IPSec-based VPNs and related algorithms

By: Plamen Nedeltchev, Ph.D. – pnedeltc@cisco.com
Radoslav Ratchkov, B.S. in Mathematics - rado@lsil.com

February’2002

This article is trying to achieve two main objectives:

- The existing IPSec-based VPN solutions are based on Diffie-Hellman key agreement protocol, MD5 and SHA-1 message authentication protocols, DES/3DES encryption/decryption protocols, and Null-encryption protocol. The article will provide a concise description of the listed protocols.

- In the development and troubleshooting of VPN solutions, the engineer faces “a payload” which is encrypted, coded, and hashed, which makes the manual calculation of values cumbersome and time-consuming. For automated calculations, the article proposes www.ratckov.com/vpn.

Background

Virtual private networking (VPN) is about running private data over public networks. The anticipated growth of the VPN market in the following decade increases the existing variety of VPN standards, vendors, and solutions, thus creating an extensive area of expertise. The abundance of available sources sometimes is more descriptive than informative.

The RFC 2764 defines the framework for Virtual Private Networks running across the IP backbones. VPN tunneling precludes a “tunnel”, which generally means connecting two VPN endpoints as a basic building block. An IP tunnel operates as an overlay across the IP backbone, and the traffic sent through the tunnel is incomprehensible to the underlying IP backbone. In effect the IP backbone is being used as a link layer technology, and the tunnel forms a point-to-point link, sometimes referred to as “Wire in the Cloud”. There are numerous IP tunneling mechanisms, including IP/IP, Generic Routing Encapsulation (GRE) tunnels, Layer 2 Tunneling Protocol (L2TP), IPSec, and Multiprotocol Label Switching (MPLS), which were not initially considered as VPN mechanisms.

IPSec is considered the best choice, whenever there is a requirement for strong encryption or strong authentication. Originally, IPSec was conceived as an extension for IPv4 with added security features. Now IPSec is an Internet standard framework for the establishment and management of data privacy between network entities, based on the architecture model defined in RFC 2401\(^1\). IPSec VPNs use the services defined within IPSec to ensure confidentiality, integrity, and authenticity of data communications across public networks. A group of standards

---

\(^1\) RFC 2401 "Security Architecture for the Internet Protocol"
starting from RFC 2402, 2403, …, 2412 and numerous other protocols define all the IP-based VPN solutions existing in the industry.

Every IPSec-based VPN solution includes the following components:

- Security associations (SA)
- Authentication, digital certificates, and signatures
- Non-repudiation
- Key generation and management
- Data integrity
- Encryption

IPSec operates in a peer-to-peer relationship and refers to security associations (SA) as a convention (contract) between two parties, which establishment facilitates an IPSec-based conversation between two communicating parties. Each party (device, software client) must agree on the policies or rules of their conversation by negotiating these policies with their potential peer. Every security association (SA) is uniquely identified by an IP destination address, a security protocol (AH or ESP) identifier, and a unique Security Parameter Index (SPI). There are two types of SAs—Internet Security Association Key Management Protocol (ISAKMP) SAs (also known as IKE SAs) and IPSec SAs.

- The valid authenticating methods in IPSec are Pre-shared key, Digital signature standard (DSS) signatures, RSA signatures, Encryption with RSA, Revised encryption with RSA².

- Non-repudiation prevents a party involved in a communication from later denying having participated, requires proof of identity of the sender, and is based on digital signatures and mathematical algorithms.

According to the conformance requirements of RFC 2406³, a compliant ESP implementation MUST support the following mandatory-to-implement algorithms:

- HMAC with MD5.
- HMAC with SHA-1.
- DES in CBC mode.
- NULL Authentication algorithm
- NULL Encryption algorithm

The listed algorithms exceed the requirements for IP AH⁴ and cover all of them for IP ESP. They are freely available in the public domain.

---


³ RFC 2406. “IP Encapsulating Security Payload (ESP)”

⁴ RFC 2402. “IP Authentication Header”
1. The Diffie-Hellman key agreement protocol.

Public-key cryptography and digital signatures offer solutions that address the requirement for secure digital communication. Many alternative techniques have been proposed; however, there has been no single, comprehensive reference defining a full range of common public-key techniques covering key agreement, public-key encryption, digital signatures, and identification from several mathematical families, such as discrete logarithms, integer factorization, and elliptic curves.

In the Diffie-Hellman\(^5\) key agreement (also called exponential key agreement), the two parties, without any prior arrangements, can agree upon a secret key that is known only to them (and, in particular, is not known to an eavesdropper listening to the dialogue by which the parties agree on the key). This secret key can then be used to encrypt further communications between the parties. This key is primarily used for public key exchange for use by some other private key type crypto system.

The Diffie-Hellman key agreement is an integral part of IPSec standard. The key management in IPSec begins with the overall framework called Internet Security Association and Key Management Protocol - ISAKMP. The Internet Key Exchange (IKE) protocol is defined within that framework. IKE relies on another protocol known as OAKLEY, which uses Diffie-Hellman. Cisco uses ISAKMP/OAKLEY as its main key exchange technique.

1.1 Cisco Key management

Internet Key Exchange provides three modes for exchanging key information and setting up IKE SAs. The first two modes are Phase 1 exchanges, which are used to set up the initial secure channel. The other mode is the Phase 2 exchange, which negotiates IPSec SAs. The two modes in Phase 1 are “Main mode” and “Aggressive mode”, and the Phase 2 mode is called “Quick mode”. The basic idea is to bootstrap an IKE SA to provide a protected pipe for subsequent protected IKE exchanges between the IKE peers, and then use phase 2 “Quick mode” with the IKE SA to negotiate the IPSec SAs. Only IPSec uses the IPSec SA for protecting traffic.

The “Main mode” has three two-way exchanges between the initiator and the receiver. In the first exchange, the algorithms and hashes are agreed upon. The second exchange uses Diffie-Hellman to agree on a shared secret and to pass nonces\(^6\). The third exchange verifies the identity of the other side.

In the “Aggressive mode”, fewer exchanges with fewer packets are necessary. On the first exchange, almost everything is squeezed in - the proposed SA (algorithm, hashes, and mode), the Diffie-Hellman public value, a nonce that the other party signs, and an ID packet, which can be used to verify identity via a third party. The receiver sends back everything that is needed to complete the exchange. The only thing left is for the initiator to confirm the exchange. The “Aggressive mode” is faster than the “Main mode”; however, its potential weakness is that in

---

\(^5\) The Diffie-Hellman key agreement protocol was developed by Whitfield Diffie and Martin Hellman in 1976 and published in "New Directions in Cryptography."

\(^6\) Nonces is a random numbers sent to the other party, signed and returned to prove their identity - they are signed only if encrypted nonces or digital signatures are being used
aggressive mode both sides have exchanged information before the establishment of a secure channel. Therefore, it is theoretically possible to intercept the transmission and discover who formed the new SA.

1.2 Methodology of Diffie-Hellman algorithm.
Diffie-Hellman is not a conventional encryption mechanism because we do not use it to encrypt data. Instead, it is a method for secure exchange of keys that are subsequently used for data encryption. Diffie-Hellman accomplishes this secure exchange by creating a "shared secret" (sometimes called a "key encryption key") between two devices. The shared secret then encrypts the symmetric key (or "data encryption key" - DES, Triple DES, CAST, IDEA, Blowfish, etc.) for secure transmission. **The basis for the technique is the difficulty of calculating logs in modular arithmetic.** An asymmetric key system uses two keys - one called the "private key" that the user keeps secret and one called the "public key" that can be shared with the world. Unfortunately, the asymmetric key systems are extremely slow for any sort of bulk encryption. Today, it is a typical practice to use a symmetric system to encrypt the data and an asymmetric system to encrypt the symmetric keys. That is precisely what Diffie-Hellman is capable of doing - and does do when used for key exchange as described here.

1.3 Key exchange process

The process of key exchange begins when each side of the communication generates a private key. Each side then generates a public key, which is a derivative of the private key. The two systems then exchange their public keys. Each side of the communication now has its own private key and the other system's public key.

**Parameter generation.**
The central authority or vendor selects two parameters:

- Integer \( p \) called prime\(^7\),
- Integer \( g \), called base or generator, where \( 0 < g < p \)
- Integer \( l \), the private-value length in bits - optionally chosen by the vendor, that satisfies \( 2^{l - 1} \leq p \)
- The length of the prime \( p \) in octets is the integer \( k \) satisfying \( 2^{8(k - 1)} \leq p < 2^{8k} \)

The algorithm is based on calculation of modulo, which returns the remainder after a number is divided by a divisor. If \( p \) is sufficiently large, \( n \) cannot be discerned from the result, which makes the algorithm a preferable technique for key agreement in VPN. Modulo \( p \) from \( n \) can be expressed using the INTEGER function and the equation:

\[
\text{Modulo}(n, p) = n - p \times \text{INT}(n/p)
\]

And here is the main equation of Diffie-Hellman for calculating an integer secret key \( S \), which satisfies the following:

---

\[
S = (B)^x = (g^y)^x = (g^x)^y = A^y \mod p,
\]
where \(x\) and \(y\) are private values\(^8\). This mathematical relationship is the reason the two entities arrive at the same key.

The algorithm includes the following steps – see the Example\(^ 9\).

Example.
Let’s assume that two parties in the process are: Initiator (side A) and Receiver (side B).

Let’s assume that both parties don’t have any agreement on parameters. Let’s say side I uses a public generator\(^10\) \(g=3\), public prime modulus \(p=A48B\) and a secret exponent \(x=1A5F\). After the calculations, side I sends the following to side B: \(g = 3\), \(p = A48B\), \(A = 4B0\). Now, side R does not select a session key directly but rather selects a random number \(y=1ABD\) from which it calculates the session key \(S = 7C86\). In order for side R to send to side I the session \(S\) key, side R only calculates \(B\) and \(B=40A1\). Now side I calculates \(S\), based on a newly received \(B\) and its own \(x\), which is \(S = 7C86\). Now both parties have the session key and can exchange messages.

---

\(^8\) The expression \(x = y \mod m\) reads as "\(x\) is equivalent to \(y\), modulo \(m\)". The statement \(a+kp=a \mod p\) should be fairly obvious. **Chinese Remainder Theorem:** if \(x = y \mod p\) and \(x = y \mod q\), then \(x = y \mod pq\), if \(p\) and \(q\) are coprimes. **Fermat/Euler Theorem** rules that if \(p\) is prime, \(x^{p-1} = 1 \mod p\). RSA correctness rules that if \(d\) and \(e\) are generated, such that \(de=k(p-1)(q-1)+1\), and \(k\) is an integer, the encryption/decryption process can be written as \(x^{ed} \mod pq\). [Jones Burton W. Modular arithmetic, Blaisdell Publishing Company, 1964]


\(^{10}\) All numbers in this example are in hexadecimal format
using this shared-key as the key to a symmetric cipher such as RC4. Note that \( x \) and \( y \) has never change hands, and neither has \( S \). Side I can not discover \( y \) and side R can not discover your \( x \), and generally speaking it is not necessary. Noting that the public key is a derivative of the private key is important - the two keys are mathematically linked. However, in order to trust this system, you must accept that you cannot discern the private key from the public key and that’s what requires trust in the mathematical experts. At this point, the Diffie-Hellman operation could be considered complete.

The cost of some methods for computing discrete logarithms depends on the length of the prime, while the cost of others depends on the length of the private value. The purpose of selecting a private-value length is to reduce the computation time for key agreement, while maintaining a given level of security. The time to execute the algorithm by software is proportional to \( D^3 \), where \( D \) is the number of bits of \( p \). Thus, going up from 200 to 800 bits raises the complexity by a factor of 64.

The length of the primes defined so called DH-Groups, or Oakley Default Groups. They are:

- Group 1 - default 768-bit (300h)
- Group 2 - alternate 1024-bit (400h) - recommended
- Group 3 - EC2N group on GP[2^155]
- Group 4 - EC2N group on GP[2^185], and
- Group 5- prime length 1536 bits (600h).

1.4 The real numbers

The previously described simple example with only 16 bit keys is useless for security purposes. The algorithm defines groups of primes \( p \) to choose from. Diffie-Hellman’s well-know group I uses a 768 bit (0x300) key and it is typical for Altiga 3.0 concentrator, and Cisco’s VPN Client 2.5.x and 2.6.x. Diffie-Hellman well-known group II uses a 1024 (0x400) bit prime key \( p \), typical for Cisco VPN 3000 series concentrator and Cisco VPN Client 3.02 and higher. More about this topic can be seen at [www.dns.net/dnsrd/rfc/rfc2539.html](http://www.dns.net/dnsrd/rfc/rfc2539.html), where the prime for 1024 bit prime can be calculated using \( p = 2^{1024} - 2^{960} - 1 + 2^{64} \times \{ [2^{894} \times \pi] + 129093 \} \). For example:

Let \( g = 3 \)

Let \( p = (DH \text{ Group II} \text{ – total of 128 octets}) \)

de9b707d4c5a4633c0290c95ff30a605aeb7ae864ff48370f13cf01d49adb9f2
3d19a439f753ee7703cf342d87f431105c843ec78ca4df639931f3458fae8a94d
1687e99a76ed99d0ba87189f42fd3ad8262c54a8cf5914ae6c28ec540d714a5f
6087a171fb74f8414cf6f968d72386ef356a05180c3bec7dd5ef6fe76b0531c3

Then, \( A = 56C03667F3B50335AD532D0ADCAA2897A02C0878099D8E3AAB9D808B2B5C83E2F \)

14C78E664BCE7D209E0FD8B73F7F6822FCDF6FFADE5AF2DDBB38FF3D2270CE
BBED172D7C399F47EE9F1067F1B85CCBE8F43B721B4F9802F3EA51A8ACD1F6F
B526ECF4A56AD62B0AC17551727B6A7C7AADB9362394B410611A21A7711DCDE2
1.5 Potential Weaknesses

However, the Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack, because Diffie-Hellman key exchange does not authenticate the participants. The attack involves someone intercepting both public keys and forwarding bogus public keys of their own. The Man In the Middle (MIM) can potentially intercepts encrypted traffic, decrypts it, copies or modifies it, re-encrypts it with the bogus key, and forwards it on to its destination. If successful, the parties on each end would have no idea that there is an unauthorized intermediary. However, this is an extremely difficult attack to perform outside the laboratory.

The authenticated Diffie-Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 to defeat the man-in-the-middle attack on the Diffie-Hellman key agreement protocol\textsuperscript{11}. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public-key certificates.

One of the Cisco “trusted-user” solutions includes using the XAUTH, which stands for eXtended AUTHenticaion and was introduced to the standards to address the user authentication issue. XAUTH allows the remote user to authenticate himself to RADIUS/TACACS+ authentication servers and as a result allows per-user and per-session authentication of IPSec tunnels. When using either pre-shared keys or certificates, if a device comes under malicious control, it is possible to access the secure network, unless some form of authentication exists. Using XAUTH eliminates this risk as a user name and password are additionally required. XAUTH also provides for compatibility with existing token cards.

Another Cisco solution is using the digital certificates. The digital certificate provides mathematical proof that the sender is really who he claims to be or that the data has not been changed in transit. Additionally, the certificate provides scalable authentication. The digital certificate, signed by Certificate Authority (CA) contains the following information: Serial number; Validity dates; Issuer’s name; User’s (subject’s) name; and Subjects public Key Info. Cisco provides information for the three major Certificate authorities at www.cisco.com.

1.6 Conclusion

Once the secure exchange of the symmetric key is complete (and note that passing that key is the whole point of the Diffie-Hellman operation), data encryption and secure communication can occur. The longer the symmetric key in use, the easier it is to perform a successful cryptanalytic attack against it. Therefore, changing keys frequently is important. Changing keys is called re-keying and it is closely related to the SA Lifetime, where the SA Lifetime determines the period of time for which a security association is valid. That’s why both parties usually negotiate in the very beginning the so-called ‘Rekey Time Interval’. Both sides of the communication still have the shared secret and it can be used to encrypt future keys at any time and any frequency desired.

\textsuperscript{11} www.sans.org/infosecFAQ/encryption/diffie.htm
Re-keying is closely related to another feature, called **Perfect Forward Secrecy (PFS)**. The RFC 2401 defines the PFS, where a shared secret encryption key refreshing involves combining the current key with a random number to create a new key. PFS enforces the re-calculation of the shared secret key from scratch using the public and private key generation and DH techniques. The reason for the recalculation is to avoid a situation in which a hacker may have derived a particular secret key. PFS means that a new key can be calculated that has no relationship to the preceding key.
2. Keyed-Hash Message Authentication Code (HMAC)

The Message Authentication Codes (MAC) are used in the world of open communications and computing to ensure the integrity of the data transmitted between two parties. They are based on a secret key available to both parties. Codes, based on a cryptographic hash functions are called HMAC. RFC2104 defines HMAC as:

\[ H(K \ xor \ opad, H(K \ xor \ ipad, \text{text})) \]

Where “xor” denotes exclusive ”or”, H denotes a cryptographic hash function, K denotes a secret key and ‘text’ is the input text stream. HMAC can use any cryptographic hash function, MD5 and SHA-1 as an example, and as such, HMAC is at least as strong as the hash function it uses. This also allows for easy cryptographic analysis of the HMAC, where only the underlying hash function is being analyzed.

Assuming that H is a cryptographic hash function that iterates over block of data, with byte length B of each block, then \textbf{ipad} and \textbf{opad} are defined to be fixed length strings as follows:

- ipad = 0x36 repeated B times
- opad = 0x5C repeated B times

Let’s also assume L to be the byte-length of the hash output. The following table represents the values of B and L for different hash functions:

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>SHA-1</td>
<td>64</td>
<td>20</td>
</tr>
<tr>
<td>DES</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3DES</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

The secret key K can be of any length up to B. If an application uses longer keys, the hash of the key K is used instead. In such cases, the length of the key K will be L, which is also the minimum recommended length for keys K.

To compute the HMAC of text ‘text’, the following steps are performed in following order:

a) If K is longer that B bytes, assume K = H(K)

b) Append zeroes to K, such that K is of length B
c) Compute K XOR ipad
d) Append ‘text’ to the result of (b)e) Compute H on the result of (c)f) Compute K XOR opad
g) Append the result of (d)h) Compute H on the result of (f)

---

12 IETF RFC2104. “The Use of HMAC-SHA-1-96 within ESP and AH”
It is common practice that the output of H be truncated to a given length, so only part of the bits is outputted. As suggested in [RFC 2104] the resulting length should not be less than half of the original output length. Such truncated outputs should be denoted as HMAC-H-t, where H is the hash function and t is the length. For example, HMAC-SHA-1-96 denotes SHA-1 HMAC with its output truncated to be 96 bits.

Preneel and van Oorschot show some advantages of such truncation\(^\text{13}\). To an attacker, less information on the hash results is available, but at the same time there will be fewer bits to predict.

\(^{13}\) Bart Preneel, Paul C. van Oorschot, “MDx-MAC and Building Fast MACs from Hash Functions”
3. Message Digest 5 (MD5)

Message Digest 5 (MD5) is the Cisco choice for Remote access VPN solutions. MD5 algorithm was developed by Ronald L. Rivest and defined in RFC1321\(^{14}\). It is enhancement and direct replacement for MD4\(^{15}\), which was found to be weak. As a classical trade-off example, MD5 is slower but at the same time it is much stronger.

MD5 is a hashing algorithm, where the first block is hashed using initial seed values, next block are hashed using the result of the previous block, and the result of the last block is the hash value of the message, or also called “message digest, 128 bytes long. This technique is called block-chained cipher.

3.1 Terminology and notations

Let’s begin by defining some of the terminology we are going to use later to describe the MD5 algorithm. “Byte” defines a sequence of 8-bits, with the right-most bit being the least significant and the left-most bit being the most-significant. “word” denotes sequence of 4 bytes, or 32 bits, with the left-most being the least significant, and the right-most being the most significant. “double-word” denotes sequence of two words, or 64 bits, with the left most word being the least significant, and the right-most – the most significant. Note, that the definition of “word” defines a little-endian\(^ {16}\) scheme, so for big-endian\(^ {17}\) platforms, special steps need to be taken to reorder the bytes from the input stream. \(x[n]\)” denotes a subscript, i.e. the \(n\)th element of block \(x\). We use “\(^{\wedge}\)” to denote power, for example, \(2^{32}\) is “2 on power 32”. We use “\(!\)” to denote “bitwise not” operation. This operation inverts each bit to its opposite – “0” to “1” and “1” to “0”. We use “\((+)\)” to denote “exclusive or” operation. We use “\(+\)” to denote addition modulo \(2^{32}\). We denote the “bitwise and” operation of two words \(X\) and \(Y\) as \(XY\). We use “\(X \ll\ll s\)” operation to denote rotational shift left of word \(X\) by \(s\) bits. This operation can be expressed using “\(<<\)” (shift-left) and “\(>>\)” (shift-right) functions as:

\[
X \ll\ll s = X \ll s + X >> (s-32)
\]

\(^{14}\) RFC 1231. “The MD5 Message-Digest Algorithm”

\(^{15}\) RFC 1320. “The MD4 Message-Digest Algorithm”

\(^{16}\) Describes a computer architecture in which, within a given 16- or 32-bit word, bytes at lower addresses have lower significance (the word is stored ‘little-end-first’)

\(^{17}\) Describes a computer architecture in which, within a given multi-byte numeric representation, the most significant byte has the lowest address (the word is stored ‘big-end-first’)}
3.2 Message padding

Let’s assume \( L \) is the length of the initial message in bits. First, we append “0”s to the end of the message until its length \( N \) is congruent to 448. At minimum no bits are appended, at maximum 448 “0”s are appended. Next, we append the double-word representation of \( L \). If \( L \) is larger than \( 2^{64} - 1 \), only the last 64 bits are appended. With this our message has a length, which is multiple of 512.

3.3 Round functions

We will define four functions, or also called round functions, as follows:

\[
\begin{align*}
F(X, Y, Z) &= XY + (!X)Z \\
G(X, Y, Z) &= XZ + Y(!Z) \\
H(X, Y, Z) &= X (+) Y (+) Z \\
I(X, Y, Z) &= Y (+) (X + !Z)
\end{align*}
\]

3.4 Initial seed

Let’s assume:

\[
\begin{align*}
A &= 0x01234567 \\
B &= 0x89abcdef \\
C &= 0xfedcba98 \\
D &= 0x76543210
\end{align*}
\]

We also seed block \( T \) of 64 words such that \( T[n] \) is equal to the integer part of \( \frac{4294967296}{\text{times } \sin(n)} \), with \( n \) between 1 and 64. Below are given the exact values of \( T \) for each \( n \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( T )</th>
<th>( n )</th>
<th>( T )</th>
<th>( n )</th>
<th>( T )</th>
<th>( n )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xD76AA478</td>
<td>17</td>
<td>0xF61E2562</td>
<td>33</td>
<td>0xFFFFA3942</td>
<td>49</td>
<td>0xF4292244</td>
</tr>
<tr>
<td>2</td>
<td>0xE8C7B756</td>
<td>18</td>
<td>0xC040B340</td>
<td>34</td>
<td>0x8771F681</td>
<td>50</td>
<td>0x432AFF97</td>
</tr>
<tr>
<td>3</td>
<td>0x242070DB</td>
<td>19</td>
<td>0x265E5A51</td>
<td>35</td>
<td>0x6D9D6122</td>
<td>51</td>
<td>0xAB9423A7</td>
</tr>
<tr>
<td>4</td>
<td>0xC1BDCEE</td>
<td>20</td>
<td>0xE9B6C7AA</td>
<td>36</td>
<td>0xFDE5380C</td>
<td>52</td>
<td>0xFC93A039</td>
</tr>
<tr>
<td>5</td>
<td>0xF57C0FAF</td>
<td>21</td>
<td>0xD62F105D</td>
<td>37</td>
<td>0xA4BEEAA44</td>
<td>53</td>
<td>0x655B59C3</td>
</tr>
<tr>
<td>6</td>
<td>0x4787C62A</td>
<td>22</td>
<td>0x02441453</td>
<td>38</td>
<td>0x4BDECFA9</td>
<td>54</td>
<td>0x8F0CCC92</td>
</tr>
<tr>
<td>7</td>
<td>0xA8304613</td>
<td>23</td>
<td>0xD8A1E681</td>
<td>39</td>
<td>0xF6BB4B60</td>
<td>55</td>
<td>0xFFEFF47D</td>
</tr>
<tr>
<td>8</td>
<td>0xFD469501</td>
<td>24</td>
<td>0xE7D3FBC8</td>
<td>40</td>
<td>0xBE8FBC70</td>
<td>56</td>
<td>0x85845DD1</td>
</tr>
<tr>
<td>9</td>
<td>0x698098D8</td>
<td>25</td>
<td>0x21E1CDE6</td>
<td>41</td>
<td>0x289B7EC6</td>
<td>67</td>
<td>0x6FA87E4F</td>
</tr>
<tr>
<td>10</td>
<td>0x8B44F7AF</td>
<td>26</td>
<td>0xC33707D6</td>
<td>42</td>
<td>0xEAA127FA</td>
<td>68</td>
<td>0xFE2CE6E0</td>
</tr>
<tr>
<td>11</td>
<td>0xFFFF5BB1</td>
<td>27</td>
<td>0xF4D50D87</td>
<td>43</td>
<td>0xD4EF3085</td>
<td>69</td>
<td>0xA3014314</td>
</tr>
<tr>
<td>12</td>
<td>0x895CD7BE</td>
<td>28</td>
<td>0x455A14ED</td>
<td>44</td>
<td>0x04881D05</td>
<td>60</td>
<td>0xE80811A1</td>
</tr>
<tr>
<td>13</td>
<td>0x6B901122</td>
<td>29</td>
<td>0xA9E3E905</td>
<td>45</td>
<td>0xD9D4D039</td>
<td>61</td>
<td>0xF7537E82</td>
</tr>
<tr>
<td>14</td>
<td>0xFD987193</td>
<td>30</td>
<td>0xFECEFA3F8</td>
<td>46</td>
<td>0xE6DB99E5</td>
<td>62</td>
<td>0xBD3AF235</td>
</tr>
<tr>
<td>15</td>
<td>0xA679438E</td>
<td>31</td>
<td>0x676F02D9</td>
<td>47</td>
<td>0x1FA27CF8</td>
<td>63</td>
<td>0x2AD7D2BB</td>
</tr>
<tr>
<td>16</td>
<td>0x49B40821</td>
<td>32</td>
<td>0x8D2A4C8A</td>
<td>48</td>
<td>0xC4AC5665</td>
<td>64</td>
<td>0xEB86D391</td>
</tr>
</tbody>
</table>
3.5 Calculating the message digest

We process the message in blocks of 16 words, which will be denoting X. Calculating the message digest consists of four rounds of calculations, each of them consisting of 16 steps. In a generic form each step can be written as\(^{18}\):

\[ A = B + (A + \Phi(B, C, D) + X[n] + T[m]) \ll s \]

Where \(\Phi\) is a round dependant function, and \(T\) is a step dependant constant.

For each block X we repeat the following steps.

Lets \(AA = A, BB = B, CC = C, DD = D\).

Round 1

\[
\begin{align*}
A &= B + (A + F(B, C, D) + X[0] + T[1]) \ll 7 \\
D &= A + (D + F(A, B, C) + X[1] + T[2]) \ll 12 \\
C &= B + (C + F(D, A, B) + X[2] + T[3]) \ll 17 \\
B &= B + (B + F(C, D, A) + X[3] + T[4]) \ll 22 \\
A &= B + (A + F(B, C, D) + X[4] + T[5]) \ll 7 \\
D &= A + (D + F(A, B, C) + X[5] + T[6]) \ll 12 \\
C &= B + (C + F(D, A, B) + X[6] + T[7]) \ll 17 \\
B &= B + (B + F(C, D, A) + X[7] + T[8]) \ll 22 \\
A &= B + (A + F(B, C, D) + X[8] + T[9]) \ll 7 \\
D &= A + (D + F(A, B, C) + X[9] + T[10]) \ll 12 \\
C &= B + (C + F(D, A, B) + X[10] + T[11]) \ll 17 \\
B &= B + (B + F(C, D, A) + X[11] + T[12]) \ll 22 \\
A &= B + (A + F(B, C, D) + X[12] + T[13]) \ll 7 \\
D &= A + (D + F(A, B, C) + X[13] + T[14]) \ll 12 \\
C &= B + (C + F(D, A, B) + X[14] + T[15]) \ll 17 \\
B &= B + (B + F(C, D, A) + X[15] + T[16]) \ll 22 \\
\end{align*}
\]

Round 2

\[
\begin{align*}
A &= B + (A + G(B, C, D) + X[1] + T[17]) \ll 5 \\
D &= A + (D + G(A, B, C) + X[6] + T[18]) \ll 9 \\
C &= B + (C + G(D, A, B) + X[11] + T[19]) \ll 14 \\
B &= B + (B + G(C, D, A) + X[0] + T[20]) \ll 20 \\
A &= B + (A + G(B, C, D) + X[5] + T[21]) \ll 5 \\
D &= A + (D + G(A, B, C) + X[10] + T[22]) \ll 9 \\
C &= B + (C + G(D, A, B) + X[14] + T[23]) \ll 14 \\
\end{align*}
\]

B = B + (B + G(C, D, A) + X[4] + T[24]) <<< 20
D = A + (D + G(A, B, C) + X[14] + T[26]) <<< 9
C = B + (C + G(D, A, B) + X[3] + T[27]) <<< 14
B = B + (B + G(C, D, A) + X[8] + T[28]) <<< 20
C = B + (C + G(D, A, B) + X[12] + T[31]) <<< 20
B = B + (B + G(C, D, A) + X[13] + T[32]) <<< 20

Round 3

A = B + (A + H(B, C, D) + X[5] + T[33]) <<< 4
D = A + (D + H(A, B, C) + X[8] + T[34]) <<< 11
C = B + (C + H(D, A, B) + X[11] + T[35]) <<< 16
B = B + (B + H(C, D, A) + X[14] + T[36]) <<< 23
A = B + (A + H(B, C, D) + X[1] + T[37]) <<< 4
D = A + (D + H(A, B, C) + X[4] + T[38]) <<< 11
C = B + (C + H(D, A, B) + X[7] + T[39]) <<< 16
B = B + (B + H(C, D, A) + X[10] + T[40]) <<< 23
A = B + (A + H(B, C, D) + X[12] + T[41]) <<< 4
D = A + (D + H(A, B, C) + X[0] + T[42]) <<< 11
C = B + (C + H(D, A, B) + X[3] + T[43]) <<< 16
B = B + (B + H(C, D, A) + X[6] + T[44]) <<< 23
C = B + (C + H(D, A, B) + X[12] + T[47]) <<< 16
B = B + (B + H(C, D, A) + X[2] + T[48]) <<< 23

Round 4

A = B + (A + I(B, C, D) + X[0] + T[49]) <<< 6
D = A + (D + I(A, B, C) + X[7] + T[50]) <<< 10
C = B + (C + I(D, A, B) + X[14] + T[51]) <<< 15
B = B + (B + I(C, D, A) + X[5] + T[52]) <<< 21
A = B + (A + I(B, C, D) + X[12] + T[53]) <<< 6
D = A + (D + I(A, B, C) + X[3] + T[54]) <<< 10
C = B + (C + I(D, A, B) + X[10] + T[55]) <<< 15
B = B + (B + I(C, D, A) + X[1] + T[56]) <<< 21
A = B + (A + I(B, C, D) + X[8] + T[57]) <<< 6
D = A + (D + I(A, B, C) + X[15] + T[58]) <<< 20
C = B + (C + I(D, A, B) + X[6] + T[59]) <<< 15
B = B + (B + I(C, D, A) + X[13] + T[60]) <<< 21
C = B + (C + I(D, A, B) + X[2] + T[63]) <<< 15
B = B + (B + I(C, D, A) + X[9] + T[64]) <<< 21

A = A + AA
B = B + BB
C = C + CC
D = D + DD

3.6 Output

The output of the above iterations is given by (A, B, C, D)

3.7 Performance considerations.

As described in [RFC 1810], MD5 is a sequential algorithm. The resulting digest for each 512bit blocks is the seed, used to compute the digest of the next block. As such, it cannot be parallelized.

As mentioned before, MD5 is a little-endian algorithm, thus it is best suited for such hardware platforms. When used on big-endian platform or for network applications, that use big-endian architecture, there is a significant overhead from reordering the bytes.

3.8 HMAC-MD5 and HMAC-MD5-96

HMAC-MD5-96 is a HMAC that uses MD5 for its hash function, and the resulting hash is truncated to 96 bits. Its performance is that of MD5. Although some of the rounds in MD5 have been successfully attacked, there is no known attack against this HMAC at this time.
4. Secure Hash Algorithm (SHA-1)

Securr Hash Algorithm (SHA-1)\(^\text{19}\) is the Cisco’s preferable choice for LAN-to-LAN, (Site-to-Site) VPN solutions. Given the fact that the resulting digest is longer than MD5, it is considered stronger.

The algorithm starts with 5 words of data, also called initial seed, and calculates 5 words digest for the first 512 bits of the input stream. The result of this calculation is 5 words, and is used as seed for calculating the digest of the next block. The result after calculating the digest for the last block is also the digest for the input stream, or also is called “message digest with length of 160 bytes. As you can see this is the same block-chained cipher technique. This algorithm computes the digest for message with length of up to 2^64 bits.

![SHA-1 Digest Algorithm Diagram]

4.1 Terminology and Notations

Let’s first define some of the terms and notations we are going to use. “byte” denotes a sequence of 8 bits, with the right-most bit being the least significant, and the left-most bit – the most significant. “word” denotes a sequence of 4 bytes, or 32bits, with the right most byte being the least-significant, and the left-most – the most significant. “double word” denotes a sequence of two words, or 64 bits, with the right-most word being the least significant, and the left-most word – most significant. This defines big-endian scheme. (Remember that MD5 uses little-endian scheme).

We use “!” to denote “bitwise not” operation. This operation inverts each bit to its opposite – “0” to “1” and “1” to “0”. We use “(+)” to denote “exclusive or” operation. We use “*” to denote bit-wise logical “or” operation. We denote the bitwise and operation of two words X and Y as XY. We use “X <<< s” operation to denote rotational shift left of word X by s bits. This operation can be expressed using “<<<” (shift-left) and “>>” (shift-right) functions as:

\[ X <<< s = X << s + X >> (s-32) \]

4.2 Message padding.

Similar to MD5, SHA-1 requires the initial message to be padded with bits. Let’s assume the length of the initial message “text” in bits is L. The scheme for padding the “text” in SHA-1 is as follows:

\(^{19}\) FIPS PUB 180-1. “SECURE HASH STANDARD”
a) “1” is appended to the initial message
b) Append “0”的 “text” until its length is congruent to 448 modulo 512. Thus at these two steps at minimum 1 bit is appended and at maximum 448 bits are appended.
c) Append double-word representation of L at the end of the message.

After the above steps the length of the message is congruent to 0 modulo 512.

Let’s look at the following example. Let’s assume the initial message “text” is equal to “abc”. Its binary representation is:

01100001 01100010 01100011

After step a) the message is:

01100001 01100010 01100011 1

Since the length of this message is 25, we need to append 423 “0”s to it. The message will be:

01100001 01100010 01100011 10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Next, we need to append the double-word representation of L. Since L = 24, it representation is:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00011000

Our complete initial message in hexadecimal representation will be:

61626380 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

4.3 Initial seed.

Let’s assume:

H0 = 0x67452301
H1 = 0xefcdab89
H2 = 0x98badcfe
H3 = 0x10325476
H4 = 0xc3d2e1f0

4.4 Constants and functions

Let’s t is an integer in [0, 79]. As per [FIPS180-1] we define two functions K and f. Let’s assume
is a constant function K(t) as follows:

K(t) = 0x5a827999, for each t in [0 ,19]
K(t) = 0x6ed9eba1, for each t in [20, 39]
K(t) = 0x8f1bbcdc, for each t in [40, 59]
K(t) = 0xca62c1d6, for each t in [60, 79]

Depending on the implementation, K(t) can be assumed to be a sequence of 80 constant words.
Let’s also define logical function f(t, B, C, D), where B, C and D are 32 bit words as follows:

f(t, B, C, D) = BC + (!B)D, for each t in [0, 19]
f(t, B, C, D) = B (+) C (+) D, for each t in [20, 39]
f(t, B, C, D) = BC + BD + CD, for each t in [40, 59]
f(t, B, C, D) = B (+) C (+) D, for each t in [60, 79]

4.5 Calculating the message digest.

As defined in [FIPS1801] the final padded message is used for calculating the message digest.
The message is regarded as a sequence of 512 bit blocks. Two 5 word buffers are used, each
marked as H0, H1, H2, H3, H4 and A, B, C, D, E. We will also use one word TEMP.
The following steps are performed in a sequential manner for each block:

a) Divide the block into 16 words W(0) to W(15), with W(0) being the let-most word.
b) Compute W(t) for t in [16, 79] as follows:

W(t) = (W(t-3) (+) W(t-8) (+) W(t-14) (+) W(t-16)) <<< 1

c) Let :

A = H0
B = H1
C = H2
D = H3
E = H4

d) For each t in [0, 79] compute:

TEMP = A <<< 5 + f(t, B, C, D) +E + W(t) + K(t)
E = D
D = C
C = B <<< 30
B = A
A = TEMP

e) Let:

H0 = H0 + A
H1 = H1 + B
H2 = H2 + C
H3 = H3 + D
H4 = H4 + E

After computing steps a) through e) the message digest is defined as:

H0 H1 H2 H3 H4

4.6 Performance considerations

As defined in 4.1 SHA-1 algorithm is best suited for big-endian platforms and applications. When implementing on little ending platforms special byte-reordering needs to be performed on the input stream.

At the time of writing of this document there is no known attack against SHA-1. One of the main differences is the fact that the constants array is built based on the message block, thus adding extra strength to the algorithm and making it impossible to crypto-analyze.

4.7 HMAC-SHA-1 and HMAC-SHA-1-96

HMAC-SHA-1-96 is a HMAC that uses SHA-1 for its hash function, and the resulting hash is truncated to 96 bits. Its performance is that of SHA-1.
5. Data Encryption Standard (DES) - DEA/3DEA

DES encrypts the data by converting into unintelligible form. It is also used for decrypting already encrypted data. The algorithm uses 64 bit key, where the 56 bits are the key, and in each byte the eight bit is used for parity. This key is used for encipherment, as well as decipherment. The algorithm works on block of 64 bits. Each block is subject to Initial Permutation \( IP \), then to a key dependent computation, and finally to a reverse permutation \( IP^{-1} \). Each key dependent computation can be defined as cipher function \( f \), and a function \( KS \), called key schedule.

5.1 Enciphering.

The figure below represents the steps needed to encipher a block of 64 bits. Given two blocks \( L \) and \( R \), \( LR \) represents the results of concatenating the two blocks. Note, that is an associative operation, i.e. \( B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 \) represents a block consisting of the bits of \( B_1 \), then followed by the bits of \( B_2 \), then \( B_3 \) and so on, followed by \( B_8 \).

The encryption algorithm is illustrated in the figure below [FIPS 46-3] 21:

\[\text{Figure: Data Encryption Algorithm} \]

---


In 1972 National Bureau of Standards (NBS), now the National Institute of Standards and Technology (NIST), initiated a program to protect computer and communications data. In the May 15, 1973 Federal Register they issued a public request for proposals for standards for cryptographic algorithm. The responses indicated considerable interest from the public, but they also indicated little knowledge and expertise in the field, and none of the submissions came close to the requirements.

In August 27, 1974 Federal Register NBS issued a second request for proposals for standard. This time NBS received a promising candidate – an algorithm based on one IBM developed in the early 1970s. The algorithm was complicated, but straightforward as it used simple logical operations on small groups of bits.

The NBS requested NSA’s help in evaluating the algorithm and determining its suitability as a federal standard. IBM had already filed for a patent, but NBS eventually received nonexclusive, royalty-free license to make, use, and sell equipment that uses the algorithm.

In March 17, 1975 Federal Register published details of the algorithm, and IBM’s statement granting nonexclusive, royalty-free license, and requested comments. In August 1, 1975 Federal Register appeared another notice requesting comments from the public and agencies. There were many comments. Many were wary of the NSA’s “invisible hand” and were afraid NSA had installed a trapdoor. They complained that NSA had reduced the key size from 128 bits to 56. In 1976 NBS held two workshops to evaluate the proposed algorithm – one to discuss the mathematics behind the algorithm and the possibility of a trapdoor; and one to discuss the possibility of increasing the key length.

Despite criticism, the Data Encryption Standard was adopted as a federal standard on November 23, 1976, and approved for use for all unclassified communications. The official description of the standard, FIPS PUB 46, “Data Encryption Standard”, was published on January 15, 1977 and became effective six months later. In 1980, FIPS PUB 81, “DES Modes of Operation” was published.

This standard was unprecedented. For a first time NSA-evaluated algorithm was de public. As explained in [APLC], probably this was due misunderstanding between NSA and NBS. NSA thought DES was for hardware only, and the standard mandated hardware implementation, but NBS published enough details, that enabled software implementation of the algorithm. Off the record, NSA has characterized DES as one of their biggest mistakes. DES galvanized the field of cryptanalysis, than did anything else – there was an algorithm to study, which NSA has said to be secure. The next government standard algorithm, Skipjack, was classified.

Today, DES is considered inadequate and, as specified in FIPS 46-3, should be used for legacy systems only. The Triple Data Encryption Algorithm, also known as 3DES, should be the symmetric encryption algorithm of choice.

21 FIPS PUB 46-3, “DATA ENCRYPTION STANDARD”
The initial permutation $IP$ is given by the following matrix [FIPS 46-3]:

\[
\begin{array}{cccccccccc}
58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 \\
60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\
62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 \\
64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \\
57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 \\
59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\
61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 \\
63 & 55 & 47 & 39 & 31 & 23 & 15 & 7
\end{array}
\]
That is the permuted input has bit 58 of the input, followed by bit 50 of the input, followed by bit 42, and so on, and followed by bit 15, and bit 7 of the input as a last bit. Then the permuted input is subjected to a complex key-dependent calculations, the output of which is called pre-output. It is subject to the inverse input permutation $IP^{-1}$. Respectively, $IP^{-1}$ is given by [FIPS 46-3]:

\[
\begin{align*}
40 &\quad 8 &\quad 48 &\quad 16 &\quad 56 &\quad 24 &\quad 64 &\quad 32 \\
39 &\quad 7 &\quad 47 &\quad 15 &\quad 55 &\quad 23 &\quad 63 &\quad 31 \\
38 &\quad 6 &\quad 46 &\quad 14 &\quad 54 &\quad 22 &\quad 62 &\quad 30 \\
37 &\quad 5 &\quad 45 &\quad 13 &\quad 53 &\quad 21 &\quad 61 &\quad 29 \\
36 &\quad 4 &\quad 44 &\quad 12 &\quad 52 &\quad 20 &\quad 60 &\quad 28 \\
35 &\quad 3 &\quad 43 &\quad 11 &\quad 51 &\quad 19 &\quad 59 &\quad 27 \\
34 &\quad 2 &\quad 42 &\quad 10 &\quad 50 &\quad 18 &\quad 58 &\quad 26 \\
33 &\quad 1 &\quad 41 &\quad 9 &\quad 49 &\quad 17 &\quad 57 &\quad 25 \\
\end{align*}
\]

Let look at the computations at each step. If $L$ is 32 bits and $R$ is 32 bits, then the 64 bit input is denotes as $LR$. At each step we calculate:

$L' = R$

$R' = L \oplus f(L, Kn)$,

where $\oplus$ denotes bit by bit addition modulo 2, or exclusive or operation. At each step, a different 48 bit block $Kn$ is chosen from the cipher key. This is denoted as:

$Kn = KS(n, KEY)$,

where the function $KS$ is the key schedule. The $KS$ function is illustrated below [FIPS 46-3]:

![Diagram of the key schedule process](image-url)
In the above figure two permutation matrices $PC_1$ and $PC_2$ are used. The two matrices are defined as follows:

$PC_1$

\[
\begin{array}{cccccccc}
57 & 49 & 41 & 33 & 25 & 17 & 9 \\
1 & 58 & 50 & 42 & 34 & 26 & 18 \\
10 & 2 & 59 & 51 & 43 & 35 & 27 \\
19 & 11 & 3 & 60 & 52 & 44 & 36 \\
63 & 55 & 47 & 39 & 31 & 23 & 15 \\
7 & 62 & 54 & 46 & 38 & 30 & 22 \\
14 & 6 & 61 & 53 & 45 & 37 & 29 \\
21 & 13 & 5 & 28 & 20 & 12 & 4 \\
\end{array}
\]

Note, that the this matrix is split into two parts. The first part will produce $C_0$ and the second part $- D_0$. That is, $C_0$ will be bit 57 of KEY, next is 49, next is 41 and so on, and the last bits of $C_0$ are 44 and 36 f the KEY, and $D_0$ being bits 63, 55, 47, and so on, with the bits being 12 and 4 of the KEY.

$PC_2$

\[
\begin{array}{cccccccc}
14 & 17 & 11 & 24 & 1 & 5 \\
3 & 28 & 15 & 6 & 21 & 10 \\
23 & 19 & 12 & 4 & 26 & 8 \\
16 & 7 & 27 & 20 & 13 & 2 \\
41 & 52 & 31 & 37 & 47 & 55 \\
30 & 40 & 51 & 45 & 33 & 48 \\
44 & 49 & 39 & 56 & 34 & 53 \\
46 & 42 & 50 & 36 & 29 & 32 \\
\end{array}
\]

Shift left is defined as rotation left of the bits, and for each iteration the number of shift is defined as follows:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Number of Shifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
</tbody>
</table>
Let’s look at a figure illustrating the function $f(R, K)$.

In the above figure, $E$ denotes a function, that takes 32 bits as its input and produces 48 bit output value. As per [FIPS 46-3] $E$ is defined as follows:

<table>
<thead>
<tr>
<th>32</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

Also, each of the $S_0$, $S_1$, $S_2$, ..., $S_7$ functions takes 6 bits as an input, and produces 4 bit value. $S_0$ is illustrated in the table below:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

So, if $S_0(B)$ is a function that takes 6 bit value $B$ as input, and return 4 bit value it is calculated from the above table as follows: bits 0 and 5 from B form number k between 0 and 3; bits 1, 2, 3, and 4 form a number l between 0 and 15; then the value of $S_0(B)$ is the number in the k-row and l-th column. For example, if is 100110, then k is 10, or 2, 1 is 0011, or 3, then $S_0(B)$ is 14. The rest of the functions is defined as follows:
<table>
<thead>
<tr>
<th>(S_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
Then the input to P is defined as:

\[ S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 \]

The permutation function P is defined as follows:

\[
\begin{array}{cccccccccccccccc}
16 & 7 & 20 & 21 \\
29 & 12 & 28 & 17 \\
1 & 15 & 23 & 26 \\
5 & 18 & 31 & 10 \\
2 & 8 & 24 & 14 \\
32 & 27 & 3 & 9 \\
19 & 13 & 30 & 6 \\
22 & 11 & 4 & 25 \\
\end{array}
\]

Thus the output of P(L) is obtained from L by taking the 16\(^{th}\) bit of L as first, 7\(^{th}\) bit as second, and so on, until 25\(^{th}\) bit is taken as 32\(^{nd}\) bit.

### 5.3 Deciphering.

The same algorithm is used for deciphering. Mathematically, this is expressed as:

\[
\begin{align*}
R_{n-1} &= L_n \\
L_{n-1} &= R_n(+)f(L_n, K_n)
\end{align*}
\]

We only have to ensure that at each step, the same Kn as enciphering is being used.

### 5.4 Modes of Operation – CBC Mode.

DES has four modes of operation - the Electronic Codebook (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback (CFB) mode, the Output Feedback (OFB) mode, that are described in FIPS 81. We are going to look at CBC mode and its application as MAC.

When CBC mode is used for encryption, the first plaintext block is XORed with pseudo-random 64 bit block IV. The resulting encrypted block is used to XOR the second 64 bit block of plain data, and so on.

To decrypt a message, the first 64 bit block of cipher text is decrypted and then XORed with IV. Second block is decrypted by first XORing it with the first cipher text block, and then processing it through the algorithm, and so on.

When used as MAC, the intermediate cipher texts are discarded, and the MAC of a message is given by the last 64 bit block of cipher text. If the length of the message is such, that the last is
5.5 Triple DES (3DES).

As described in ANSI X9.52, “Triple Data Encryption Algorithm Modes of Operation”, 3DES has seven modes of operation – the TDEA Electronic Cookbook (TECB), the TDEA Cipher Block Chaining (TCBC), the TDEA Cipher Block Chaining - Interleaved (TCBC-I), the TDEA Cipher Feedback Mode (TCFB), the TDEA Cipher Feedback Mode - Pipelined (TCFB-P), the TDEA Output Feedback Mode (TOFB), the TDEA Output Feedback – Interleaved (TOFB-I). TECB, TCBC, TCFB and TOFB are based on ECB, CBC, CFB and OFB respectively and are obtained by replacing the encryption/decryption operation with 3DES encryption/decryption operation.

Let $E_{K_1}$ denotes DES encryption operation using key $K_1$, and $D_{K_1}$ denotes DES decryption operation using key $K_1$. Given set of keys $K_1, K_2, K_3$, [ANSI X9.52] defines the 3DES encryption/decryption operation as follows:

a) 3DES encryption of 64 bit plain text block $I$ to 64 bit cipher text $O$ as:

$$O = E_{K_3}(D_{K_2}(E_{K_1}(I)))$$

b) 3DES decryption of 64 bit cipher text block $O$ to 64 bit plain text $O$ as:

$$O = D_{K_1}(E_{K_2}(E_{K_3}(I)))$$

In the above scheme, if $K_1 = K_2 = K_3$ and 3DES is used TECB, TCBC, TCFB, or TOFB modes it is equivalent to DES.

As described in [RFC2410], NULL does nothing to alter the plain text. It is a convenient way of implementing the option of not applying encryption and authentication. Mathematically, NULL is defined using the identity function I:

\[ \text{NULL}(B) = I(B) = B \]

The NULL authentication algorithm needs to be considered in the context of IPsec Encapsulating Security Payload to provide authentication and integrity without confidentiality. ESP allows an optional authentication algorithm to provide message authentication and integrity. ESP_NULL does not include the IP header in calculating the authentication data. As far as IPsec Authentication Header, RFC2402 specification provides a similar service, by computing authentication over the data portion of a packet as well as the immutable in transit portions of the IP header. This can be useful in providing IPsec services through non-IP network devices, or when implementing IPsec solutions with boosting techniques in satellite and WAN wireless environment, where usually some form of transformation of the TCP or IP header occurs.

The NULL encryption algorithm is a convenient way to implement ESP without applying encryption and it is referred to as ESP_NULL. It is simply a convenient way to represent the optional use of applying encryption within ESP. ESP can then be used to provide authentication and integrity without confidentiality. In the same time, RFC 2406 allows an optional encryption algorithm to provide confidentiality for ESP. Unlike AH these services are not applied to any part of the IP header, which does not make ESP_NULL less secure than AH.

In conclusion
But the evolution of encryption standards hasn’t stopped with 3DES. The latest encryption standard is the Advanced Encryption Standard (AES) (FIPS Pub 197), which was approved in November 2001 by National Institute of Standard and technology (NIST). This standard specifies Rijndael as a FIPS-approved symmetric encryption algorithm that may be used by U.S. Government organizations (and others) to protect sensitive information to organizations, institutions, and individuals outside the US government. AES offers greater flexibility than DES, and even 3DES because it allows for multiple key sizes and encoding passes.

The AES is being developed to replace DES, but National Institute of Standard and technology (NIST) anticipates that Triple DES will remain an approved algorithm (for U.S. Government use) for the foreseeable future. Single DES is being phased out of use, and is currently permitted in legacy systems only.

---

22 RFC 2410. “The NULL Encryption Algorithm and Its Use With IPsec”
23 Refers to a well-known issue when IPsec VPNs are experiencing degradation of the throughput over wireless and satellite connections.
24 http://csrc.nist.gov/encryption/aes/