RCOMP - Redes de Computadores (Computer Networks)

2021/2022

Lecture 10

- Network security and security layers, SSL/TLS and IPsec.
- Virtual Private Networks, types and technologies, PPP, L2TP, PPTP and SSTP.

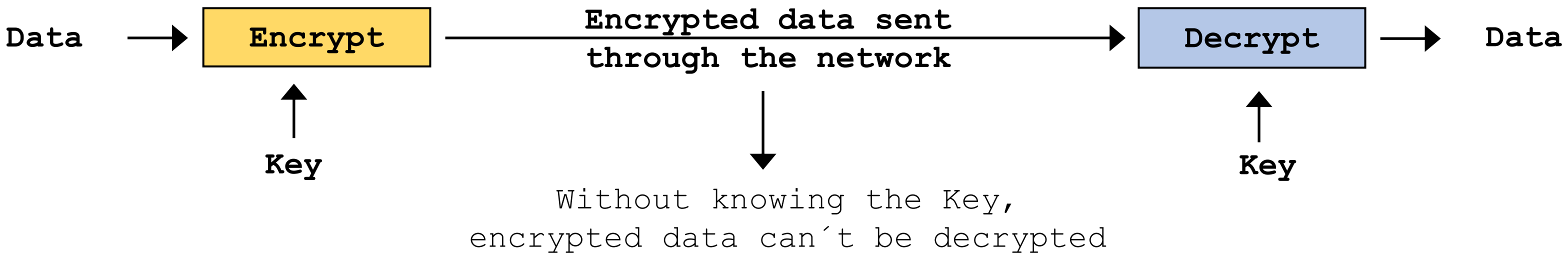
Network security

Transferring data between two applications throughout a network presents a serious security risk. This is so because networks are public and any attacker can access them, a few exceptions may be pointed out, like for instance a DMZ. Even so, to play safe we should always regard networks as unsecure. To secure transactions throughout a network, two crucial issues must be encompassed:

- **Authentication** – Guaranteeing the application is talking with the intended counterpart application, thus, avoiding attacks known as man-in-the-middle (MITM).
- **Privacy –** Guaranteeing packets transferred through the network are not readable by attackers (sniffing attacks). Because controlling the physical access to a network is in most cases impossible, the way to solve this issue is by encrypting data.

Also, it's pointless enforcing privacy without prior authentication. The issue is, if the dialogue is taking place with an intruder impersonating the intended counterpart (MITM), then the data encryption schema will be established with the intruder, and not with the intended counterpart, thus the intruder is able to decrypt all data, and there's no privacy.

Encryption algorithms (cyphers) are public; however, they operate by using a key, and such key may be kept secret:



Cyphers

Cyphers are algorithms used to encrypt and decrypt data, in fact each encompasses two operations, one reverses the other to obtain the original data. Even tough cyphers are public, they use keys, in such a way without the suitable key, decryption is impossible.

Having said that, an attacker could guess the key. Although specific weaknesses of the cyphers may be exploited, ultimately there's one infallible method: trying all possible keys. This is usually referred to as "**brute force attack**". In simple terms, by picking a chunk of encrypted data, the attacker would try all possible keys until getting something that makes sense.

So, **any encrypted data can be decrypted by an attacker**, the point is, how long will it take, and that, beyond the cypher itself, depends on the key size. So the longer the key size, the longer will it take. Each cypher uses a fixed key size, and this is one factor used to establish the cypher's strength. Nowadays cyphers using key sizes bellow 128-bits are regarded as unsafe.

One attack to encrypted network transactions may take place without actually guessing the key, is called the **replay attack**. The attacker will simple record the encrypted data sent to an application, later, sends the same exact data to the same application. Even without knowing what that encrypted data means, expects it will be accepted by the application as before.

There are several strategies to avoid replay attacks, one most straightforward is including timestamps on every network transaction.

Cyphers themselves have a strategy to avoid this issue. When a cypher is used for the first encryption operation, beyond the key, they also require an **initialization vector** (IV), and the same IV is required for decryption. So when establishing a safe session between two applications, before starting to use the cypher, both applications should settle a common random IV, unique, and for that session only.

Block cyphers and stream cyphers

Some cyphers operate block by block, each block having a fixed size, they are called block cyphers (e.g., DES, IDEA, RC5, Blowfish, and Rijndael/AES).

Block cyphers receive a data block to be encrypted and produce an encrypted block of data, of course if using the same key, two equal input blocks will produce the same encrypted output block. In networking practice, data is a flow, splitting that data in blocks and encrypting blocks one by one is a bad idea, this is called Electronic Codebook (ECB). The issue is, when an attacker sees two equal blocks of encrypted data, he would know they correspond to a pair of equal blocks of unencrypted data.

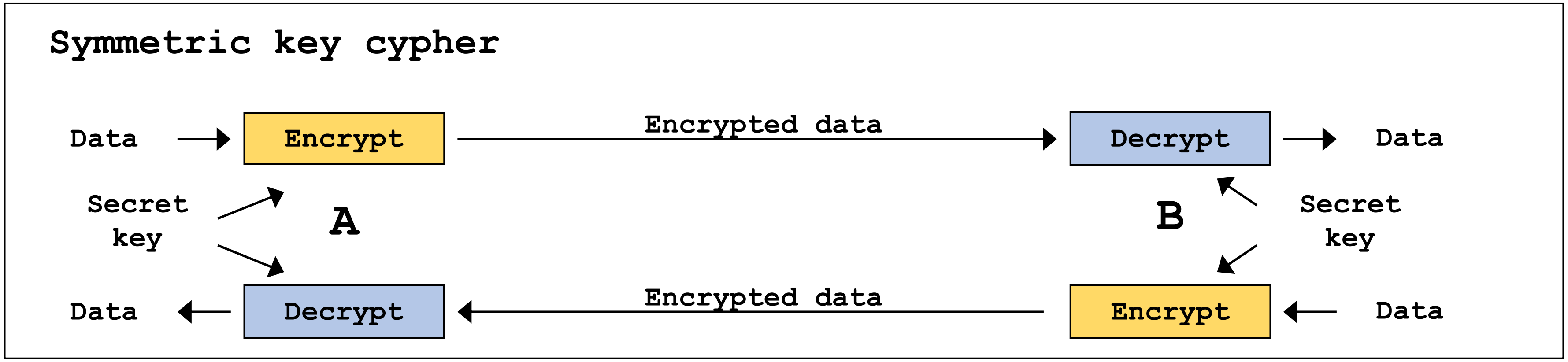
In alternative to the ECB schema, other **block cipher modes of operation** should be used, in simple terms they transform the block cypher into a stream cypher.

The general operation schema is by somehow refeeding the input with the encrypted output of the previous block. Of course, for the first encrypted block there's no encrypted output for refeed, and that's where the **initialization vector** (IV), is used. Some of those block cipher modes of operation are: Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Galois/Counter Mode (GCM). They all encompass an IV to be used on the first block encryption. Of course, to properly decrypt data, the counterpart is required to use the same block cipher mode of operation, and the same IV.

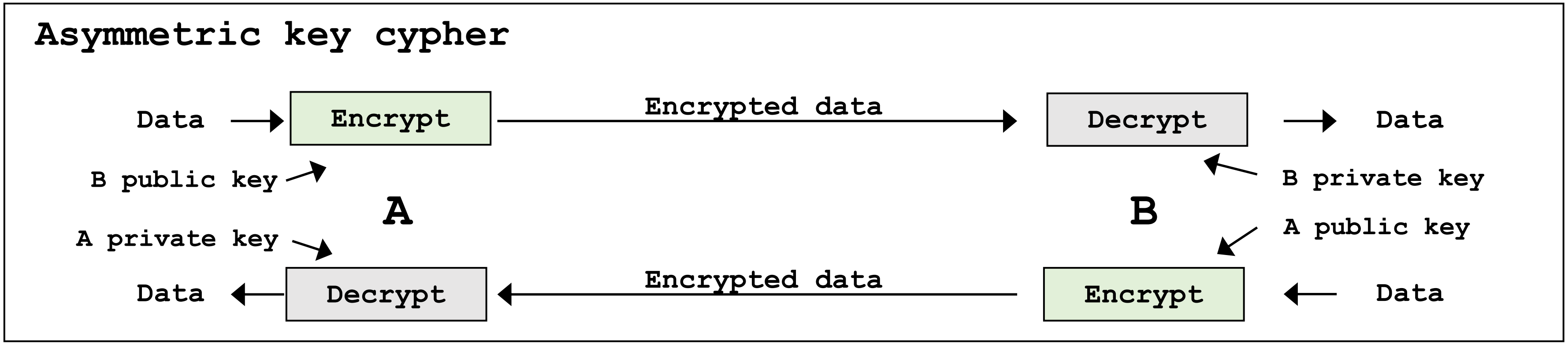
Some cyphers are inherently stream cyphers (e.g., RC4 and ChaCha), they receive a continuous flow of data and produce a continuous flow of encrypted data, yet they should as well receive an IV to start operating.

Symmetric key cyphers and asymmetric key cyphers

With symmetric key cyphers, there's **a single secret key**, it must be secret, and known only by the two parties. The same key is used in both directions.



Asymmetric key cyphers, require **one pair of keys for each direction**. Each party has its own key pair (public + private). Public keys are used for encryption, private keys for decryption. One major advantage is that public keys are not required to be secret, and thus they can be directly sent to the counterpart without security concerns. Only the private key is secret, but unlike what happens with a symmetric key, it's required to be known only by one party.

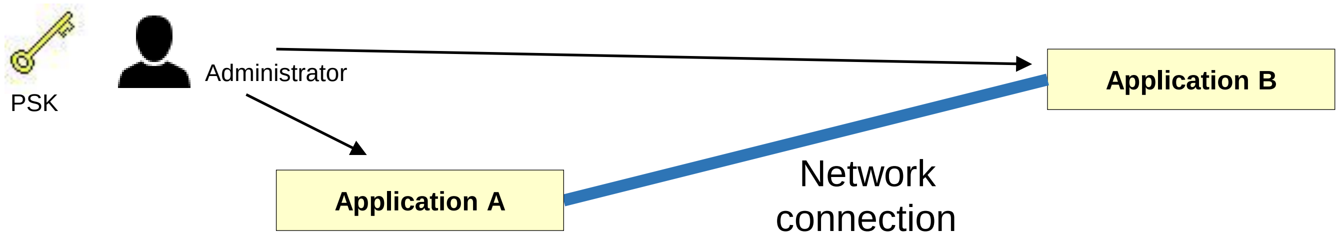


Authentication with symmetric key cyphers

When using a symmetric key cypher, authentication is guaranteed from start. This is because it's assumed that only the two legitimate applications known the secret key, thus if the counterpart is able to dialogue by encrypting and decrypting data using the secret key, that definitely assures authenticity.

Nevertheless, this delegates the authentication to the process by which that secret key was provided to both applications and no others.

If both applications are under control of the same administrator, then the secret key could be manually provided to both sides, this is usually called a Pre-Shared Key (PSK).



One other option is the **co-generation of the secret key**, meaning the two applications will establish through an initial dialogue the same secret key on both sides. This must be achieved in such a way listening attackers won't be able to guess it. The Diffie–Hellman (DH) key exchange mechanism is regarded as an a asymmetric key cipher and allows this.

When a user password based authentication is encompassed, that may as well be used to co-generate the secret key. If there's a user providing a secret password to one client application and the other application is a server with access to that same password, then that's a secret both known and nobody else knows. Thus, it may be used as input of an algorithm to co-generate the same secret key on both sides, usually a random challenge is also included.

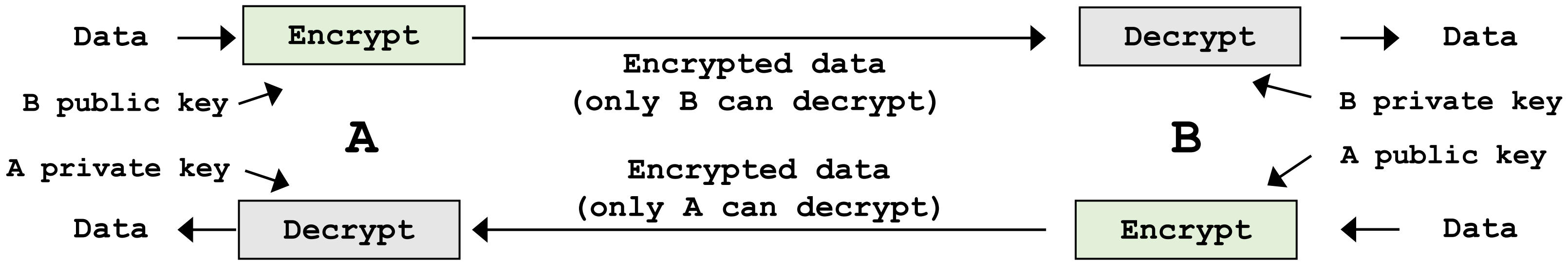
RSA (Rivest–Shamir–Adleman)

RSA is not the only existing asymmetric key cypher, but it was one of the firsts and it's currently widely used. The use of asymmetric key cyphers is called asymmetric cryptography, and also known as public-key cryptography.

The major advantage of public-key cryptography is the key used for encryption is public (not secret), therefore, if an application desires to receive confidential data from other applications, it will simply send them its public key, data encrypted with that key can only be decrypted by using the corresponding private key. The private key must be kept secret, and it's never shared with anyone.

This, definitely solves the issue of safely providing encryption keys through the network. The catch is, unlike with symmetric key ciphers, authentication is not assured. Because the key is public, anyone can use it, so receiving encrypted data tells nothing about the counterpart's identity.

Yet, in network transactions, data is transferred in both directions, between the two applications, so if they can dialogue by encrypting information with the counterpart's public key they know they are talking with the correct counterpart. Yet, for this to be true, each application must be sure it's using the public key belonging to the desired counterpart.



RSA and digital signatures

RSA is based on the remainder of the division evolving very big prime numbers. To be safe, very big prime numbers are used, usually 515-bits, 1024-bits, or 2048-bist long keys are used. This makes RSA very secure, however it's pretty compute-intensive, and thus also rather slow. Considerably slower than symmetric cyphers.

RSA is not used for large amounts of data transfer, instead it's used in strategic phases of dialogues, for authentication, and symmetric keys exchange.

One interesting RSA feature is its use may be reversed, meaning encrypting with the private key and then decrypting with the public key, of course then, anyone can decrypt.

So, what's this useful for? Well, because the private key is known only by its owner, anyone can use the corresponding public key to prove the encrypted data was created by the private key's owner. So this is a digital signature.

A digital signature authenticates the author of a set of data, and also proves who was the author such as he can´t deny it, thus provides non-repudiation.

To produce a digital signature, the RSA algorithm is applied to a hash code resulting from the data to be signed and not the data itself.

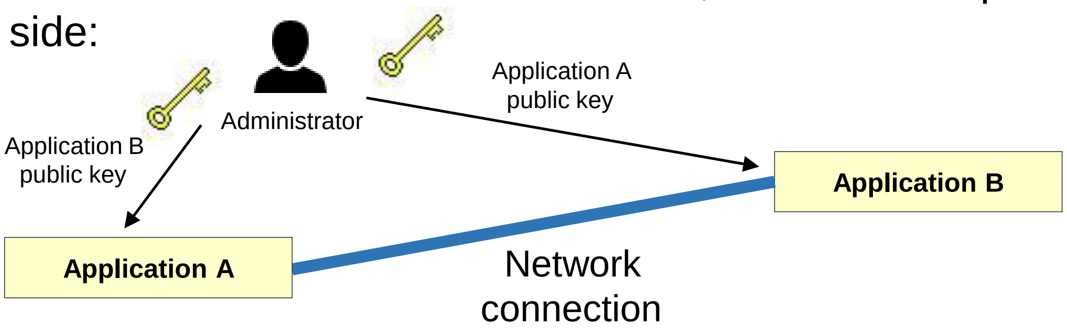
It's all the same, hash functions (e.g. MD5, SHA) guarantee the fixed size hash code they produce depends on the whole input content. Hash functions are also resilient to malicious attacks, namely it's almost impossible to change an input content and yet attain the same hash code.

Authenticating public keys

When using public-key cryptography in a network dialogue between two applications, if both parties are sure they are using the public key belonging to the counterpart they really what to talk to, then authentication is guaranteed.

So the authentication with public-key cryptography is simply a matter of authenticating the public keys. Again, if both applications are under control of the same administrator, the authentic public keys may be manually placed on each side:

However, this it not the case for typical clients and servers that are not aware of each other existence before the first time they interact.



The solution is the use of **public key certificates**. A public key certificate is created by a certification authority (CA), some main elements of a public key certificate are a **public key**, the identification of an **entity**, and a **digital signature**. The meaning is, the CA declares that public key belongs to that entity, and the CA is digitally signing such declaration.

The entity, also called the certificate’s **subject**, is most often a DNS name representing a network node. The public key certificate has some other information, namely the CA identifier (**issuer**) and the dates from when to when it’s valid.

If an application trusts a CA, then it trusts public key certificates signed by the CA. Thus, if an application owns a public key certificate, signed be a CA everybody trusts, then by presenting that certificate it’s able to prove the its identity to anybody.

Certification Authority (CA)

By what means is a CA trustworthy? One by one, they must be provided to applications.

The public key certificate, belonging to each and every CA to be regarded as trustworthy, has to be provided to applications. When an application is checking a public key certificate, the issuer has to be a trusted CAs, otherwise it’s not accepted. Often, this list of trusted certificates is stored in an application folder, a system folder, or in memory (certificate verification storage), they are kept updated while the system itself is updated.

A CA certificate may be issued by another CA, this means a certification chain is built, trust is transitive along the chain, from the issuer CA to the subject CA. Of course a CA may issue certificates for several CAs, so it a tree. It’s also know as PKI (Public Key Infrastructure).

The CA on the base of each tree is suitably known as ROOT CA, because there’s no CA bellow to issue and sign, the ROOT CA certificate is a self signed certificate (issuer=subject).

We would anticipate that, it would be enough to have a ROOT CA certificate in the certificate verification storage, to make trustworthy every certificate on the tree starting there. In principle that’s true, however, applications must follow the chain to validate, thus corresponding certificates (the full chain) must be available on the certificate verification storage.

For network transactions, this problem may be anticipated, instead of sending to the counterpart only the specific public key certificate, the full chain of certificates is sent.

Self signed certificates (issuer=subject) are a safe solution when applications are under control of the same administrator, it’s just a matter of adding the counterpart’s self signed certificate to the local certificate verification storage in each side. Elsewhere these certificates are untrusted.

Entities identification and validation

The purpose of a public key certificate is assuring the included public key is the one that belongs to the identified owner (the subject field). The subject field has to be meaningful, otherwise the whole purpose would be lost.

The content of the subject field is formatted as a X.500 directory object, it’s a set of attribute-value pairs called Relative Distinguished Names (RDNs), at the highest level the country code (C), down to the organizational unit (OU), and the common name (CN).

Public key certificates are to be used by applications. To be useful, there must be a relationship between the subject’s identification, and what the application actually does that encompasses authentication. The attribute taking the most relevant role here is the common name (CN).

- One typical case is for email. To send an encrypted email message or to check a digital signature in a received email message, the correct public key to be used is provided by a public key certificate where the subject’s common name (CN) matches the counterpart´s email address. This is a specific additional validation that must be enforced in this case.
- A wide-ranging, and most relevant, case is network nodes authentication, the goal is avoiding attackers impersonating network nodes, and for instance, carry out a MITM attack. In this case the DNS node name is used, the correct public key to be used is provided by a public key certificate where the subject’s common name (CN) matches the counterpart´s DNS node name. In this scenario, the application is required to perform an additional validation on the certificate: when communicating with some network node, the used public key has to come from a public key certificate where the subject's common name (CN) is that node's DNS name.

SSL/TLS - Secure Sockets Layer/Transport Layer Security

SSL/TLS is a protocol to enforce authentication and privacy at the transport layer (SSL refers to older versions, but they are compatible).

It's designed to be a complement ensuring security on existing application protocols without actually requiring any modification on those protocols. It is enforced at layer four, meaning directly over UDP and TCP. Existing standard application protocols using UDP or TCP are turned secure by, once the network session is created, before sending any application protocol related data, enforcing SSL/TLS. Both authentication and privacy are assured, once security is guaranteed, the standard application protocol may start exchanging its own data, but it’s now encrypted and secured.

By enforcing SSL/TLS, standard protocols like HTTP or SMTP become secure, usually they are represented with a S suffix, thus on the above examples HTTPS and SMTPS. Different default service port numbers are also used, e.g. 80 for HTTP and 443 for HTTPS.

Nevertheless, both secure and unsecure version can be provided on the same service port number, this is called Opportunistic TLS. The service defaults to unsecure, and by default TLS is not enforced, however, if the STARTTLS message is sent by the client, then the TLS handshake is enforced and the session is secured, some application protocols that support this feature are IMAP, POP3, SMTP, LDAP, and HTTP (through the upgrade HTTP header field).

To establish the secure session, TLS negotiates the version to be used and the authentication/privacy/integrity mechanisms, known as cypher suites to be used. The client-server model is implicit, the client sends a **Client Hello** message to the server, this message includes the TLS client's version and a list of cypher suites it supports and it's willing to use, they are specified in preference order.

SSL/TLS negotiation

Once the service receives the Client Hello is responds back with a **Server Hello**, this response includes the TLS version to be used, usually the client's requested version, but it could be a lower version if the requested one is unsupported by the server. In addition the Server Hello response message declares the elected cypher suite among those provided by the client. Both the Client Hello and Server Hello also include random numbers generated by both parties that will avoid some attacks (e.g., replay attacks).

Now things become rather dependant on the elected cypher suite, most often it will encompass public keys authentication (e.g., RSA) but it may also be based on a Pre-Shared Key (PSK).

Let us focus in the most frequent case of public keys authentication. The server will immediately send its public key certificate, certificates along the chain up to it should also be sent. In addition the server may or not request a client's certificate, this is optional.

If the server demands a client's certificate, the client is required to send it on the next message. Both the client and the server (if applicable) validate each other public key certificate.

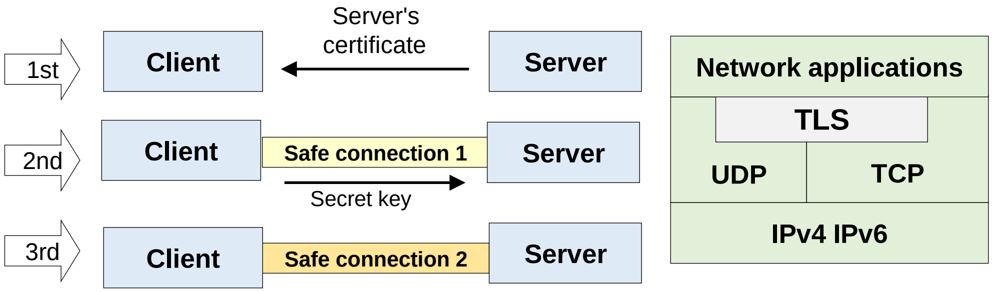
At this stage the server is authenticated (and the client may be authenticated or not), then the authenticated public keys may be used to securely establish a symmetrical key for encryption using a symmetrical key cypher. This depends on the elected cypher suite, and may also encompass other mechanisms, like for instance Diffie–Hellman (DH).

Anyway, from this point on TLS negotiation is done. It has established a symmetrical key cypher to be used and a secret key known only by both parties, from now on data is encrypted and the standard protocol can now proceed with its stuff sending and receiving data encrypted by TLS.

SSL/TLS negotiation

Most often the scenario is a lot of anonymous clients connecting to a public server, then, establishing the client's authenticity is not that important, however, establishing the server's authenticity is vital for clients. Thus most often servers will not demand a public key certificate from the client.

The image on the right presents a simplified view. On the first stage the client receives and validates the server's public key certificate. Then the server's public key may be used so send the symmetrical cypher's key.



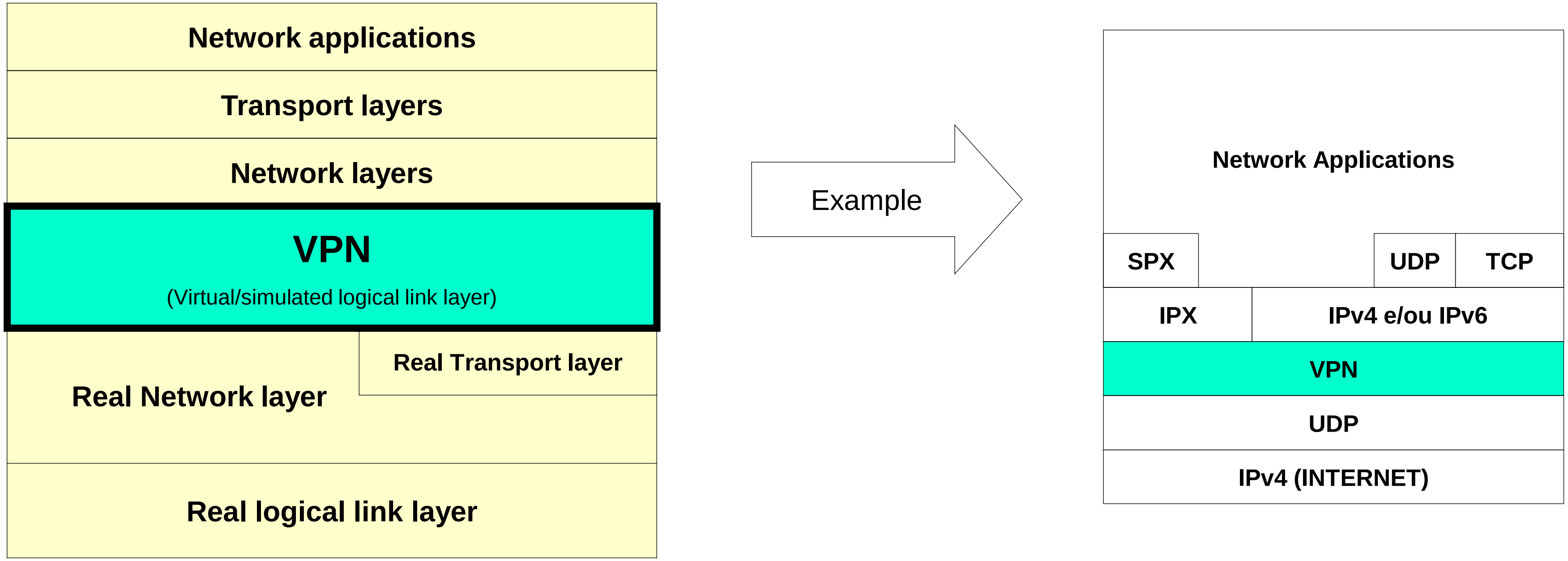
Finally communications are encrypted by using by using the symmetrical key cypher.

The validation of the server's certificate is vital for the client:

- A certificate is valid between two dates it contains (not before and not after).
- By following the certificate's chain, a trusted certificate in the client's local certificate verification storage must be reached.
- The CN attribute of the server's certificate subject field is required to match the DNS name of the server being contacted.
- In addition, some clients may require the checking if the certificate has been revocated. This requires the direct contact of the issuing CA by using a special protocol like OSCP (Online Certificate Status Protocol).

Virtual Private Network (VPN)

A VPN is a **layer two link** simulated over an already existing infrastructure, normally a layer three network infrastructure, most often the internet.



Virtual stand for not being a real physical infrastructure, instead it’s a **point-to-point tunnel** created over an already existing network. Because information sent through the tunnel is encrypted, it’s also named as **private**. Even though every VPN uses the same principles, they can be classified into types by following different criteria.

The network tunnel concept

The main concept around a VPN is the tunnel. What a VPN adds to this concept are security related features, namely authentication and data privacy.

Network tunnels are created by application level tunnelling protocols and are based on encapsulation, this means packets being transported by other packets as payload.

Any pair of network applications, able to communicate with each other, can establish a tunnel. The tunnel is essentially a logical communication channel between two applications, it’s named a network tunnel not so much for what it is, but more due to what is done with it.

Instead of using the network tunnel to transport application’s data, it’s used to transport layer two frames or layer three packets.

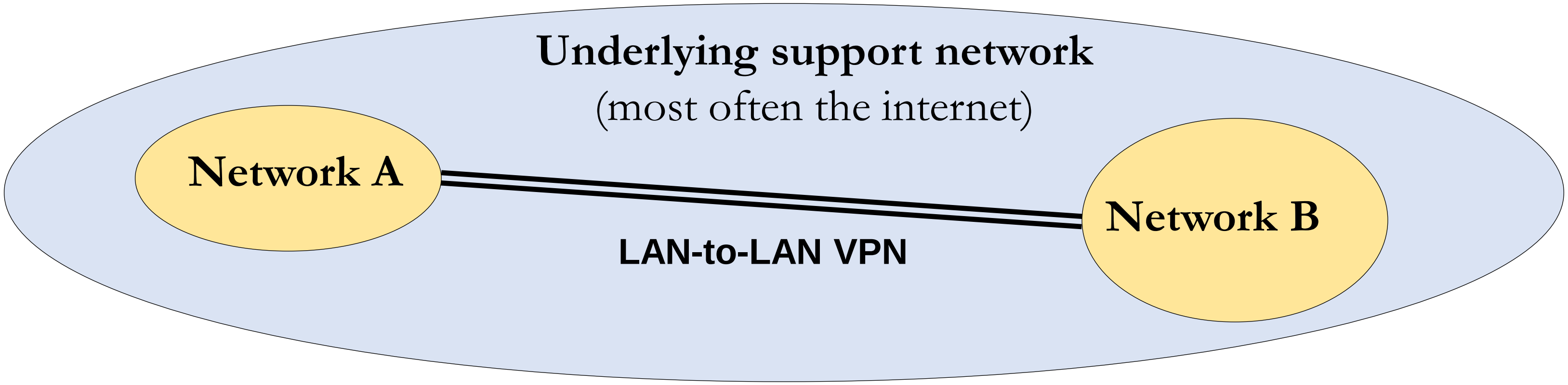
A humble TCP connection may be regarded as a simple and ready to be used network tunnel. Instead of being used for application’s data transfers, it’s used for packets transfer. For instance, Ethernet frames or IP datagrams.

Network tunnels have two endpoints, one application at each end. Both these applications receive packets from local networks and forward them through the tunnel to the other end application. They can be therefore regarded as switches or routers, depending on forwarding layer two frames or layer three packets.

LAN-to-LAN VPN (or Site-to-Site VPN)

One important VPN use is creating a permanent direct link between two remote networks. Of course, this is a task for network administrators, the same way it would be if we were talking about a physical cable. This VPN usage is called LAN-to-LAN.

Network users' traffic passes through the VPN without users being aware it exists, again, as it would be if it was a physical cable installed by the administrator.



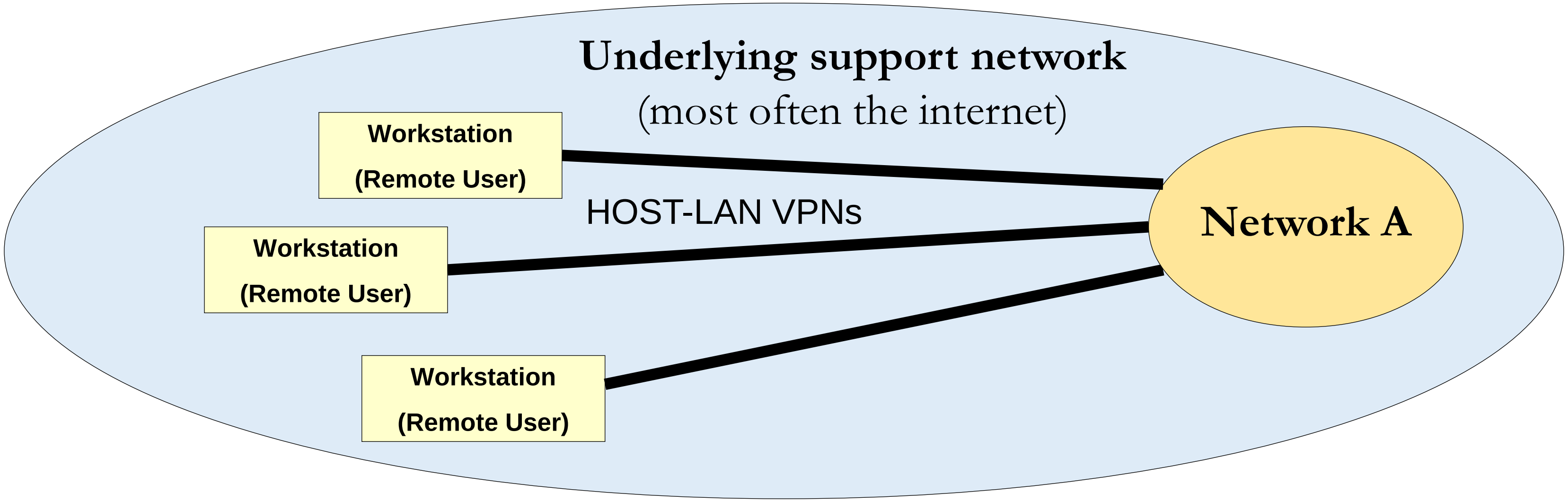
There are several reasons why network administrators may want to do this:

- Networks are private; thus, they can't use the internet to talk to each other.
- Networks are using a layer two or layer three protocol not supported by the internet. For instance, creating an IPv6 tunnel over an IPv4 only network.
- For the sake of security regrading all traffic between those networks.

Host-LAN VPN (or Remote-Access VPN)

Network administrators may offer their users a VPN service. This will allow users to create personal VPN connections to the network whenever they need. VPN connections are created on user demand, this is known as Host-LAN VPN.

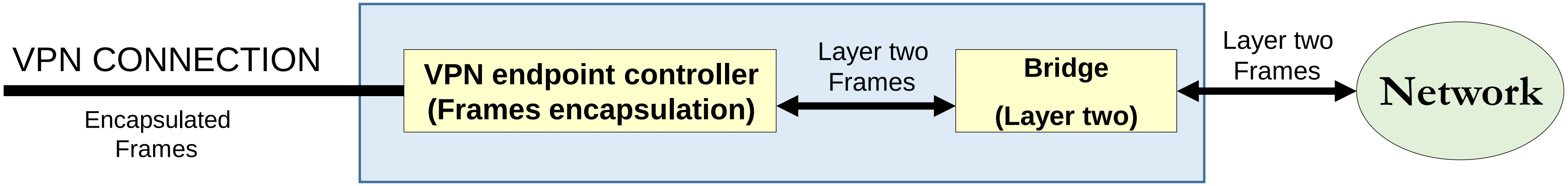
The remote user endpoint workstation runs a VPN client software. On the opposite endpoint there's a VPN server shared by all users.



Users' authentication and access control are enforced by the VPN server. Once the access is granted, the user´s workstation is provided with an IP address belonging to the remote network. The workstation operates as if it was directly connected to that network, this allows the VPN client to bypass firewalls around the remote network and also guarantees privacy on transactions with the network.

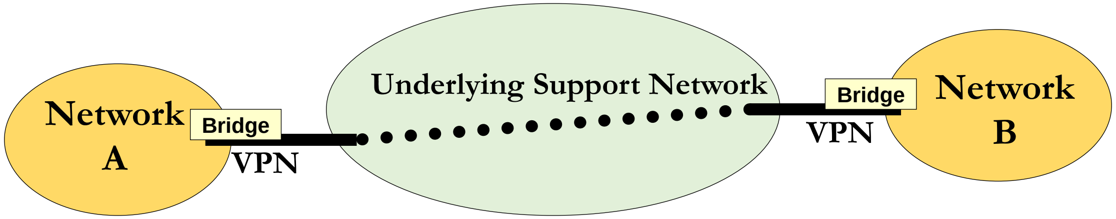
Layer two VPN

A VPN is a point-to-point physical link equivalent. As with a physical link, it can be used for several purposes, one way to connect two networks, or a node to a remote network, is by forwarding **layer two frames**. This means the VPN connection will behave like a **layer two bridge or switch**.



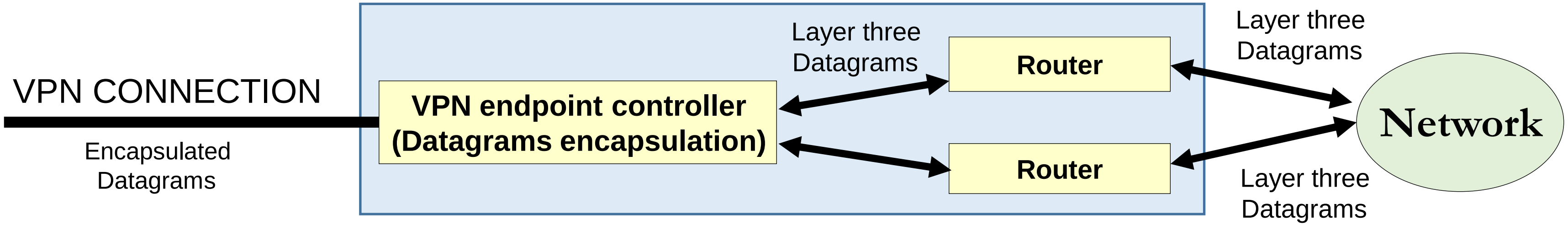
This has some pros and cons. Every layer two frame is forwarded; therefore, the VPN is effective for whatever layer three and upper levels protocols are being used. Broadcast layer two traffic is forwarded as well, thus protocols like ARP and DHCP can operate normally, both sides of the VPN connection become a single network. Nevertheless, sending broadcast traffic through the VPN may represent a significate traffic burden for the VPN itself. Also, both VPN sides must use the same layer two technology, otherwise frame formats would be incompatible.

The image show how Network A and Network B become two segments of the same layer two network, interconnected by a layer two bridge. For instance, under the IP point of view, they are a single network.



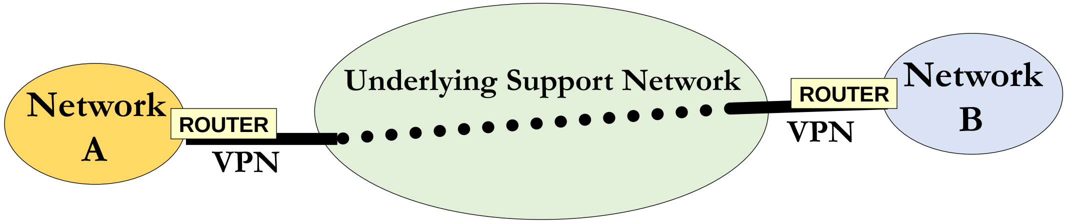
Layer three VPN

The VPN can also be used to forward layer three packets (datagrams), now VPN endpoints operate like routers. If different layer three protocols are to be used, then for each, a router is required (a multiprotocol router).



Because endpoints are routers, broadcast traffic is not forwarded. For a layer three VPN, routing configuration is required because it's not a single network any longer.

The image shown two networks connected by a layer three VPN. In fact, three network addresses are required because there are three different networks.

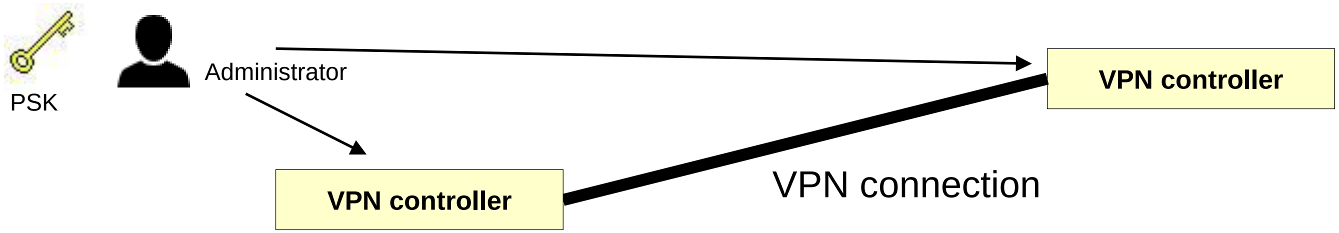


Network A, Network B and VPN connection, they all are different layer three networks, thus, each must have its own valid network address. Layer three routing must also be settled in all routers to guarantee all networks are reachable.
Due to the need of having different layer three networks and routing, generally speaking a layer three VPN is more complex to configure.

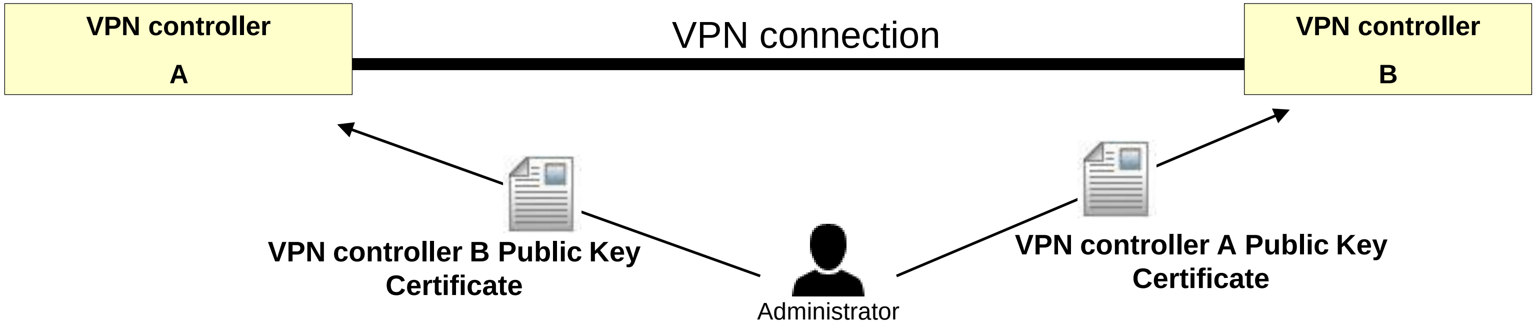
VPN security

In simple terms, a VPN can be seen as a secure network tunnel, security is therefore a key feature for a VPN. Because the underlying support network is presumed to be untrusted it must be regarded as exposed to malicious attacks. We already know security requires authentication prior to privacy.

For a LAN-to-LAN VPN, where both endpoints are under control of the same administrator one possible approach is deploying the same Pre-Shared Key (PSK) on both sides:



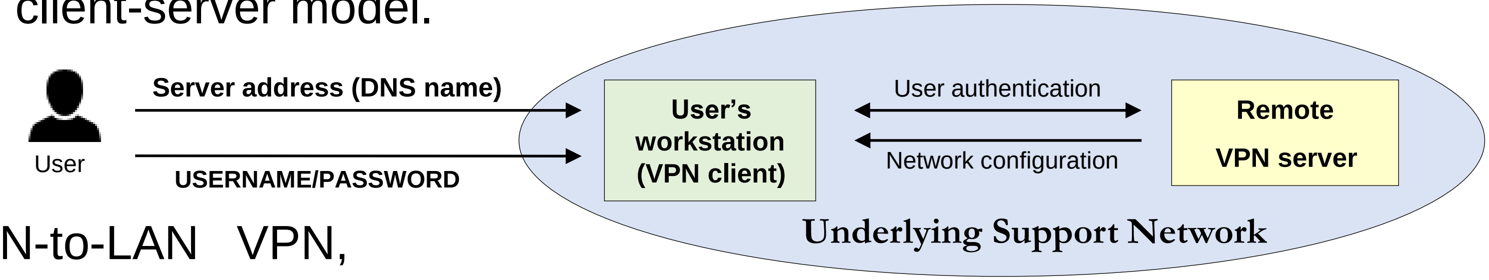
A similar approach is by using public key authentication, then the administrator would add to each side's local certificate verification storage the public key certificate of the counterpart:


VPN controller
A
VPN connection
VPN controller
B
VPN controller B Public Key Certificate
Administrator
VPN controller A Public Key Certificate

Self signed the public key certificates will do very well. Of course, if the used certificates are in a chain already present on local certificate verification storages, they are not required to be added.

Host-LAN VPN and user authentication

A host-LAN VPN has some specific characteristics, most of them are user-oriented and based on the client-server model.


User
Server address (DNS name)
USERNAME/PASSWORD
User's workstation (VPN client)
User authentication
Network configuration
Remote VPN server
Underlying Support Network

Unlike in a LAN-to-LAN VPN, now there are clearly two roles.

One VPN server accepts sessions from several VPN clients. VPN clients are supposed to receive IP configuration information from the server for automatic configuration, either by DHCP or by other equivalent mechanism.

The VPN server demands each client's authentication. Because each client is under a user's control, one simple solution is enforcing user authentication, most often by username/password. Before sending the user password, the client must be sure the server is authentic and transmitted data is being safely encrypted. Regarding the server authenticity, one option is providing to the client a public key certificate. Anyway, data privacy must be coordinated with user authentication.

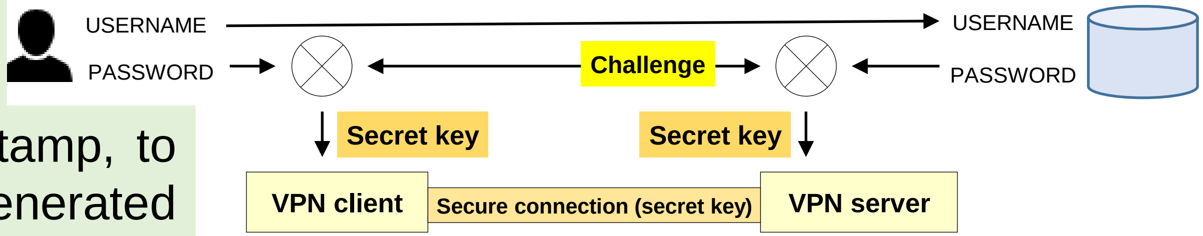
The direct use of a symmetric key cypher with a pre-shared secret key is not realistic because for each user one different key would be required. However, the user's password can be used to generate the same secret key on both sides.

Host-LAN VPN – secret key authentication and privacy

If both the user and the VPN server have access to the user password, it´s a secret nobody else knows, therefore it can be used to co-generate a secret key know only to the VPN client and the VPN server. In this process, a challenge is included.

The challenge is a long server generated, random number.

It should also include a timestamp, to avoid replay attacks, so the generated secret key is unique for each session.



At first glance, this looks magnificent. The server is authenticated, the user is authenticated, and data privacy is assured, also the user password is never sent through the network. As far as the password is known only to the user and the VPN server, both authenticity is guaranteed. This results from being impossible to generate the secret key without knowing the password.

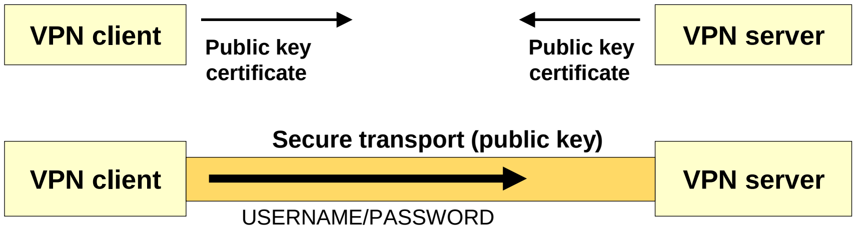
Bear in mind that most user accounts databases store password hashes and not passwords, but that’s no obstacle, the stored hash is used on the server side and the same hash is generated on the client’s side from the user provided password.

This is an all-in-one technique, but there is **one major drawback**. It all depends on the user’s password quality, and that’s very hard to warrant. Notice the challenge is public and passwords are highly susceptible to dictionary attacks, with some effort, the password may be guessed.

Host-LAN VPN – public key authentication and privacy

One better way to protect the user password is by using a strongly secured transport in the first place. This can be achieved with public key encryption and public key certificates.

Public key certificates are exchanged between client and server, this will guarantee both authenticities. Though, the server may not be too keen on the client certificate as it will rely on the user's authentication.



If the server's public key certificate is valid, the client can safely send anything to it by encrypting data with that server's public key. So, the client may then securely send the username and password, and thus, authenticate the user.

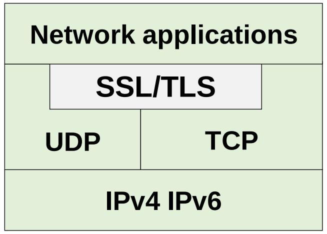
This technique is widely used, though normally the public key secure transport is later switched to a much faster symmetric key cypher.

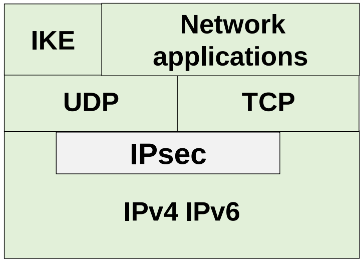
In this case, public keys are used as an intermediate step. Once the public key secure transport is established, the client generates a random secret key and safely sends it to the server. Afterwards, the asymmetric cypher is abandoned, and a symmetric cypher is used instead.

Regarding user authentication, it works the same, but username and password are now encrypted using the symmetric cypher.

Another security layer - IPsec (IP level security)

We have already discussed in some detail the SSL/TLS security layer, on the left image.





IPsec (on the right image) is another security layer, but it runs at a lower layer, over IPv4 and IPV6. The purpose is analogous to TLS, however, as shown in the image, it doesn't interact directly with applications.

Before IPsec starts operating, assuring nodes authentication and traffic privacy, cyphers and keys must be pre-established. Such data is stored in security associations (SA) managed by each node. For each party to where a node wants to send secured data, a SA is required, it's unidirectional, thus, to receive secured data from the same node another SA is required.

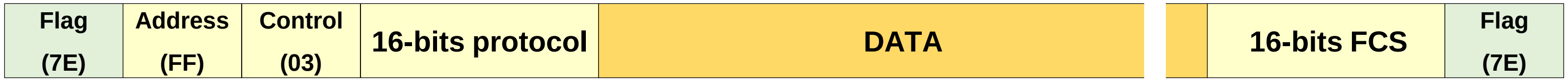
One interesting resulting feature is that communications between two nodes may use different cyphers in each direction.

Other options exist, but IKE (Internet Key Exchange) is the most widely used protocol to create the first SA for IPsec, it operates by using public key certificates and the Diffie–Hellman key exchange.

IPsec is composed by two protocols: **AH (Authentication Header)**, providing authentication and data integrity, but no privacy, and **ESP (Encapsulating Security Payload)**, providing both authentication and privacy. IPsec is an additional protocol to IPv4 but it's included in IPv6 as an extension header.

PPP – Point to Point Protocol

PPP is pretty old, it comes from the primordial HDLC (High-Level Data Link Control). Despite its age, it’s widely used whenever packets are to be transported over a point-to-point layer two link. Thus, many VPN implementations use PPP. The image bellow represents a PPP frame:



Flag (7E)	Address (FF)	Control (03)	16-bits protocol	DATA	16-bits FCS	Flag (7E)

Frames start and end with an 8-bits flag, yet between two consecutive frames, only one flag is required. The sender must ensure the flag doesn’t appear elsewhere on the transmitted frame; a technique called bit stuffing is used to guarantee that.

In a point-to-point connection, addresses are implicit and not really required. The 8-bits address field, inherited from HDLC has fixed value 0xFF, meaning broadcast. The 8-bits control field comes as well from HDLC, the fixed 0x03 value means unnumbered frame.

The 16-bits protocol field is used to identify the payload type, some payload types will be upper-level data like IPv4 or IPv6 datagrams, yet several others are used by PPP itself. One of those protocols is LCP (Link Control Protocol - 0xC021).

LCP defines several commands, including Configure-Request, Terminate-Request, Echo-Request and Echo-Reply. Configure-Request commands are used to negotiate the link configuration (LCP options), they include MRU (Maximum-Receive-Unit), **authentication protocol to be used** and header compression.

The original PPP specification in RFC1661 defines two authentication protocols: PAP (Password Authentication Protocol) with number 0xC023 and CHAP (Challenge Handshake Authentication Protocol) with number 0xC223. By default, PPP doesn’t require authentication.

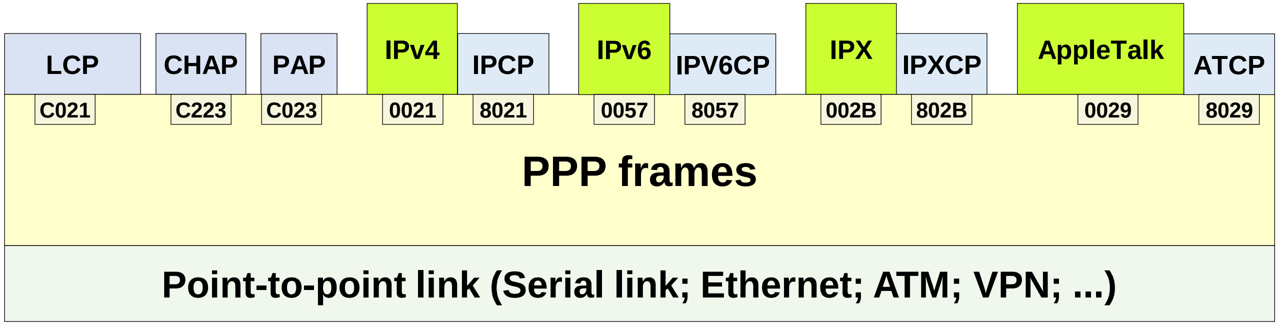
PPP – NCP (Network Control Protocol)

Once LCP negotiation phase is finished, including an optional authentication phase, upper layer protocols can be used, for each there will also be a dedicated NCP (Network Control Protocol).

Network layer protocols are identified by numbers ranging from 0x0000 up to 0x3FFF, for each, there's a corresponding same offset NCP identifier on the 0x8000 to 0xBFFF range. NCP handles the corresponding layer three protocol issues.

For instance, IPv4 has is number 0x0021, the corresponding NCP has number 0x8021 and it's known as IPCP (Internet Protocol Control Protocol). Among other things, IPCP may handle automatic IP node configuration in a DHCP style.

The image below presents a view on global a usage scenario for PPP:



L2TP - Layer 2 Tunneling Protocol

L2TP is the name by which a popular VPN type is known, though the name doesn't tell the whole story, in fact, L2TP is solely a tunnelling protocol.

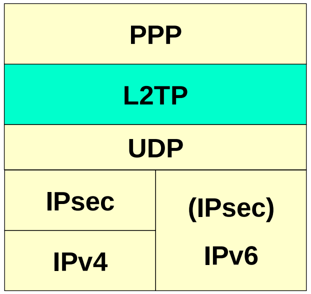
L2TP creates and maintains network tunnels by encapsulating data into UDP packets. Tunnel endpoints are called LAC (L2TP Access Concentrator), and LNS (L2TP Network Server). Usually, the LAC acts as client and establishes a tunnel control connection with the LNS on UDP port 1701. Over the L2TP tunnel, one or several L2TP sessions can be created, each L2TP session is used to transport PPP frames for one PPP session.

L2TP has an optional CHAP authentication procedure during the tunnel creation, but PPP, running over L2TP sessions may itself enforce some security features. But beyond this, L2TP by itself offer no guarantees regarding authentication or privacy. To achieve a viable VPN, L2TP is run over IPsec.

On what is called an L2TP VPN, before L2TP entering in action, IPsec must already be settled, this usually requires IKE. IKE negotiates authentication and encryption by sending UDP datagrams to port number 500.

Two main techniques are available:

- A secret pre-shared-key (PSK), manually placed on both endpoints.
- Public key certificates (required to be valid for both endpoints).


PPP

L2TP

UDP

IPsec

(IPsec)

IPv4

IPv6

Regarding user authentication, in can be enforced within the PPP protocol running over an L2TP tunnel session.

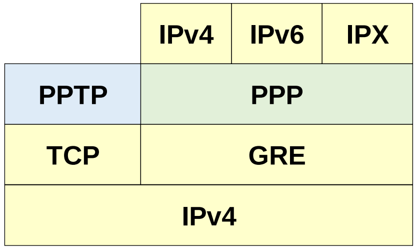
PPTP - Point-to-point tunnelling protocol

PPTP is an L2TP predecessor but is still widely used by Microsoft and also by Cisco. Since it doesn’t use IPsec or TLS, it turns out to be simpler for end users because it doesn´t require neither pre-shared keys nor public key certificates.

PPTP uses a TCP control connection (server port number 1723) to control the data tunnel. The tunnel encapsulates PPP packets into GRE (Generic Routing Encapsulation) packets. GRE was developed by Cisco with the specific purpose of tunnels creation, it runs directly over IPv4 or IPv6 with protocol identifier 47.

One issue with GRE is it hasn’t port numbers, thus NAPT devices will have a difficult time handling GRE packets, many NAT routers may require a specific option, usually called **PPTP pass-through**.

Neither GRE or PPTP have security features, both authentication and privacy must be guaranteed by PPP itself.



IPv4 IPv6 IPX

PPTP PPP

TCP GRE

IPv4

PPP is rather flexible on supported authentication and privacy protocols, currently Microsoft PPTP uses MSCHAPv2 user authentication. During the user authentication a secret key is generated for MPPE (Microsoft Point-to-Point Encryption), this cypher is based on RC4 (Rivest Cipher 4). Along with MPPE, MPPC (Microsoft Point-to-Point Compression) can also be used.

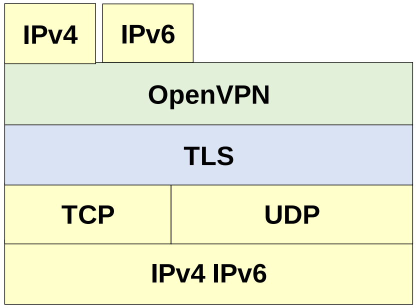
SSL/TLS based VPNs

TLS is a security layer under constant scrutiny and thus regarded as very solid. Several VPN implementations use it.

OpenVPN is open source and rather popular, it creates TLS protected tunnels either over UDP or TCP. Authentication/privacy may use a secret shared key or public key certificates. Both a secret key or public key certificates are suitable for a LAN-to-LAN VPN. On a Host-LAN VPN, however, public key certificates are the best option, the server certificate is especially relevant to the client, also user authentication will be required.

OpenVPN doesn't use PPP, so it must handle user authentication on its own, also client automatic configuration in DHCP style must be performed by OpenVPN on its own.

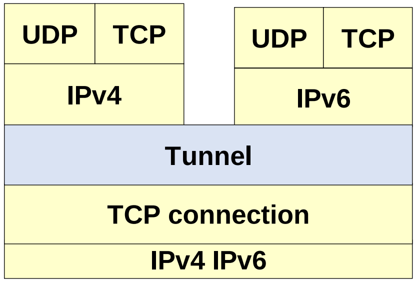
Either running over UDP or TCP, the OpenVPN server listens on port number 1194.



Secure Socket Tunneling Protocol (SSTP) uses a TLS protected TCP connection to port number 443. Therefore, firewalls will regard SSTP traffic as being HTTPS traffic, though internally SSTP has no relation to HTTP. Nodes authentication is established with public key certificates; a **PPP** session is established, and then the user authentication is demanded by the server.

The TCP meltdown issue

TCP based tunnels, like for instance in SSTP and possibly in OpenVPN, suffer from one common drawback called TCP meltdown.



The origin of this issue comes from a reliable TCP connection (the tunnel) being regarded as unreliable by protocols using it, they see it as an unreliable layer two physical link.

Whilst the underlying physical network is not congested and presents no significant error rate, everything runs smoothly, possibly better than over a UDP based tunnel.

Yet, when significant delays and packet loss start occurring on the physical network, problems will come to the surface.

For protocols using the tunnel, data loss is rather normal because they assume they are using an unreliable layer two physical link, so they immediately forget about it and send data again. But the TCP tunnel connection doesn't forget, it will insist on retransmitting every byte until success is achieved, even though the protocol above has long ago forgotten that byte even exists. This will eventually overload and stall the TCP tunnel.

Imagine for instance a bunch of TCP connections running over the tunnel, on failure they will all start retransmitting, and for each retransmission, the tunnel TCP connection will fail to deliver and also starts retransmitting.