SECURE PACKET SHIELD (SPS)

Core Technology

Eva Bozoki and **Aharon Friedman**

Fortress Technologies Inc.
Tampa, FL.

Presented at the Annual CSG Conference, Apr 28, 1998, St Petersburg, FL.

SPS

SPS [1] is Fortress Technologies' patented advanced system of smart encryption, authentication and data integrity, which allows the implementation of fast, simple and secure VPNs. It fully automates all critical cryptographic functions and implements all security functions without human intervention. It uses the Diffie-Hellman key exchange protocol to negotiate a common secret key used by the symmetric IDEA encryption algorithm [2] and it applies the LZRW1 algorithm [3] for compression of the payload in the IP packet. It uses two sets of private and public keys to compute two secret common keys between any two communication partners; one static — which is used to encrypt the public key exchanges — and one dynamic, which is used to encrypt communication and which is changed automatically every 24 hrs. SPS provides strong authentication, checking the data-integrity-in-transit as well as the integrity of the crypto keys, prevents IP spoofing and hides the communicating parties MAC addresses while limiting communication to within a Company unit by sharing the same Unique Company Signature.

Encryption in SPS

The strength of encryption security rests in:

a) the cryptographic strength of the algorithm (how resistant it is against cryptoanalytical methods of cracking); and

b) the length of the encryption key.

The longer the key, the tougher it is to crack it with brute force method. For these reasons, Fortress Technologies prefers to use the 128-bit International Data Encryption Algorithm (IDEA) in its VPN products. IDEA was designed to be computationally efficient. It is a symmetric algorithm, which operates on 64-bit blocks of plaintext, encrypting 64-bit blocks of plaintext into 64-bit blocks of ciphertext. IDEA was developed in the mid 1970s. It has been widely scrutinized since then, with no known instances of having been broken. Each 64-bit superblock is divided into four16-bit blocks. IDEA operates in 17 rounds, where the odd and even rounds are different. The 128-bit key is expanded into 52 16-bit keys, from which four are used in the odd and two are used in the even rounds. The IDEA scheme is illustrated in Figure 1. IDEA uses three reversible operations, the bitwise exclusive or (⊕), a slightly modified add (+) and a slightly modified multiply (⊗). The reversibility of operations is important to run IDEA backwards, e.g., to decrypt.

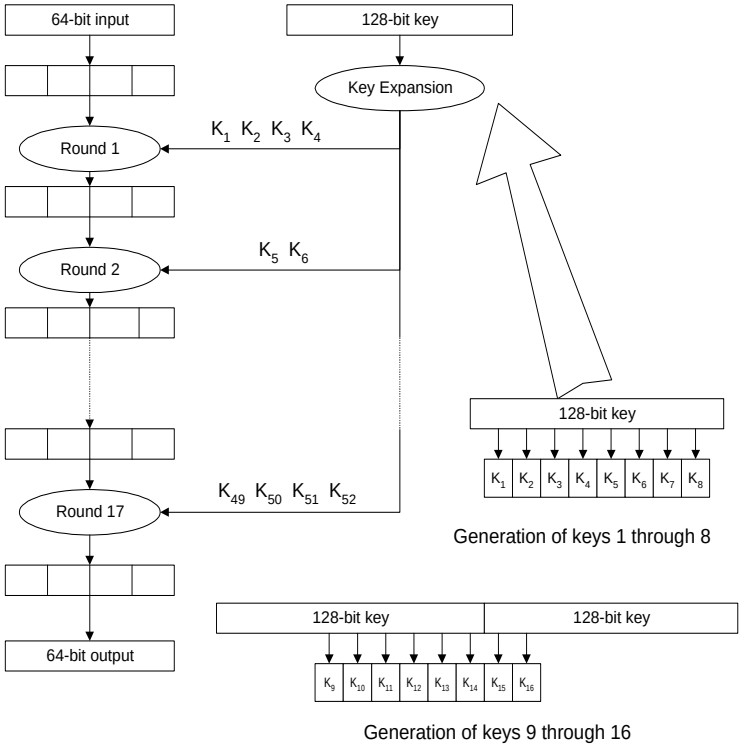
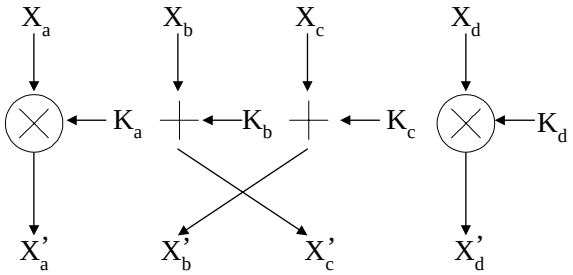
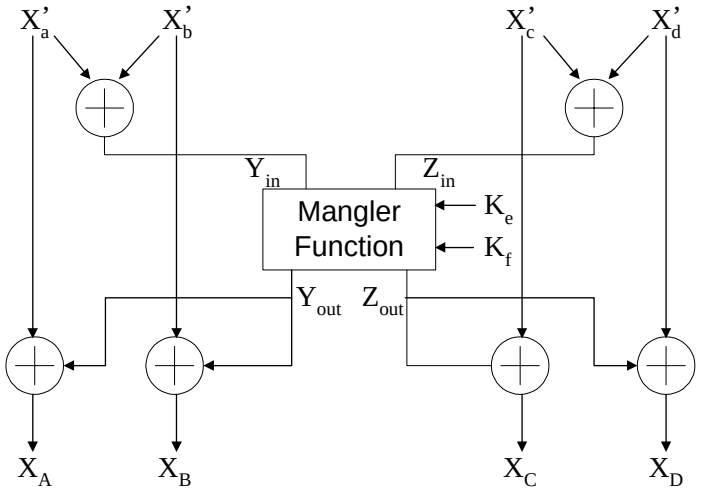


Figure 1. IDEA Scheme of Encryption

Figure 2 shows the arithmetic of the odd and even rounds. X_a, X_b, X_c, and X_d are the four 16-bit input blocks of each round and X_A, X_B, X_C and X_D are the four 16-bit output blocks (odd and even). Y_{in}, Z_{in}, Y_{out} and Z_{out} are the input and output of the mangler function used in the even rounds.



IDEA Odd Round



$$Y_{out} = ((K_e \otimes Y_{in}) + Z_{in}) \otimes K_f$$

$$Z_{out} = (K_e \otimes Y_{in}) + Y_{out}$$

IDEA Even Round

Figure 2

Establishing a common crypto key

The Diffie-Hellman key exchange protocol provides secure distribution of a symmetric crypto key between two nodes of a network. In this protocol both nodes compute their common crypto key from their own private and the other node's public key. The nodes exchange their public keys, but the crypto key never leaves the nodes.

In this scheme each node has two keys; a private key, x_i, known only to that node, and a public key which is calculated as

$$X_i \equiv q^{xi} \pmod n \qquad \text{(Eq. 1)}$$

where n is a large prime number, and $q < n$ is a primitive root of n [4]. These numbers are known to both parties. The i-th and j-th node can calculate their common secret key from their private key and the other node's public key:

$$K_{ij} \equiv X_i^{xj} \pmod n \equiv X_j^{xi} \pmod n \qquad \text{(Eq. 2)}$$

By exchanging only their public keys, the i-th and the j-th nodes are able to establish their shared secret key.

This method, however, is vulnerable against a man-in-the-middle attack. The instants of vulnerability can be significantly reduced by introducing two private and two public keys in each node; static, which never changes and dynamic which is changed periodically. One can use the static common crypto key, developed via a Diffie-Hellman key exchange, to encrypt every consecutive dynamic key exchange. In this scheme there is only one vulnerable key exchange between every pair of nodes. Only this first instance of key exchange do we have to trust as secure, all the consecutive key exchanges will be encrypted and thus known to be secure.

The possibility of a man-in-the-middle attack can be completely eliminated using an out of bound key exchange, or if the static key exchange is also encrypted say, with a “hard” key.

The static keys are unique for each encryption device; they are properties of the unit itself. The static private key is calculated by a random number generator from a seed which is characteristic to that unit only, and the corresponding public key is calculated from it according to Eq.1. Even if the unit is turned off and back on again, the generated static keys will be always the same, and they never change.

On the other hand, the dynamic keys are regenerated every 24 hours and every time they are different. The dynamic private key is randomly generated from a seed related to the time when it is generated.

At the start of communication between two nodes, first they exchange their public static keys and then calculate their static common secret key according to the Diffie-Hellman protocol. After the static keys are exchanged, the Diffie-Hellman protocol is performed again, this time with the dynamic keys. However, this communication is encrypted with the common secret static key.

The relevant information for a communication partner (indexed by its IP), like the static and dynamic common keys, are kept in two tables, one is permanent the other is volatile. The former contains information for secured communication partners only[1], and they are stored in non-volatile memory. The size of this table is unlimited. The latter contains information for secured and unsecured sites; this table is stored in volatile memory. The size of this table is limited to 1024 entries with the oldest entry overwritten by the 1025th IP-address.

Finite State Machine

SPS uses the Diffie Hellman key exchanges to automatically recognize if the other node is secured or not (a secured node is one which communicates through a similar encryption unit) and to automatically build its Security Association, the so called permanent and volatile databases.

In this section we explore the protocol through which a secured node will recognize without human intervention whether the communicating node is secured or not, and if it is secured how they will automatically establish their common secret key. We will describe the process in terms of a finite state machine.

state 1: The encryption unit starts out having neither the static nor the dynamic common key with the communicating node. Consequently, packets from/to the communicating node are dropped. The unit transits into state 2 by initiating or responding to a static Diffie-Hellman key exchange (i.e. when the unit sends its static public key to the communicating unit).

state 2: The unit waits for the static public key from the communicating unit and upon receiving the key, it calculates the static common secret key, K_s. In this state there is still no crypto key; all packets from/to the communicating node are dropped. The unit transits into state 3 after the K_s is calculated, or into state 4 if no public key was received.

state 3: The communicating node is known to be secure and the two nodes share a static common key, which is used to encrypt/decrypt all packets from/to the communicating node. The unit transits into state 5 by initiating or responding to a dynamic Diffie-Hellman key exchange.

state 4: The communicating node is known to be secured, there is no common crypto key, all packets from/to the communicating node are 3.

state 5: The unit waits for the dynamic public key from the communicating unit and upon receiving the key it calculates the dynamic common secret key, K_d. The unit transits into state 6 upon calculating K_d or into state 7 if no public key was received. In state 5 there is still no dynamic crypto key, and all packets from/to the communicating node are encrypted/decrypted using the static crypto key.

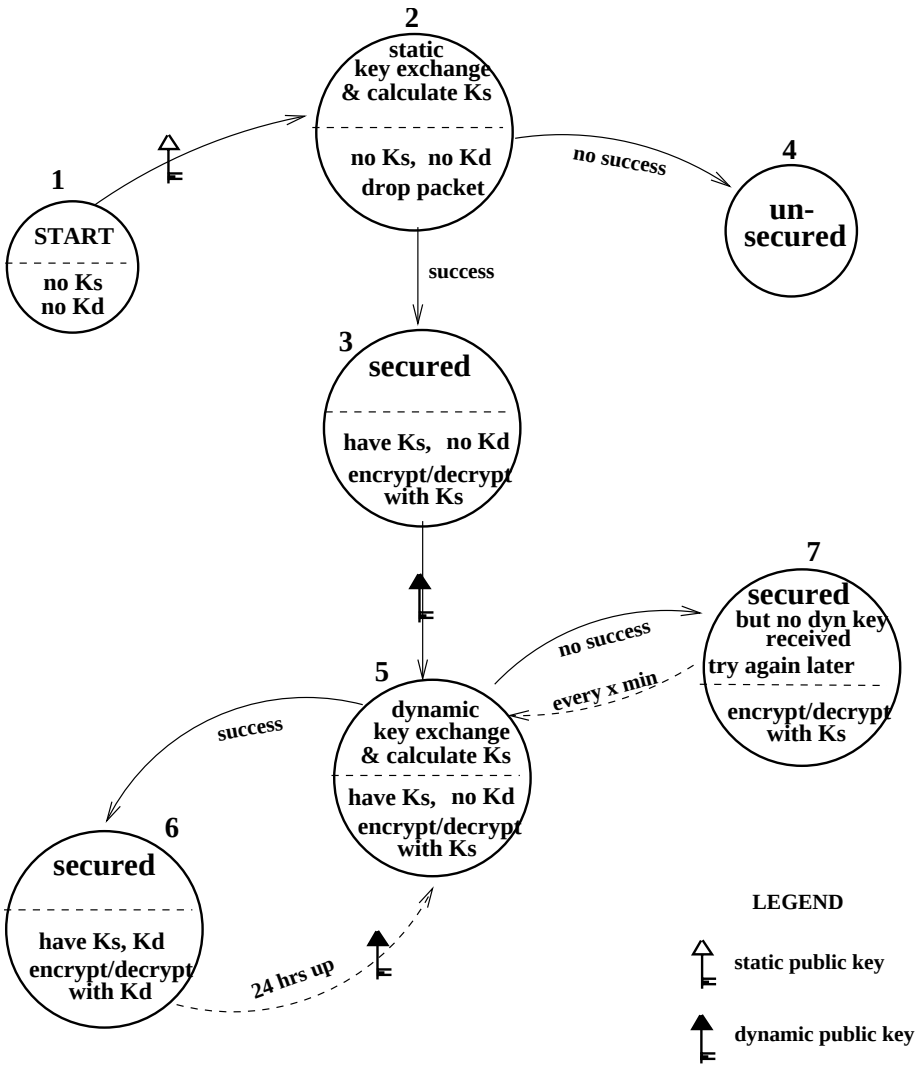


Figure 3. Finite State Machine of the automatic key exchange

state 6: The communicating node is secure and the two nodes share a dynamic common key, which is used to encrypt/decrypt all packets from/to the communicating node. Every 24 hours the node's dynamic keys are regenerated, and upon the first packet from/to the communicating node, the state transits into state 5 by initiating or responding to a dynamic Diffie-Hellman key exchange.

state 7: The communicating node is known to be secure, the two nodes share a static common key, but no dynamic public key was received, thus the dynamic common key could not be calculated. K_s is used to encrypt/decrypt all packets from/to the communicating node. Periodically (every x min) this state transits into state 5 to retry negotiating the dynamic key.

Smart Encryption

The protocol described in the previous section provides the so-called Smart Encryption, which all Fortress VPN

products utilize. Smart Encryption is a completely transparent feature that enables each product (hardware and software) to automatically recognize when encryption is necessary and when it is not. It eliminates the need for users to manually initiate encryption, because it recognizes a request to communicate with “secure” IP addresses and begins encrypting automatically. Likewise, Smart Encryption recognizes that encryption is not necessary for communications to non-secure IP addresses

Prevention of IP spoofing

Of a Communication Partner

After exchanging their static key each communication partner stores the IP address of the other, along with its common static key. After this, no one can spoof the client’s IP to another unit. Even if an intruder purchases a similar encryption device, he/she would not be able to masquerade as a legitimate user, since he/she has no way to obtain the common secret key of the spoofed node.

Of a Client

An encryption module that secures multiple nodes (clients) builds a MAC table with its client(s)’ IP and MAC addresses so it can deliver packets to its clients. This table also provides a means to prevent IP spoofing of any of its clients, since the source IP of each packet received via the NetFortress’ eth1 port is matched against the IP addresses listed in the MAC table.

Smart Bridge

Another benefit the MAC table provides is that packets from one client to another of the same encryption unit are forwarded internally, without the packet appearing on the “world-side” ethernet port.

Secure IP packet handling in SPS

When sending out an IP packet for transmission, the payload is compressed. This increases the throughput of data. However, the main purpose of the compression is not to save space, but to change and hide the actual size. The original size, which was recorded in the original IP header, is moved to the payload for later encryption.

The original MAC address in the MAC header of the packet is replaced by MAC address of the VPN device itself; nobody can see behind the VPN device from the outside ***(MAC hiding)***.

An FTI-header and an FTI-tail are added just before and after the compressed payload. The header, which will not be encrypted contains an internal protocol-number and a checksum. This checksum will be used to test the ***data integrity in transit***. The tail, which will be part of the encrypted payload contains information from the original IP header (protocol, length, fragment information) and a checksum. This latter, encrypted checksum is used to test the integrity of crypto keys and for ***strong authentication***.
The original protocol identifier, packet length and fragment-flag are moved from the IP header to the FTI tail. The IP protocol identifier is replaced with the Fortress Technologies protocol that reveals nothing about the original service. The original packet length is replaced by the new one.

The compressed IP payload, the original size, and the original transport protocol identifier are then encrypted with IDEA. The size of the new, encrypted payload is computed and inserted into the IP header. If, after compression and encryption, the new datagram exceeds the allowed size, the VPN units fragment the datagram before sending and defragments at the receiving end.

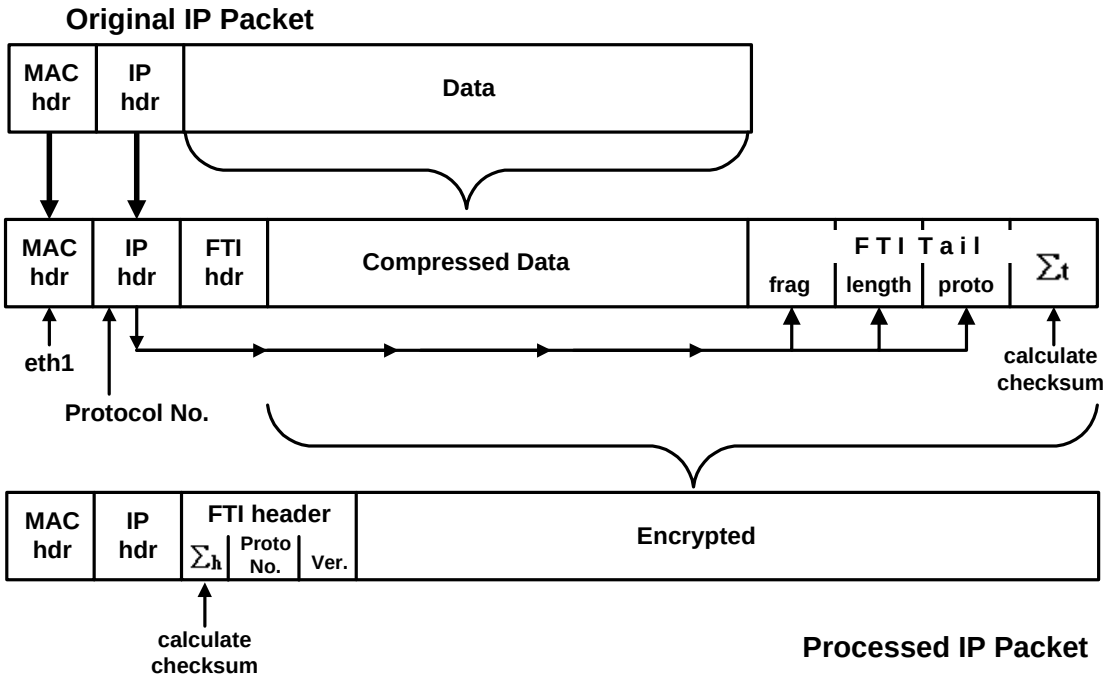


Figure 4. Illustration of the modification steps of the IP packet by the SPS protocol.

The new IP datagram exposes no information — except the source and destination IP addresses, the payload size that doesn't match the actual size, a generic transport protocol identifier which reveals nothing about the originating system, and an encrypted payload that can't be read.

Double-integrity checking

The checksum in the FTI-header on the sender's side is calculated after encryption and checked on the receiver's side before decryption, thus providing an ***integrity test of the data in transit***.

On the other hand, the checksum in the FTI-tail on the sender's side is calculated before encryption and checked on the receiver's side after decryption providing an ***integrity test of the encryption system.*** The checksums differ if the common secret key is not identical on both sides. If the tail-checksums differ after decryption with the dynamic common key, then the VPN unit tries to decrypt the packet with the static common key. If the checksums still differ, then the unit initiates a dynamic key exchange.

Authentication

The second, encrypted, checksum also serves as a means of ***strong authentication***[1] since the static and the dynamic common crypto keys used to encrypt the checksum are secret and shared by the sender and the receiver only.

This encrypted checksum in the dynamic key-packet (encrypted with the static common key) verifies that the dynamic public key is truly coming from the sender, thus authenticating the sender. The same encrypted checksum in a regular packet, encrypted with the dynamic common key, authenticates the packet (that the sender of the packet is who it claims to be).

[1] *Strong authentication is when someone can prove that he knows a secret without revealing it. The fact that after decryption the checksum in the packet agrees with the checksum calculated from the packet by the receiver proves that the sender and receiver share the same secret (key).*

MAC hiding

The original MAC address in the MAC header of the packet is replaced by MAC address of the VPN device itself preventing the outside world to see behind the VPN device.

IP locking

The first message from a secured client (whose communication is going through an encryption device) will identify the client(s) and its IP (and MAC) address will be burned into the nonvolatile memory of the device. From that point on the device is "bound" to its client(s); it cannot be stolen.

Secure Packet Shield Components

The full range of features in S*PS* technology are:

1. **Network layer encryption** (all header information is hidden) – only
2. **Encrypted Diffie-Hellman key exchange** (prevents man-in-the-middle attacks.)
3. **Dynamic key regeneration** - encryption keys are changed every 24 hours
4. **Strong Authentication**
   Sender: encrypted checksum in dynamic key packets assures that the parties share the same static common secret key
   Origin of the packet: encrypted checksum assures that the parties share the same dynamic common secret key
5. **IDEA encryption/decryption** - algorithm with 128-bit long key
6. **IP Locking**
7. **Client MAC address is hidden** in both Host and LAN type NetFortress VPN units
8. **Smart Bridge** – client-to-client communication is hidden from the outside
9. **Prevents IP spoofing** of a communication partner and IP spoofing of a client of a LAN type NetFortress™ VPN unit
10. **Double data integrity checking** (before and after encryption)
    a. integrity of data in transit
    b. integrity of encryption/decryption (keys & algorithm)
11. **Unique Company Signature** – key exchange (therefore communication) only within a company
12. **Built-in "Firewall" features**

[1] A secured node or partner is one which communicates through a similar encryption unit.