Your journey, secured.

TINA VPN Protocol

What TINA VPN protocol adds to a Barracuda CloudGen Firewall and/or Barracuda SecureEdge deployment

WHITE PAPER

NETWORK SECURITY

Introduction

Today's organizational structures rely on Internet connectivity more than ever, be it a client-to-site connection for road warrior or a site-to-site connection from a branch office to headquarters or data centers (on-premise or cloud). All of these cases depend on tamper-proof connections via the Internet. Standard IPsec is most likely what comes to mind when talking about secure connectivity. But standard IPsec can also limit usability in the context of business-critical connectivity setups. To name just a couple of limitations and how Barracuda mitigates:

- **Dead peer detection**: The IPsec RFC specifications for dead peer detection mechanisms leave a lot of room for interpretation, resulting in incompatibilities and suddenly stopping VPN tunnels. Heart-beat detection, deeply integrated into Barracuda TINA, typically identifies dead peers more reliable and faster than other VPN solutions.

- **NAT and proxy traversal**: While IKEv2 made some steps improving the ability for NAT traversal, proxy traversal is still not officially implemented. The Barracuda VPN protocol offers full support for NAT and proxy traversal.

- **VPN and roaming**: With standard VPN client-to-site connections, a change in network and IP address (like when putting the laptop to standby, traveling to a different location, opening up the laptop again) routinely results in connection loss or at least the need to re-authenticate to the VPN server. With Barracuda VPN the client instantaneously and imperceptible to the user reconnects to the VPN server without session loss or the need to log on again.

- **Ability to control traffic inside the VPN tunnel**: The standard IPsec protocol, even when using IKEv2, provides very limited capabilities to control the traffic and the sessions inside the VPN tunnel. This has been built into the products CloudGen Firewall and SecureEdge from initial conception. QoS and even performance-based application control is fully available in the VPN tunnel. In addition, multiple physical uplinks can be used to share the load of a single logical VPN tunnel. For load optimization the admin can reassign sessions between the physical uplinks manually or use the built-in SD-WAN features to achieve performance-based load balancing with session-based optimization applied automatically.

- **Tunnel monitoring**: Related to the above, the standard IPsec protocol does not provide much help when maintaining or troubleshooting a VPN tunnel. Barracuda VPN includes tunnel heartbeat monitoring to control a VPN tunnel and –if needed– failover to a different line much quicker and smoother than standard IPsec implementations that need to wait for a timeout, then fail-over, and potentially update routing information.

- **Built-in packet loss mitigation**: Encrypted connections with the TINA protocol are designed for high-speed networking across shared lossy lines such as internet broadband or 4G/5G. The underlying forward error correcting (FEC) technology to remediate packet loss is based on a new set of algorithms in the category of random linear network codes (RLNC). Algorithms based on RLNC codes react much faster to losses, remediate these faster on the fly, requiring fewer packet retransmissions and reducing overhead on the devices. This results in high quality voice and video calls even in high packet loss scenarios and with many subscribers on the shared line.

- **Self-healing connectivity**: To achieve the best possible user experience across the WAN and to the cloud service, SecureEdge site devices proactively measure the available bandwidths and quality of all internet uplinks and between VPN endpoints. The results are directly available to the security and SD-WAN policy engine to select the best suitable uplink per application or to disqualify an uplink if the bandwidth or latency fall outside of acceptable limits. Adaptive Session Balancing technology ensures using the best available uplink for the application profile, for all encrypted tunnels across SD-WAN sites. If the health state of the initial uplink recovers, encrypted SD-WAN traffic transparently switches back to this previously defunct uplink.

- **Cloud and provider friendliness**: Standard IPsec is and will always be detected as what it is: IPsec. There is no hiding from it. Some cloud providers and internet providers limit the bandwidth available to IPsec traffic or require more expensive "Business" tariffs. Barracuda's TINA VPN is visible as a standard network connection, not falling under these limitations.

Contents

In order to improve the reliability and performance of such remote connectivity, Barracuda has created its own transport-independent network architecture - **TINA** for short.
The TINA protocol can encapsulate encrypted ESP payload in TCP or UDP packets, thus adapting to underlying transport network quality and providing failure-resistant, high-speed VPN connections.
TINA also substantially improves VPN connectivity by adding:

- Multiple concurrent physical transport paths per logical tunnel
- Session-level or packet-level transport aggregation for increased total tunnel throughput
- Adaptive traffic shaping depending on VPN transport availability
- Packet loss mitigation by applying Forward Error Correction techniques
- Fallback transports in case of uplink failure
- Traffic compression
- DHCP and NAT support
- Heartbeat monitoring for failover scenarios

The TINA protocol is used in the following settings:

Client-to-SASE service

- Dedicated SASE agents for Windows, macOS, iOS, Android, and Linux

Client-to-site connectivity

- Dedicated VPN clients for Windows, macOS, and Linux
- Clientless, browser-based SSL VPN (transparent agent)
- CudaLaunch app for Windows, macOS, iOS, and Android

Site-to-site connectivity between

- on-premise/virtual CloudGen Firewall or SecureEdge
- on-premise/virtual CloudGen Firewall and public cloud offerings like Amazon Web Services, Microsoft Azure, or Google Cloud Platform
- on-premise/virtual SecureEdge and public cloud offerings like Amazon Web Services, Microsoft Azure, or Google Cloud Platform
- public cloud offerings like Amazon Web Services, Microsoft Azure, or Google Cloud Platform

Phases of an encrypted session

During an encrypted session a client/partner server passes through the following phases:

Phase 1	Pre-handshake and handshake phase (authentication phase)
Phase 2	Configuration phase
Phase 3	Traffic phase
Phase 4	Rekey phase
Phase 5	Termination phase

Phases 1 and 2 cover the authentication process and ensure that only clients with appropriate credentials may proceed to the next phases.

Possible credentials

- Possession of a private RSA key that matches a known public key. ("Known" means either explicitly known or through a X509 certificate)
- A valid login password combination.
- [optional] Additional two-factor authentication

Depending on the configuration, one or both type of credentials must be present.

Verification of credentials

RSA key - Depending on the capabilities of the RSA key (keys on smartcards or tokens may have a restricted key usage) a valid digital signature or a private decryption of dynamically (!) generated data (avoid replay) is required. In case the public key is known through an X509 certificate, the certificate is also verified.

X509 certificate - The root certificate of the issuer has to be known. The certificate must be signed by that root certificate. The certificate may not be expired. The certificate may not be on the CRL belonging to its root certificate, or optionally, an OCSP call verifies the validity of the certificate.

Login/password - Login name and password are verified at an external authentication server (LDAP, NTLM, ADS, ACE, Radius)

Note:
No matter how many intermediate steps the authentication involves (e.g., DoS save methods, cookies, or DH intermediate keys), authentication always reduces to the two credentials mentioned above.
This also applies to the IKE protocol.

Phase 1 - Pre-handshake and handshake phase (authentication phase)

We distinguish three scenarios for the handshake phases:

Case A. The partner public key for a specific tunnel is known to the system and, optionally, a password is required.

Case B. The public key is not known but will be received via an X509 certificate.
Variant A: Signing packet with client private key
Variant B: No signing packet, public / private encrypt

Case C. No public key/ X509 certificate is required. Valid login password combination is sufficient.

For case A, a client may proceed to the handshake phase immediately. For case B and C, the client has to pass the pre-handshake phase first, which will exchange the partner's public keys.

General comment:
In the following, the term *cookie* will appear. A cookie is simply a sequence of random numbers that may be used to assure that a reply was answered by the correct partner. Unencrypted cookies help to avoid communication with attackers using IP spoofed addresses. Encrypted cookies play the role of the dynamic part of encrypted data so "Replay Attacks" cannot be performed (i.e., if network traffic is intercepted and replayed at a later time).
The term **client** refers to the initiator of the handshake. This can be a VPN client used for personal or group VPN, the SecureEdge Access Agent (all commonly referred to as client) or another VPN server (active partner) in the case of site-to-site VPN.

The pre-handshake phase

A client that wants to pass the pre-handshake phase must first obtain a so-called *cookie* that is required for further communication. Since in the pre-handshake phase no keys are known by the communication partners, pre-handshake cookies (*PHS cookie*) do NOT involve encryption and are built using a MD5 hash of the IP address of the requestor and a secret 16-byte sequence, which is changed randomly every 10 seconds.

- **Client PHS step 1**:
  Generate a 16-byte random sequence: *client cookie*.
  Send a *PHS cookie Request* to the server including the *client cookie*.

- **Server PHS step 1**:
  Reply a *PHS cookie* for the client IP, the *client cookie* and the server X509 certificate.

- **Client PHS step 2, case B, variant A**:
  Verify the *client cookie*.
  Verify the server X509 certificate.
  Create a 16-byte random string.
  Send a *PHS request* including the client X509 certificate, the replied *PHS cookie*, the random string, and the client public key used for further encryption.
  Packet is signed using the client private key.

- **Server PHS step 2, case B, variant A**:
  Verify the *PHS cookie*.
  Verify the X509 certificate and the packet signature.
  Reply the random string, a unique name for the tunnel, and the server public key used for further encryption.
  16-byte random string and server public key are hashed.

- **Client PHS step 2, case B, variant B**:
  Verify the *client cookie*.
  Verify the server X509 certificate.
  Create a 16-byte random string.
  Send a *PHS request* including the client X509 certificate, the replied *PHS cookie*, the random string, and the client public key used for further encryption.

- **Server PHS step 2, case B, variant B**:
  Verify the *PHS cookie*.
  Verify the X509 certificate.
  Create *secret cookie* and hash with *PHS cookie*.
  Send a *PHS NO SIG RESPONSE* including random string and secret hash.

- **Client PHS step 2.1, case B, variant B**:
  Verify the random string.
  Send a *PHS NO SIG REQUEST* including secret hash, client X509 certificate, replied *PHS cookie*, random string, and client public key used for further encryption.

- **Server PHS step 2.1, case B, variant B**:
  Verify the *PHS cookie*.
  Verify the X509 certificate.
  Verify the secret hash.
  Reply a *PHS Reply* including a random string, a unique name for the tunnel, and the server public key used for further encryption.
  16-byte random string and server public key are hashed.

- **Client PHS step 2, case C**:
  Verify the *client cookie*.
  Verify the server X509 certificate.
  Create a 16-byte random string.
  Encrypt the login and password using the server public key (from X509 certificate).
  Send the encrypted login/password, the replied cookie, the random string, and the client public key used for further encryption.

- **Server PHS step 2, case C**:
  Verify the *PHS cookie*.
  Decrypt the login and password.
  Verify the login/password combination.
  Reply the random string, a unique name for the tunnel, and the server public key used for further encryption.
  16-byte random string and server public key are hashed.

In this way, the server as well as client have the possibility to validate the authenticity of their partners. Both PHS cookie and client cookie are important to avoid attackers using IP spoofing from sending requests or replies at a fast rate in order to keep the receiver busy decrypting.

After a successful pre-handshake, each partner possesses a public key of the communication partner that can be used for further decryption and encryption and a unique name of the tunnel that will be used to reference it for the further communication in the handshake phase. Note that the purpose of the pre-handshake phase is solely the exchange of the public keys. For establishing a tunnel, a successful handshake phase is required as a next step. The public keys exchanged do not have to be identical with the public key contained in the X509 certificate. While X509 public key is used for authentication purposes, the passed public key is used for further decryption and encryption. This is due the private part of the public key in the X509 might reside on smartcards or security tokens. Decryption and encryption on tokens is not very fast, so Barracuda uses that key for authentication only and exchange the public part of a dynamically generated 2048-bit RSA key that will be used for all further operations.

Cases B and C may occur in a combined request, where both an X509 certificate AND a login/password combination is required.

The handshake phase

Prior to the start of the handshake phase, the client as well as the server possesses the public key of their communication partner and has knowledge of the unique name for the tunnel. The server holds a so-called *server prebuild cookie* for each known (unique name) partner. The server prebuild cookie is a random sequence of 16 bytes that is publicly encrypted with the partners public key. This cookie is available before the communication starts, is valid for one single request, and is only replaced when used in a properly validated request. In this way, the server will only have to perform expensive encryption operations for the next request which implies a successful prior request.

Prior to communication

- **Client HS, step 0**:
  Build *client cookie I* and *II* (encrypted, using server public)
- **Server HS, step 0**:
  Build *server prebuild cookie* packet (encrypted, using client public)

Communication starts

- **Client HS, step 1**:
  Send a *cookie request*.
- **Server HS, step 1**:
  Reply to the *server prebuild cookie* packet (encrypted).
- **Client HS, step 2**:
  Decrypt the received *server prebuild cookie*.
  Send a *verify request* including the decrypted *server prebuild cookie* and the encrypted *client cookie I*.
  Packet is signed using the client's private key.
- **Server HS, step 2**:
  Validate the received *server prebuild cookie* (cleartext).
  Validate the packet signature.
  Decrypt the *client cookie I*.
  Generate a *server request cookie* (encrypted).
  Reply to the decrypted *client cookie I* and the encrypted *server request cookie*.
  Packet is signed using the server private key.
- **Client HS, step 3**:
  Validate the *client cookie I* (cleartext).
  Validate the packet signature.
  Decrypt the received *server request cookie*.
  Send a *handshake request* including the decrypted *server request cookie* and the encrypted *client cookie II* and handshake PAYLOAD (see below).
  Packet is signed using the client private key.
- **Server HS, step 3**:
  Validate the *server request cookie* (cleartext).
  Validate the packet signature.
  Decrypt the received *client cookie II*.
  Set the tunnel as ACTIVE.
  Reply to the handshake request including the decrypted *client cookie II* and handshake PAYLOAD (see below).
  Packet is signed using the server private key.
- **Client HS, step 4**:
  Validate the *client cookie II* (cleartext).
  Validate the packet signature.
  Set the tunnel as ACTIVE.

Purpose of the **PHS cookie**:

The server will only talk to clients that have successfully decrypted the *PHS cookie* which makes it virtually immune to DoS attacks.

Purpose of the **verify request** as an intermediate step:
In the handshake process, the operations for client and server are not symmetric. Since the client is initiating the process, it has the disadvantage - compared to the server - that it has to perform a decryption of a reply packet in order to proceed. Furthermore, the client could be attacked by a man-in-the-middle that is replaying old *cookie replies*. The purpose of *verify request* is to filter out the replies coming from the "real" server and bringing the client and server into a state where they have mutually exchanged cookies that can then be used to perform the actual transaction. It also solves the problem that packets might get lost and retransmissions occur, which have to be distinguished from replayed packets. Of course, the client can be subject to a DoS attack involving the need to decrypt data, but this is only for the limited period of time it takes to get from step 2 to step 3.

This sequence for performing a request/reply operation is not limited to the handshake phase but is also used for various other request types (see below). In this case, steps 3 to 4 are replaced with other specific request and reply packet payloads.

By default, CloudGen Firewall and SecureEdge support Perfect Forward Secrecy (PFS) and Elliptic Curve Cryptography (ECC). The VPN service sends and responds to PFS/ECC requests and uses ECC if it is also supported by the remote service.

At the end of a successful handshake phase, the client and the server have exchanged the following information (passed as encrypted PAYLOAD of handshake request and reply):

Client-to-server
- Hashing algorithm used
- Hashing key
- Encryption algorithm used
- Encryption key
- SPI number (security parameter index)
- Tunnel mode
- Client OS information
- Client version information
- Rekey time
- Alive heartbeat rate
- Alive heartbeat timeout
- Password (only case A with required password)
- Preferred client IP

Server to Client
- Result of the handshake
- Human readable result string
- Hashing algorithm used
- Hashing key
- Encryption algorithm used
- Encryption keys (ECDH-derived)
- SPI (security parameter index)
- Assigned client IP (client only)
- Assigned client default gateway (client only)
- Assigned network routes (client only)
- Assigned domain name suffix (client only)
- Assigned name server (client only)
- Assigned WINS server (client only)
- Server time

Client and server now have exchanged a so-called **transport key** which is a symmetric key that will be used to encrypt network traffic.

The type of key used (cipher) is negotiated and can be either DES, 3DES, BLOWFISH, CAST, AES128 , AES256, or a NULL cipher. The server decides (configuration) if a proposed cipher is acceptable. The same is valid for the hashing algorithm that can be either MD5, SHA, RIPEMD160, SHA256, SHA512, GCM, or NOHASH.

Note:
The product's recommended authentication algorithm, NOHASH, is sometimes referred to as 'TINA standard hash' in the product. However, this method does not use a traditional hashing mechanism and derives its name from this difference. Instead, it is based on a similar concept as used in GCM for creating a hash, i.e., by using metadata and payload information to derive an authentication tag. NOHASH complements a regular block cipher, providing encryption for data confidentiality and authenticity within a single, efficient algorithm. To guarantee data packet integrity, NOHASH authentication employs the following method:
- In addition to the payload, encrypted packets include metadata such as the nohash-result and a sequence number.
- After decryption,
  - the included nohash-result is verified for authenticity
  - the included sequence number is verified as a replay protection and as anti-tampering check.

Upon successful handshake, the server activates the tunnel by setting up an SA (security association, see IPsec).

In case the client is a personal or group VPN client, the client uses the received IP address, gateway, and network routes to setup his network configuration.

Phase 2 - configuration phase

In addition to the information passed along with the handshake reply, the client needs more information for the server.

This involves:
- An online firewall rule set
- An offline firewall rule set
- A message of the day for user information
- A corporate logo bitmap
- A registry check set

Due to the size of this information, this data is passed with separate requests after the handshake request. The patterns of these requests are exactly the same as for a handshake request except that the request type for steps 3 to 4 differs.

Phase 3 - traffic phase

The client and server exchange network traffic that is encrypted using the communicated transport key. The payload is equivalent to an IPsec ESP (encapsulation security payload) packet but contains a 4-byte Barracuda header for packet type identification (packet type=17). It features:

- SA lookup by SPI number and peer
- Authentication header
- Encrypted payload with padding check
- Sliding packet bitmap for replay protection

From time to time (alive heartbeat time) client and server exchange so-called **heartbeats**, which are used to probe the availability of the tunnel. So-called **alive requests** are not like the handshake requests performed with the asymmetric RSA keys but rather with inexpensive transport keys (transport encrypted request/reply).

Phase 4 - rekey phase

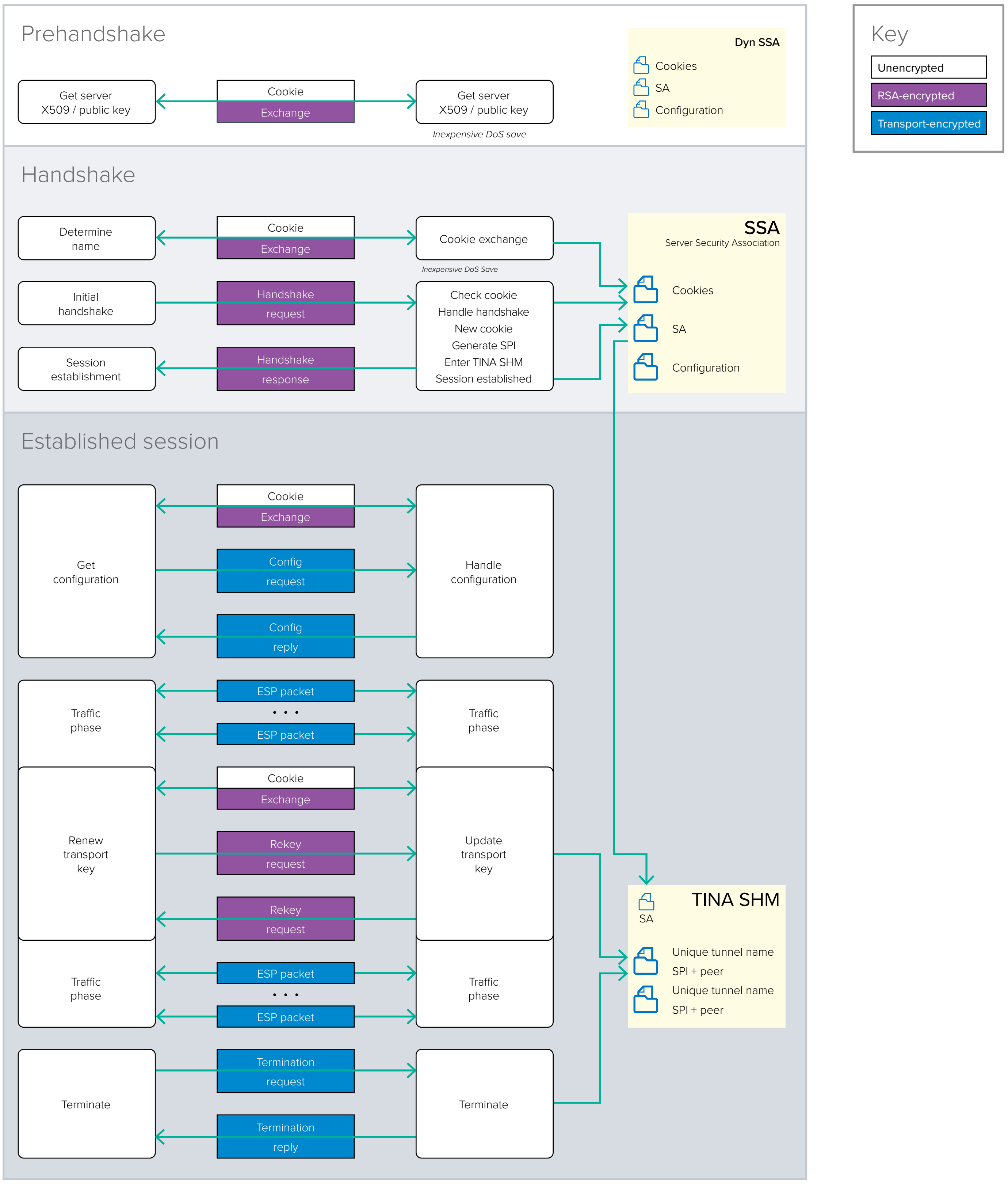
In order to update the transport keys, the client and server have agreed on a rekey time (the shorter time wins) after which a transport key is supposed to be replaced. Replacement means that both keys, server and client, are refreshed. The rekey phase is again handled like a handshake request (cookie > verify > rekey request) with the new transport key as payload.

Since the initial credentials (large asymmetric keys) are required for that operation, the operation is said to have
“perfect forward secrecy”.

Phase 5 - termination phase

Termination can be performed by either client or server and is a simple transport-encrypted request/reply. Termination may also occur without notification in case that alive heartbeat packets are not replied over a period of time (alive heartbeat timeout) and the tunnel is assumed to be non-functional.

Appendix - Timeline of a session



Your journey, secured.