to use with uRPF

Verification Commands:

```
show ip cef summary
show ip int fa0/0  (look for "IP verify source …")
```

Guidelines:

- Symmetric routing and FIB trusted – use Strict uRPF
- Asymmetric routing or FIB untrusted – use anti-spoofing ACLs
- Eliminate bogon-sourced traffic – use Loose uRPF
- Don't use default route with Loose uRPF

Netflow:

- High-level reporting, diagnostics, and anomaly detection
- Similar to granularity of a cell phone bill
- Telemetry is pushed to Netflow collectors
- Variety of metrics are reported
- **Flow is defined by 7 keys:**
  SRC IP, DST IP, SRC PT, DST PT, L3 PROTO TYPE, COS/TOS, IFINDEX
- Version 5 is common, Version 9 is the future (and supports Flexible Netflow)

```
flow exporter RICHARD_EXPORTER
destination 192.168.2.10
transport udp 9996
export-protocol [netflow-v5 | netflow-v9]

flow monitor RICHARD_MONITOR
record netflow ipv4 original-input            (use classic netflow)
exporter RICHARD_EXPORTER                     (the name you used above)

int fa0/0
ip flow monitor RICHARD_MONITOR input
```

Control Plane Security:

(other planes include Management Plane and Data Plane)

The Control Plane is a logical part of the router that performs base functions such as building the routing and forwarding tables (dynamic routing protocols, PIM, HSRP, non-IP protocols such as ARP, IS-IS, etc.)

Protection includes: iACLs, CoPP, CPPr, and Routing Protocol Auth/Filtering

iACLs (Infrastructure ACLs):

Legacy technology – if not using CoPP/CPPr, you should at least use iACLs
They are usually applied INBOUND on interfaces facing the user, or an external network (e.g. the Internet).

CoPP (Control Plane Policing):

Configuration Example:

```
ip access-list extended MATCH-OSPF
permit ospf 10.0.0.0 0.255.255.255 any

class-map COPP-CLASS
match access-group name MATCH-OSPF

policy-map COPP-POLICY
class COPP-CLASS
police rate 250 pps conform-action transmit exceed-action drop
class class-default
police rate 10 pps conform-action transmit exceed-action drop

exit (back to global configuration mode)

control-plane (host is default, other options are cef-exception and transit)
service-policy input COPP-POLICY
```

CPPr (Control Plane Protection):

Although it is similar to Control Plane Policing (CoPP), CPPr has the ability to restrict/police traffic using **finer granularity** than that used by CoPP.

CPPr divides the aggregate control plane into three separate control plane categories, known as sub-interfaces: (1) host, (2) transit, and (3) cef-exception

CPPr configuration same as CoPP, except applied to all sub-interfaces!

Verification Commands:

```
show policy-map control-plane host
```

<< KNOW THIS!!!

FPM (Flexible Packet Matching) – uses PHDF files:

Configuration Example:

```
load protocol flash:ip.phdf
load protocol flash:tcp.phdf
```

```
class-map type stack match-all FPM-STACK-CLASS
match field ip protocol eq 0x6 next tcp
```

^^^ In English: we are matching ip protocol equal to 6, which is TCP – also notice the class-map is of type STACK and we are using MATCH-ALL (logical AND), instead of the default MATCH-ANY (logical OR)…

```
class-map type access-control match-all FPM-AC-CLASS
match field tcp dest-port eq 80
match start tcp payload-start offset 0 size 256 regex "[cC][mM][dD]\.[eE][xX][eE]"
```

^^^ In English: we are using **RegEx** to look at the TCP payload starting at offset 0 through 256, and we are looking for any case variation of "cmd.exe" – also notice the class-map is of type ACCESS-CONTROL and we are again using MATCH-ALL…

```
policy-map type access-control FPM-POLICY
class FPM-AC-CLASS
drop
```

```
policy-map type access-control FPM-POLICY-FINAL
class FPM-STACK-CLASS
service-policy FPM-POLICY
```

^^^ We are referencing the first policy-map we created above by calling that policy-map from within the second policy-map. We have now "glued" everything together.

```
int s0
service-policy type access-control input FPM-POLICY-FINAL
```

Routing Protocol Authentication:

…this further helps to secure our Control Plane

What are the bad guys doing?
Trying to spoof neighbor relationships! They may even send masqueraded updates to corrupt routing tables. We are going to leverage HMACs (hashes) as a countermeasure.

RIPv2 supports **Plain Text** and **MD5**
EIGRP supports **MD5**
OSPF supports **Plain Text** and **MD5** (and v3 supports **AH**)
BGP supports **MD5**

*Configuration Example for **RIPv2**:*

```
key chain RICHARD
key 1
key-string mypassword
```

You can create multiple keys, and you can use these options as well:

```
accept-lifetime 12:00:00 Jan 1 2011 11:59:59 Dec 31 2011
send-lifetime 12:00:00 Jan 1 2011 11:59:59 Dec 31 2011
```

Apply this in **INTERFACE** configuration mode:

```
int s0
ip rip authentication mode md5             (or text, which is plain text)
ip rip authentication key-chain RICHARD
```

*Configuration Example for **EIGRP**:*

… you can use the same key chain we created above

```
int s0
ip authentication mode eigrp 100 md5
ip authentication key-chain eigrp 100 RICHARD
```

*Configuration Example for **OSPF (Plain Text)**:*

```
int s0
ip ospf authentication
ip ospf authentication-key mypassword
```

*Configuration Example for **OSPF (MD5)**:*

```
int s0
ip ospf authentication message-digest
ip ospf message-digest-key 1 md5 mypassword
```

You can selectively turn on authentication for specific areas:

```
router ospf 1
area 5 authentication message-digest          -OR-          area 5 authentication
```

*Configuration Example for **BGP**:*

… no key chains used for BGP!

Apply this in **BGP ROUTING PROCESS** configuration mode:

```
router bgp 64512
neighbor 10.0.0.1 remote-as 64512
neighbor 10.0.0.1 password mypassword
```

*Configuration Example for **HSRP (Plain Text)**:*

```
standby 1 authentication mypassword
```

*Configuration Example for **HSRP (MD5)**:*

```
standby 1 authentication md5 key-string mypassword
```

Helpful debugs:

```
debug rip events
debug eigrp packets
debug ip ospf adj
debug standby errors (for HSRP)
```

Management Plane Security:

Access control for **VTY lines** can be accomplished using the **access-class** command, or via CoPP/CPPr.

Configuration Example for VTY/SSH Access Control (using access-class):

```
ip access-list standard RESTRICT-SSH
permit 10.0.0.0 0.0.0.255
deny any log
```

```
line vty 0 15
transport input ssh
access-class RESTRICT-SSH in
```

Access control can also be applied to SNMP traffic.

Configuration Example for SNMP Access Control:

```
ip access-list extended RESTRICT-SNMP
permit udp 10.0.0.0 0.0.0.255 any eq 161
deny ip any any log
```

```
snmp-server community secretcommunitystring ro RESTRICT-SNMP
```

Now we will look at access control for VTY/SSH and SMTP via CoPP/CPPr. We can restrict the traffic to a particular interface we designate as a **MANAGEMENT INTERFACE**.

Configuration Example for CoPP/CPPr Access Control:

```
ip access-list extended RESTRICT-SSH-SNMP
permit tcp 10.0.0.0 0.0.0.255 any eq 22
permit udp 10.0.0.0 0.0.0.255 any eq 161
```

```
class-map CPPR-CLASS
match access-group name RESTRICT-SSH-SNMP
```

```
policy-map CPPR-POLICY
class CPPR-CLASS
police rate 50 pps conform-action transmit exceed-action drop
```

```
control-plane host
service-policy input CPPR-POLICY
```

MPP (Management Plane Protection):

Configuration Example (same as above, but under control-plane host add this):

```
control-plane host
management-interface fa0/1 allow ssh snmp
```

RBAC Views (Role Based Access Control):

Configuration Example:

```
aaa new-model
```

(AAA MUST BE ENABLED FIRST!!!)

Enter the ROOT VIEW to start the configuration:

```
enable view
```

… then under global configuration mode …

```
parser view MYVIEW
secret mypassword
```

Commands Syntax:
commands exec [exclude | include | include-exclusive] …

```
commands exec include show access-list
commands exec include show running-config
commands exec include configure terminal

commands configure include ip access-list extended
commands configure include ip access-list standard

commands ipenacl include all deny
commands ipenacl include all permit
```

SNMP Views:

These views restrict access to ONLY certain MIB parameters!

Configuration Example:

```
snmp-server view MYVIEW interfaces included
snmp-server group MYGROUP v3 priv read MYVIEW access RESTRICT_SNMP

snmp-server user RDAVIS MYGROUP v3 auth sha mypassword priv aes 128 mysecretkey
```

There are SNMP INFORMS and TRAPS.
INFORMS are superior to traps!

```
snmp-server host 10.0.0.50 traps version 3 priv RDAVIS
snmp-server enable traps
```

CPU / Memory Thresholds:

Configuration Example (CPU):

```
process cpu threshold type total rising 80 interval 10
```

Configuration Example (Memory):

```
memory free low-watermark processor 8000
```

Zone-Based Firewalls:

Configuration Example:

```
ip access-list extended OUT-TO-IN-ACL
permit tcp host 184.75.249.120 host 172.16.10.99 eq 9997
permit ip 172.16.30.0 0.0.0.255 any

ip access-list extended OUT-TO-SELF-ACL
permit udp any eq bootps any eq bootpc
permit ahp any any
permit esp any any
permit udp any any eq isakmp
permit udp any any eq non500-isakmp
permit 41 any any
permit icmp host 184.75.249.120 any
permit ip 172.16.30.0 0.0.0.255 any

class-map type inspect match-any IN-TO-OUT-CLASS
match protocol tcp
match protocol udp
match protocol icmp

policy-map type inspect IN-TO-OUT-POLICY
class type inspect IN-TO-OUT-CLASS
inspect
class class-default
drop

class-map type inspect match-any OUT-TO-IN-CLASS
match access-group name OUT-TO-IN-ACL

policy-map type inspect OUT-TO-IN-POLICY
class type inspect OUT-TO-IN-CLASS
inspect
class class-default
drop
```

```
class-map type inspect match-any OUT-TO-SELF-CLASS
match access-group name OUT-TO-SELF-ACL

policy-map type inspect OUT-TO-SELF-POLICY
class type inspect OUT-TO-SELF-CLASS
pass
class class-default
drop log

zone security INSIDE
zone security OUTSIDE

int tunnel0
zone-member security INSIDE

int vlan10
zone-member security INSIDE

int vlan20
zone-member security INSIDE

int virtual-template1
zone-member security OUTSIDE

int fa4
zone-member security OUTSIDE

zone-pair security IN-TO-OUT source INSIDE destination OUTSIDE
service-policy type inspect IN-TO-OUT-POLICY

zone-pair security OUT-TO-IN source OUTSIDE destination INSIDE
service-policy type inspect OUT-TO-IN-POLICY

zone-pair security OUT-TO-SELF source OUTSIDE destination self
service-policy type inspect OUT-TO-SELF-POLICY
```

Verification Commands:

```
show class-map type inspect
show policy-map type inspect
show policy-map type inspect zone-pair
show zone security zonename (ex. INSIDE, OUTSIDE, self)
```

Advanced ZFW Features:

Configuration Example for Application Layer Filtering in ZFW:
… we are looking for any variation of “cmd.exe” in HTTP traffic

```
class-map type inspect match-all OUT-TO-DMZ-CLASS
match protocol http

parameter-map type regex CMD-REGEX
pattern [cC][mM][dD]\.[eE][xX][eE]

class-map type inspect http match-any OUT-TO-DMZ-APP-CLASS
match request arg regex CMD-REGEX
match req-resp protocol-violation

policy-map type inspect http OUT-TO-DMZ-APP-POLICY
class type inspect http OUT-TO-DMZ-APP-CLASS
reset
log
```

Now we are NESTING this policy map inside the already existing policy:

```
policy-map type inspect OUT-TO-DMZ-POLICY
class type inspect OUT-TO-DMZ-CLASS
service-policy http OUT-TO-DMZ-APP-POLICY
```

**** BE SURE TO REVIEW THE “IOS TREND CONTENT FILTERING” PDF! ****

IOS IPS:

This is not IDS -- it’s IPS because the router, by its very nature, is in the packet-forwarding path. It’s really a “mini” version of the full Cisco IPS (the full IPS being 4240 or 4260, for example).

IDS – generates ALERTS regarding offending traffic
IPS – can take action to mitigate the threat

Events can be sent to the:
CCP (Cisco Configuration Professional)
IME (Cisco IPS Manager Express)

IOS IPS is SIGNATURE-BASED – it can look at packet headers or payloads, and can generate alerts or take evasive action.

There are some limitations to this SIGNATURE-BASED approach – the IPS cannot detect an attack it doesn’t know about – the signature has to be there and enabled

Signatures need to be updated on a regular basis!
Demands an on-going TUNING process of the signatures!

Signature Engines provide IOS IPS functionality:
Atomic IP Engine, String Engine, Normalizer Engine (can MODIFY packets INLINE! – like IP fragmentation or TCP segmentation), **Other Engine**

These are a subset of all the engines you would have in a FULL-BLOWN SENSOR – you might call them "MICRO ENGINES"

Can have FALSE POSITIVES, FALSE NEGATIVES, TRUE POSITIVES, TRUE NEGATIVES

SEAP (Signature Event Action Processor):
This is a function that allows you to use the CCP to manage false positives, generate address-based filters, generate global actions based on risk rating

IOS IPS BEST PRACTICES:

**** IOS should be 15.0(1)M or higher and proper license*
**** Deploy IPS at **EDGE** of network – leave DISTRIBUTION and CORE alone!*
**** Need centralized monitoring solution (CCP/IME or Cisco Security Manager)*
**** Best for SMALL/MEDIUM businesses*

STEP 1 – You need to prep the router for IPS by importing Cisco's PUBLIC Key:
The IPS updates are digitally signed with Cisco's private key and can be verified using this public key:

crypto key pubkey-chain rsa
named-key realm-cisco.pub signature
key-string
PASTE HEX KEY HERE

… before continuing, you need to obtain the PKG file from Cisco – requires expensive license … "copy tftp://x.x.x.x/IOS-S480-CLI.pkg **idconf**" – "idconf" is a flash alias you must use as the destination for the signature package …

STEP 2 – Create/Apply named IPS Rule Set:

ip ips config location flash://*dirname* (e.g. iosips)
ip ips name *MYIPSNAME*
ip ips *MYIPSNAME* in
ip ips *MYIPSNAME* out

Verification Commands (after signatures have been loaded):
show ip ips signatures
show ip ips signatures count

STEP 3 – Use CCP to selectively enable the signatures you need

… at this point the video shows screenshots from CCP
in CCP click Security >> Intrusion Prevention

STEP 4 – TWEAK, TWEAK, TWEAK!

Event Risk Rating Formula (returns INT 0 -- 100):

$$ERR = \frac{ASR \times TVR \times SFR}{10,000}$$

ASR (Attack Severity Rating) has values of: **25**, **50**, **75**, or **100**
(Informational, Low, Medium, High)
Potential amount of damage that can be done
Assigned by Cisco, can be customized

TVR (Target Value Rating) has values of: **50**, **75**, **100**, **150** or **200**
(0 Rating, Low, Medium, High, Mission Cr.)
Must be manually configured
Should be part of a Risk Assessment
Could assign a value based on IP range

SFR (Signature Fidelity Rating) has values of: **0 – 100**
Accuracy of the signature
Assigned by Cisco / Trend Micro
Confidence author has in accuracy of signature

Event Action Overrides (a feature of SEAP):
This is somewhat of a misnomer. It does not actually OVERRIDE, it **APPENDS** to the already configured action. This is done based on the **ERR VALUE** – could be things like "***Deny Packet Inline***" if the Risk Rating (ERR) is between 90 – 100. *Must be enabled GLOBALLY by checking the "USE EVENT ACTION OVERRIDES" box!*

To reduce ***FALSE POSITIVES***, you can create an ***EXCEPTION*** for a management Vlan or the like … can be done for all signatures, a range, or a specific one …

Verification Commands (for Event Action Overrides):

```
show ip ips event-action-rules target-value-rating
show ip ips event-action-rules overrides
show ip ips event-action-rules filters
```

Enabling SDEE (Security Device Event Exchange):

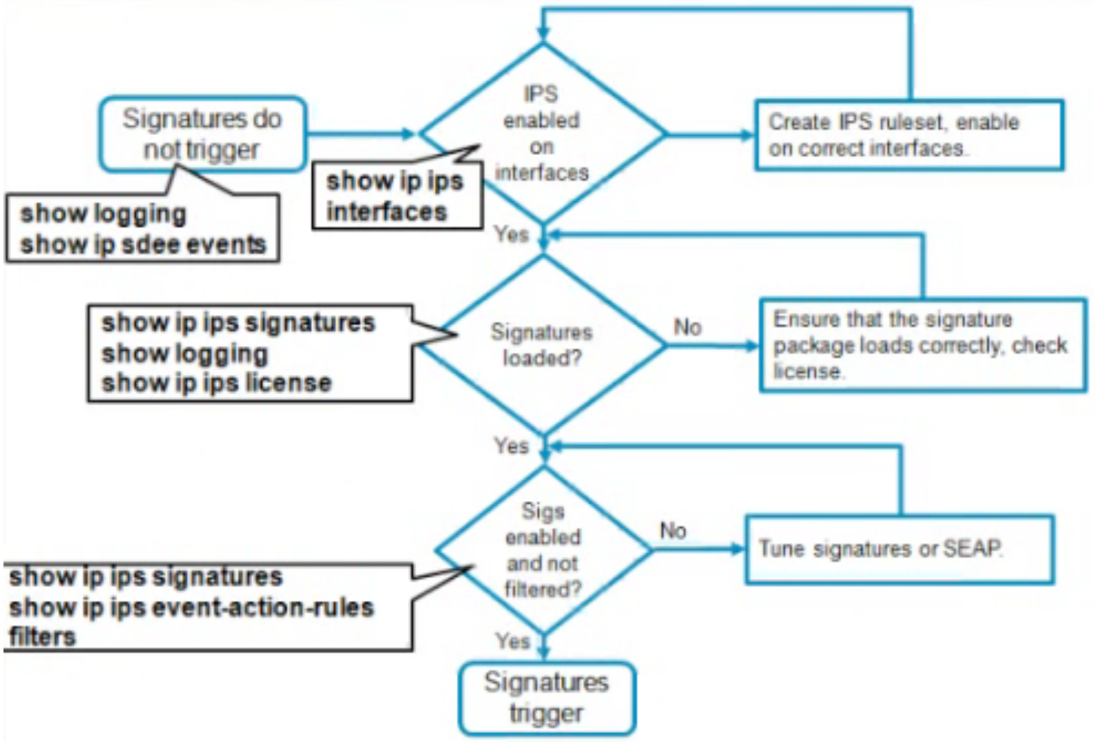
This is a proprietary Cisco protocol for IPS … CCP or IME can PULL SDEE events …

Configuration Example:

```
aaa new-model
aaa authentication login default local
aaa authorization exec default local
ip http server
ip http secure-server
ip http authentication aaa
username davisrg secret whatever
```

```
ip ips notify sdee
show ip sdee alerts
```

Study this flowchart and the show commands listed:


Signatures do not trigger
show logging
show ip sdee events
show ip ips interfaces
IPS enabled on interfaces
Create IPS ruleset, enable on correct interfaces.
Yes
show ip ips signatures
show logging
show ip ips license
Signatures loaded?
No
Ensure that the signature package loads correctly, check license.
Yes
show ip ips signatures
show ip ips event-action-rules
filters
Sigs enabled and not filtered?
No
Tune signatures or SEAP.
Yes
Signatures trigger

Last section shows the IME (IPS Manager Express) …

Site-to-Site VPN Review (see ISCW notes):

IPsec = security framework, NOT protocol

Composed of multiple protocols:

ESP: **Encapsulating Security Payload provides:**
Encryption (Confidentiality)
Data Origin Authentication
Data Integrity
Anti-Replay Protection

AH: **Authentication Header provides:**
Data Origin Authentication
Data Integrity
Anti-Replay Protection
AH = Mostly Obsolete!

IKE: **Internet Key Exchange:**
Negotiates the security parameters and authentication keys

IKE Phase 1 (Main Mode, Aggressive Mode) – BIDIRECTIONAL:

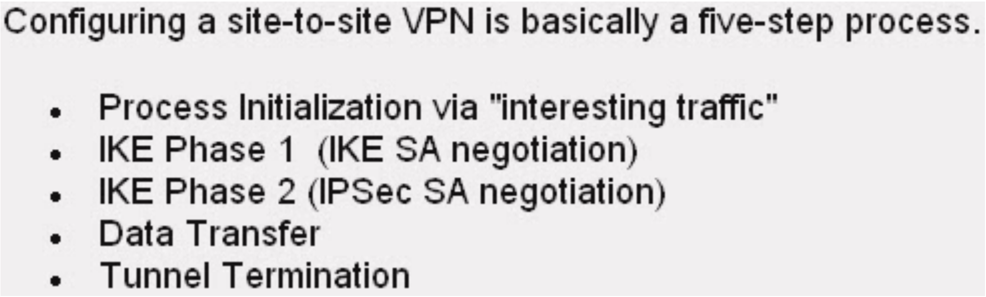
Main Mode provides protection of data but cannot be used when dynamic addressing of clients is required

Aggressive Mode is faster, does not protect data, and must be used when dynamic addressing of clients is required

Phase 1 results in the creation of a Security Association (SA) for ISAKMP itself

IKE Phase 2 (Quick Mode) – UNIDIRECTIONAL:

Quick Mode is the ONLY mode, sets up IPsec SA



Configuring a site-to-site VPN is basically a five-step process.

- Process Initialization via "interesting traffic"
- IKE Phase 1 (IKE SA negotiation)
- IKE Phase 2 (IPSec SA negotiation)
- Data Transfer
- Tunnel Termination

ISAKMP Policy Parameters:

Authentication:	pre-share, rsa-sig, rsa-encr
Encryption:	des, 3des, aes
Hash:	md5, sha
Group:	1 (768-bit) 2 (1024-bit) 5 (1536-bit) 14 (2048-bit) 15 (3072-bit) 16 (4096-bit)
Lifetime:	Seconds / Kb **(ONLY value that does NOT have to match between peers, lowest used)**

STATIC P2P VTIs (Virtual Tunnel Interfaces):

- IPsec session isn't mapped to interfaces – no crypto maps!
- VTI is a routable virtual interface with an **IPsec Profile** assigned to it
- Supports ESP or AH, multicast, scalable
- Simple, flexible, routable
- IP ONLY
- No IOS Software stateful failover

Configuration Example:

STEP 1 – Create ISAKMP Policies on both endpoints:

```
crypto isakmp policy 10
authentication pre-share
hash sha
encryption aes 128
group 14
lifetime 3600
```

(… with the other side having a matching policy)

Verification Commands:
show crypto isakmp policy
show crypto isakmp sa *(once both sides are fully configured – **QM_IDLE + ACTIVE**)*

STEP 2 – Create pre-shared key:

```
crypto isakmp key thisismysecretkey address 172.17.2.2
```

(… with the other side being 172.17.2.1)

STEP 3 – Create IPsec Transform Set:

```
crypto ipsec transform-set TSET esp-aes 128 esp-sha-hmac
```

… if you don’t create a transform set, a default will be used

STEP 4 – Create IPsec Profile:

```
crypto ipsec profile PROFILE1
set transform-set TSET
```

STEP 5 – Create Virtual Tunnel Interface:

```
interface Tunnel0
ip unnumbered gi0/0
tunnel source gi0/0
tunnel destination 172.17.2.2
tunnel mode ipsec ipv4
tunnel protection ipsec profile PROFILE1
```

STEP 6 – Configure Routing:

```
ip route 10.1.2.0 255.255.255.0 Tunnel0
```

^^^ We are routing network 10.1.2.0/24 through the tunnel

DYNAMIC P2P VTIs (Virtual Tunnel Interfaces):

Good for Hub and Spoke networks!!!

Configuration Example:

STEPS 1 - 4 ARE THE SAME AS ABOVE

Now we aren’t creating a Tunnel0 interface, rather a Virtual-Template interface …

STEP 5 – Create Crypto Keyring:

```
crypto keyring KR1
pre-shared-key address 172.17.2.2 key thisismysecretkey
pre-shared-key address 172.17.2.3 key thisismysecretkey
pre-shared-key address 172.17.2.4 key thisismysecretkey
```

STEP 6 – Create Virtual Template Interface:

```
interface Virtual-Template1 type tunnel
ip unnumbered gi0/0
tunnel mode ipsec ipv4
tunnel protection ipsec profile PROFILE1
```

STEP 7 – Create ISAKMP Profile:

```
crypto isakmp profile ISAKMP_PROFILE1
keyring KR1
match identity address 172.17.2.2 255.255.255.255
match identity address 172.17.2.3 255.255.255.255
match identity address 172.17.2.4 255.255.255.255
virtual-template 1
```

Scalable VPN Authentication:

Steps to configure Cisco IOS Certificate Server:

1. **Create Dedicated RSA Keys *(Optional, But HIGHLY Recommended)***

   ```
   crypto key generate rsa label CSKEYS modulus 2048 exportable
   ```

2. **Create PKI Trustpoint**

   ```
   crypto pki trustpoint CS
   rsakeypair CSKEYS
   ip http server
   ```

3. **Create Certificate Server**

   ```
   crypto pki server MYCS
   issuer-name CN=MYCS, OU=VPN, O=SHANNON, C=US
   ```

4. **Locate Database**

   ```
   database url flash://mycs
   database level complete
   ```

5. **Configure Issuing Policy**

   ```
   hash [md5 | sha1 | sha256 | sha384 | sha512]
   hash sha1
   lifetime certificate 730
   lifetime ca-certificate 3650
   no grant auto
   ```

6. **Configure Revocation Policy**

   ```
   lifetime crl 5
   ```

7. **Configure SCEP Interface**

   You MUST **ENABLE HTTP SERVER** (only PKCS #10 can be used otherwise)

8. **Enable Certificate Server**

   ```
   no shutdown
   ```

Verification Commands:
show crypto pki server *MYCS*

Best Practices:

- Security and time-sync is critical
- Key/cert management is main vulnerability
- VPN-only PKI is best – or subordinate ca
- Have solid cert granting procedures
- Consider remote secure storage for files
- Use “show crypto pki server” to troubleshoot
- Use IOS software PKI client if possible

Steps to configure Cisco IOS PKI Client:

1. **Create RSA Keypair**
   crypto key generate rsa label *VPNKEYS* modulus 2048 exportable

2. **Create PKI Trustpoint**
   crypto pki trustpoint *CS*
   enrollment url http://192.168.2.77 << this is the CA server’s IP
   rsakeypair *VPNKEYS*

3. **Accept Certificate (Authenticate Root CA)**
   crypto pki authenticate *MYCS*

4. **Enroll W/ CA**
   crypto pki enroll *MYCS*
   [optionally enter password]
   [subject name will be router’s name]
   [include router’s S/N: YES]
   [include router’s IP? NO]
   [request certificate from CA? YES]

5. **Approve Pending Request On CA**
   show crypto pki server *MYCS* requests
   crypto pki server *MYCS* **grant** *1* << “1” refers to the ReqID from above cmd
   -- OR --
   crypto pki server *MYCS* **reject** *1*
   -- OR –
   crypto pki server *MYCS* **revoke** *1*

Back to the requesting router →
show crypto pki certificates *MYCS*
… it will now show up

Troubleshooting:

```
debug crypto isakmp
debug crypto pki transactions
show crypto ca certificates
```

To use certs with VPN, you need to make sure you have an IKE Phase 1 policy that supports RSA signatures:

show crypto isakmp policy:
The DEFAULT PROTECTION SUITE supports rsa-sig
…but you should create a new one!

Next, we need to create a certificate map:

```
crypto pki certificate map MYCERTMAP 10
subject-name co ou=VPN, o=SHANNON, c=US
```

Next, we need to configure an ISAKMP profile:

```
crypto isakmp profile MYISAKMP
match certificate MYCERTMAP
ca trust-point MYCS
```

Next, we need to configure an IPSEC profile:

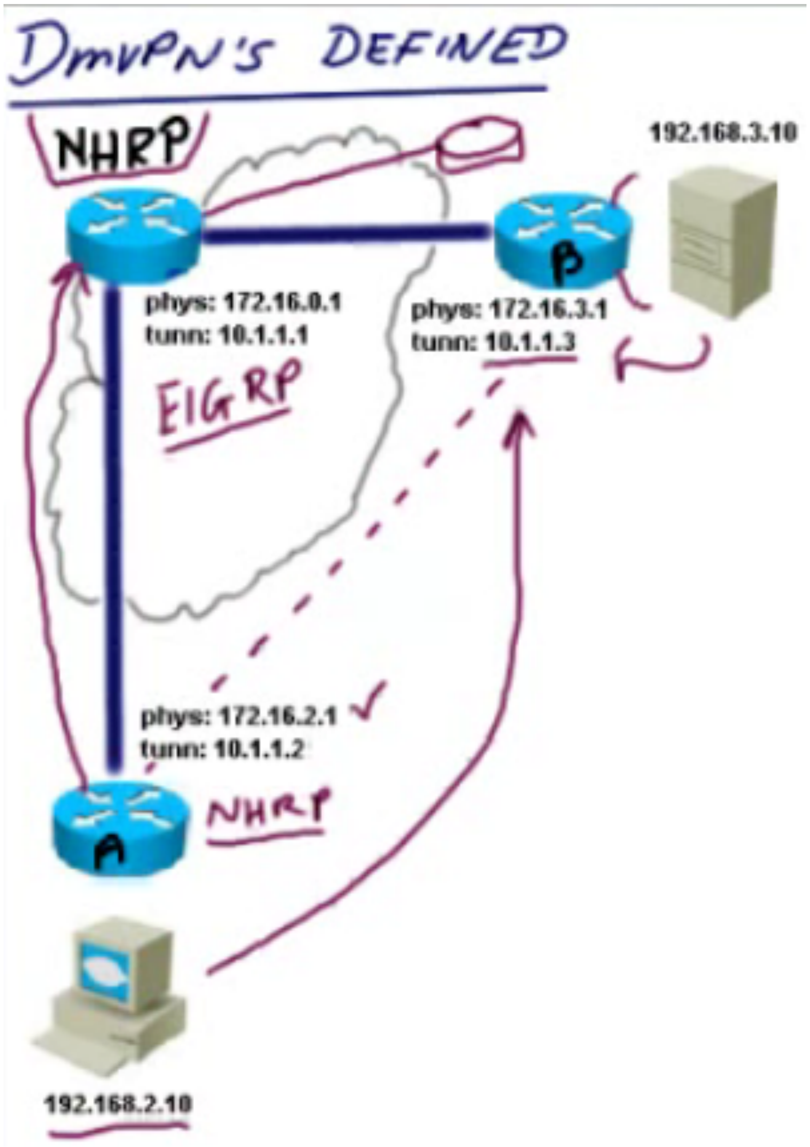
```
crypto ipsec profile MYIPSEC
set isakmp-profile MYISAKMP
```

Finally, apply the tunnel protection to the tunnel interface:

```
interface Tunnel0
tunnel protection ipsec profile MYIPSEC
```

DMVPNs (Dynamic Multipoint VPNs):

- Dynamically creates GRE over IPsec tunnels on demand (mGRE)
- Uses **NHRP (Next Hop Resolution Protocol)** to build the tunnels
- **EIGRP** recommended for DMVPN dynamic routing
- Works great in hub and spoke topology
- Very scalable, creation on ad-hoc basis
- Not much configuration, especially on hub
- **Need to use PKI authentication for scalable solution, NOT PSK**



The DMVPN server is the NHRP server, which is called NHS

In the diagram above, if HOST A (192.168.2.10) were sending an HTTP GET request towards web server HOST B (192.168.3.10), HOST A would look in its IP ROUTING TABLE and find a next hop IP of 10.1.1.3 for HOST B.

HOST A would then consult its local NHRP cache for a matching entry, find none, and send an NHRP query towards the NHS (DMVPN Server). The NHS would specify 172.16.3.1, the physical interface, as a reply to that query. HOST A would add that to its local NHRP cache. Now the spoke can create a direct IPsec tunnel across the cloud to SPOKE B. The source IP for the IKE session is 172.16.2.1, and the destination would be 172.16.3.1. IPsec SA's could then be formed… now all UNICAST packets will bypass the hub and go straight to SPOKE B. MULTICAST traffic will always go through the NHS.

This is a unidirectional tunnel from SPOKE A to SPOKE B until SPOKE B performs the same steps during its HTTP response.

Configuration Example (DMVPN Hub):

```
int Tunnel1
tunnel mode gre multipoint              << mGRE
tunnel source 192.168.2.77              << the PHYSICAL IP
ip address 10.1.1.1 255.255.255.0       << the TUNNEL IP
```

Configuration Example (DMVPN Spoke):

```
int Tunnel1
tunnel mode gre ip                      << standard GRE, not multipoint (P2P)
tunnel source 192.168.2.2
tunnel destination 192.168.2.77
ip address 10.1.1.2 255.255.255.0
```

-- end configuration example for simple NON full mesh P2P --

Configuration Example (DMVPN Hub):

```
Int Tunnel1
tunnel mode gre multipoint
tunnel source gi0/2
tunnel key 12345
ip nhrp network-id 1
ip nhrp authentication mysecretkey
ip nhrp map multicast dynamic           << routing protcls won't work w/out this!
tunnel protection ipsec profile MYPROFILE   << created in previous example above
ip address 10.1.1.1 255.255.255.0
ip mtu 1400
ip tcp adjust-mss 1360
no ip next-hop-self eigrp 1
no ip split-horizon eirgp 1
```

Configuration Example (DMVPN Spoke):

```
int Tunnel1
tunnel mode gre multipoint
tunnel source fa0/1
tunnel key 12345
ip nhrp network-id 1
ip nhrp authentication mysecretkey
ip nhrp map multicast 192.168.2.77      << routing protcls won't work w/out this!
ip nhrp nhs 10.1.1.1                    << specify the NHS!
ip nhrp map 10.1.1.1 192.168.2.77       << create static mapping spoke to hub
tunnel protection ipsec profile MYPROFILE
ip address 10.1.1.2 255.255.255.0
```

ip mtu 1400
ip tcp adjust-mss 1360

GET VPN:

- First available in 12.4(11)T
- Offers large scale protection utilizing existing routing infrastructure
- Tunnel-free for MPLS VPN, IP, FR, ATM
- Hardware acceleration recommended
- Scales easily
- Utilizes new variant of IKE called **IKE GDOI (Group Domain of Interpretation)**
- **GDOI = UDP 848**
- **Minimum AES-128, SHA-1 HMAC recommended**
- **Does NOT need NTP (uses pseudo-time)**

The **hub** is a **GCKS (Group Controller Key Server)**
The **spokes** are called **Group Members**
GCKS pushes into to Group Members
Group Members use **IKE GDOI** to register with GCKS

GDOI has 2 types of keys generated by GCKS:

- **TEK (Traffic Encryption Key)**
- **KEK (Key Encryption Key)** – protects re-keying between key server / members

The GCKS maintains a group table. When the Group Members register, the key server adds them to the correct table.

2 types of re-keying methods:

- **Unicast**
- **Multicast (highly recommended)**

When it's time for the GCKS to send out the multicast keys to the group members, it will generate a SINGLE MULTICAST RE-KEY PACKET to the ENTIRE DOMAIN. The backbone/core will replicate it for all the GROUP MEMBERS (several times). There is NO ACK, which is good.

How is GET VPN Tunnel-less?
It uses **IPsec Tunnel Mode with IP Header Preservation** – ORIGINAL source/destination address is preserved. Existing L3 info can be used. **The original IP information is copied up to another header…**

Downside to GET-VPN? If the key is compromised – bad times. The attacker could then decrypt any traffic.

Configuration Example for ***GET VPN GCKS (Group Controller Key Server)***:

STEP 1 – Create new IKE policy:

```
crypto isakmp policy 10
authentication pre-share
encryption aes 128
group 14 …
```

STEP 2 – Create keys that will be used by Group Members:

```
crypto isakmp key mysecretkey1 address 192.168.2.2
crypto isakmp key mysecretkey2 address 192.168.2.3
crypto isakmp key mysecretkey3 address 192.168.2.4
```

STEP 3 – Create RSA keys for KEK:

```
crypto key generate rsa modulus 2048 label GETKEYS [exportable]
```

STEP 4 – Create IPsec Transform Set / IPsec Profile:

```
crypto ipsec transform-set GETSET esp-aes esp-sha-hmac

crypto ipsec profile GETPROFILE
set transform-set GETSET
```

STEP 5 – Create Crypto ACL to identify interesting traffic:

```
ip access-list extended GETACL
permit ip 10.1.0.0 0.0.255.255 10.1.0.0 0.0.255.255
```

STEP 6 – Create GET VPN (GDOI) Group:

```
crypto gdoi group GETGROUP
identity number 12345              << this GET VPN cloud, members must match
server local
address ipv4 192.168.2.77          << outside IP
rekey transport unicast            << OVERRIDE DEFAULT OF MULTICAST!
rekey authentication mypubkey rsa GETKEYS
sa ipsec 10
profile GETPROFILE
match address ipv4 GETACL
```

STEP 7 – Create GET VPN (GDOI) Crypto Map:

```
crypto map GETMAP 10 gdoi
set group GETGROUP

int gi0/2
crypto map GETMAP
```

Verification Commands:

```
show crypto gdoi
```

*Configuration Example for **GET VPN Group Member**:*

STEP 1 – Create new IKE policy (MATCHING A GCKS PROFILE):

```
crypto isakmp policy 10
authentication pre-share
encryption aes 128
group 14 …
```

STEP 2 – Configure key that matches GCKS PSK:

```
crypto isakmp key mysecretkey1 address 192.168.2.77
```

STEP 3 – Create GET VPN (GDOI) Group:

```
crypto gdoi group GETGROUP
identity number 12345                  << this GET VPN cloud, members must match
server address ipv4 192.168.2.77
```

STEP 4 – Create GET VPN (GDOI) Crypto Map:

```
crypto map GETMAP 10 gdoi
set group GETGROUP

int fa0/1
crypto map GETMAP
```

GET VPN Redundancy:

- Up to 8-node co-ops (clusters)
- Delivers keys, redundancy, load balancing
- Non-preemptive election for primary (kind of like default HSRP)
- Primary creates and pushes keys to co-ops
- Members can register to any key server and point to several servers
- Beware of network splits – auto merge

SSL VPN:

SSL was created by Netscape in 1994

Remote Access VPN Technologies:

- **Clientless SSL VPN** (no client install, fewer firewall issues than IPsec)
- **Full Tunnel SSL VPN**
- **Full Tunnel IPsec VPN** (best for low latency operations)

IETF updated/refreshed SSL to TLS

TLS 1.0 == SSL 3.1

- Authenticates server to the client using X509 Digital Certificates
- Can *optionally* authenticate client to the server via same method
- Negotiates common algorithms / shared secrets
- Will establish protected tunnel for TCP/UDP connections

3 Phase Negotiation Process (kind of like a 3-way handshake):

Phase 1: Negotiation of parameters between client and server

Phase 2: One-way / mutual authentication between client and server

Phase 3: Generate session key and activate cipher suite

User Authentication:	Static OTP or RSA Certificates
Server Authentication:	RSA Certificates ONLY
Cisco SSL VPN Supported Protocols:	SSL 3.0 or TLS 1.x

SSL Integrity / Packet Authentication Algorithm:

- **SHA-1 HMAC**

SSL Encryption Algorithms:

- **RC4** (symmetric stream cipher, used by WEP also)
- **AES-128** (symmetric block cipher, recommended)
- **3DES** (symmetric block cipher)

*Configuration Example for **SSL VPN Gateway**:*

STEP 1 – Create RSA keys:

```
crypto key generate rsa modulus 2048 label SSLKEYS
```

STEP 2 – Create PKI trustpoint:

```
crypto pki trustpoint SSLCS
rsakeypair SSLKEYS
```

STEP 3 – Create PKI server:

```
crypto pki server SSLCS
issuer-name CN=SSLCS, OU=VPN, O=DAVIS, C=US
database url flash://sslcs
database level complete
hash sha1
lifetime cert 730
lifetime ca-cert 3650
lifetime crl 12
no grant auto
no shut
[enter passphrase when prompted]
```

STEP 4 – Create gateway:

The GATEWAY defines the fundamental network and crypto parameters used by server … IP address, port, trustpoint, logging. Then we create a CONTEXT to customize the portal for the user.

```
webvpn gateway SSLVPNGW
ip address 192.168.2.77 port 443        << address end users will specify in client
ssl trustpoint SSLCS
ssl encryption 3des-sha1 aes-sha1       << acceptable client encryption types
logging enable                          << enable syslog
inservice                               << make it active!
```

STEP 5 – Create context:

```
webvpn context CONTEXT1
gateway SSLVPNGW
inservice                               << make it active!
```

Verification Commands:

```
show webvpn license
show webvpn gateway
dir flash:
```

Basic User Authentication / Full Tunneling:

We have to specify the location of the Cisco AnyConnect client that will be made available to our clients:

```
webvpn install svc flash://anyconnect-win-2.1.0148-k9.pkg
```

Now we will create some local users:

```
aaa new-model
aaa authentication login LOCALVPN local
username vpnuser secret cisco
```

… and a local pool of IPs for the VPN users:

```
ip local pool MYPOOL 192.168.2.200 192.168.2.210
```

Now we go back to the webvpn context we created in Step 5:

```
webvpn context CONTEXT1
policy group MYPOLICY
default-group-policy MYPOLICY
aaa authentication list LOCALVPN        << created above
policy group MYPOLICY                   << enter the policy to make changes
banner "Welcome to the SSL VPN!"
functions svc-enabled                   << enable FULL TUNNEL
svc address-pool MYPOOL
svc default-domain mydomain.local
svc keep-client-installed
dns-server 208.67.222.222 208.67.220.220
… tons of other policy-specific commands like "homepage" can be issued here …
```

*Configuration Example for **Clientless SSL VPN Gateway**:*

```
ip host site.cisco.com 192.168.2.77
webvpn context CONTEXTCL                << new context for clientless config
url-list "MY-WEB-BOOKMARKS"             << preconfigured client bookmarks
url-txt "Intranet Server" url-value "http://13cubed.com"
policy group CLPOLICY
url-list "MY-WEB-BOOKMARKS"
```

Other notes:

To use Cisco Secure Desktop:
webvpn install csd flash://*securedesktop-ios-3.1.1.45-k9.pkg*

… then back to the context:
csd enable

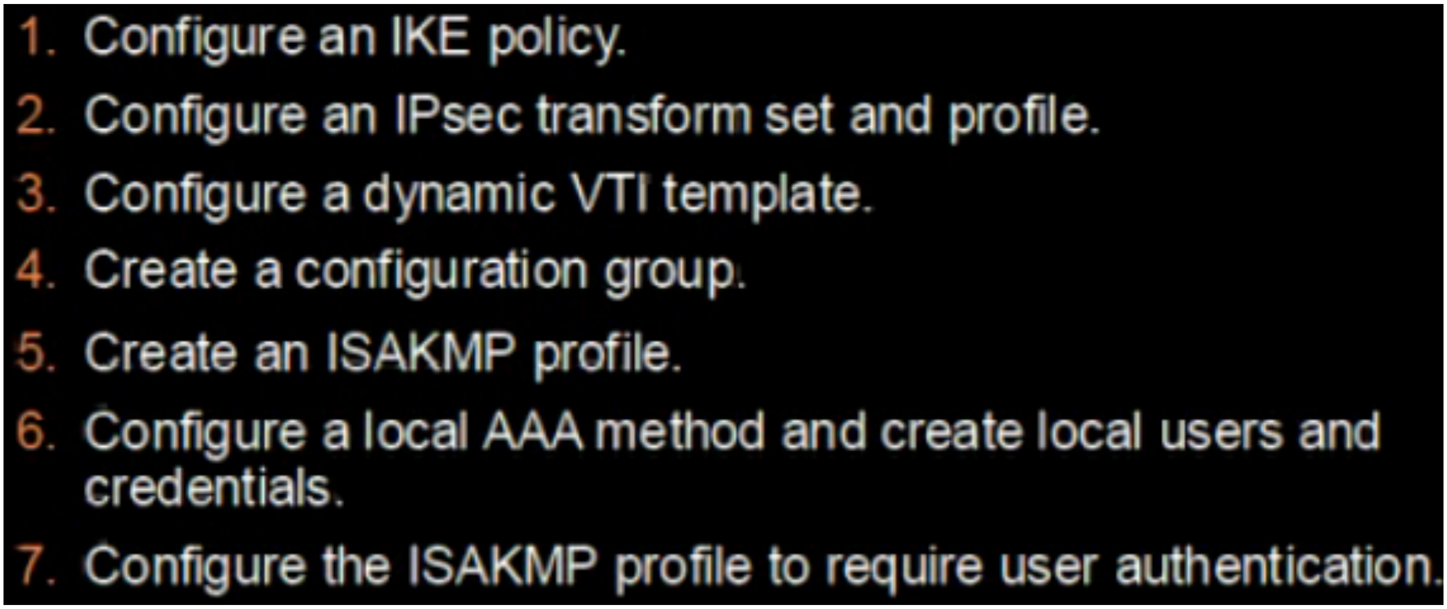
Cisco Easy VPN Server (IPsec-based):

What is **Cisco Easy VPN**?
A flexible **site-to-site** AND **remote access** VPN solution!

Using **Cisco Easy VPN Remote** (used on the REMOTE side), we can make it easy for the clients to connect because most of the settings can be defined on the **Cisco Easy VPN Server** (ISR or ASA) and PUSHED to the client!

- The client is going to authenticate the ISR with a group password
- The ISR is going to authenticate the client based on a group password and an *optional* username/password (from a local database) – this is called **Xauth**
- Can use old school crypto maps or newer VTIs (Virtual Tunnel Interfaces)
- Does NOT support AH, only ESP

Overview of Easy VPN Server configuration steps:



1. Configure an IKE policy.
2. Configure an IPsec transform set and profile.
3. Configure a dynamic VTI template.
4. Create a configuration group.
5. Create an ISAKMP profile.
6. Configure a local AAA method and create local users and credentials.
7. Configure the ISAKMP profile to require user authentication.

STEP 1 – Configure an IKE policy:

Not shown, same as always …

STEP 2 – Configure an IPsec transform set and profile:

```
crypto ipsec transform-set EZSET esp-aes esp-sha-hmac
```

… remember, ESP ONLY – NO AH!

```
crypto ipsec profile EZPROFILE
set transform-set EZSET
```

STEP 3 – Configure a dynamic VTI template:

```
interface virtual-template1 type tunnel
ip unnumbered fa0/0
tunnel mode ipsec ipv4
tunnel protection ipsec profile EZPROFILE
```

STEP 4 – Create a client configuration group:

```
ip local pool EZPOOL 10.10.10.1 10.10.10.100

ip access-list extended EZSPLIT                        << split tunnel ACL
permit ip 10.0.0.0 0.255.255.255 any

crypto isakmp client configuration group EZGROUP       << EXAM
key nuggetlab
dns 208.67.222.222 208.67.220.220
ip pool EZPOOL
acl EZSPLIT
```

STEP 5 – Create an ISAKMP profile (and bind to IPsec profile):

```
aaa authorization network LOCAL_AUTHOR local

crypto isakmp profile EZISAPROFILE
match identity group EZGROUP
isakmp authorization list LOCAL_AUTHOR
client configuration address respond
client configuration group EZGROUP
virtual-template 1

crypto ipsec profile EZPROFILE          << back to IPsec profile
set isakmp-profile EZISAPROFILE         << bind ISAKMP profile to IPsec profile
```

STEP 6 – Configure a local AAA method and create local users:

```
aaa authentication login LOCAL_AUTHEN local
username ezuser secret nuggetlab
```

STEP 7 – Configure the ISAKMP profile to require user authentication:

```
crypto isakmp profile EZISAPROFILE             << back to ISAKMP profile
client authentication list LOCAL_AUTHEN        << Xauth!
```

**Just FYI – all this can be done in the GUI via the Cisco Configuration Professional*

Verification Commands:

```
show crypto session username ezuser
```

Cisco Easy VPN Remote Modes:

- **Client Mode**
  NAT/PAT, separate VPN address space like I have at home
- **Net Extension**
  PCs and other hosts at client side of VPN should have fully routable and reachable addresses – reachable over the tunnel by the head end – one big logical network
- **Net Extension Plus** (Cisco Proprietary)
  Same as Net Extension, except you can use IKE Mode Configuration to request/automatically assign to an available loopback interface (for troubleshooting)

Overview of Easy VPN Client configuration steps:

STEP 1 – Configure a dynamic VTI template and local user, apply to interfaces:

```
interface virtual-template1 type tunnel
tunnel mode ipsec ipv4
```

```
crypto ipsec client ezvpn EZCLIENT
group EZGROUP key nuggetlab
virtual-interface 1
peer 172.16.2.2
mode client
```

```
username vpnuser secret mypassword
```

```
int gi0/0
crypto ipsec client ezvpn EZCLIENT outside
```

```
int gi0/1
crypto ipsec client ezvpn EZCLIENT inside
```

-- End --