



SAPIENZA
UNIVERSITÀ DI ROMA

IPSec e le sue vulnerabilità: passato, presente e futuro

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di laurea in Informatica

Gianluca Amato
1691171

Relatore
Francesco Parisi-Presicce

A/A 2020/2021

Sommario

L'obiettivo di questa tesi è volto allo studio e all'approfondimento de la suite di protocolli IP security, il quale ha il fine di garantire confidenzialità, autenticità ed integrità.

La semplicità con cui la mole dei dati ad oggi viene condivisa, ha permesso, nel corso della storia di Internet, la nascita e lo sviluppo di vulnerabilità, di attacchi malevoli, e di gestione di dati sensibili da parte di malintenzionati, per tale motivo l'attenzione verrà focalizzata sui diversi protocolli Authentication Header, Encapsulating Security Payload e Internet Key Exchange i quali definiscono la modalità di cifratura del traffico dei dati e dell'interscambio delle informazioni attraverso i canali di comunicazione all'interno della rete stessa ma soprattutto verso l'esterno.

Sono state comunque identificate delle lacune di sicurezza nella suite IP security, verranno dunque analizzate le vulnerabilità degli algoritmi utilizzati per la cifratura, autenticazione e integrità in IP security. Tali vulnerabilità permettono a terze parti di catturare comunicazioni criptate, impersonare dispositivi legittimi, rubare dati per scopi malevoli e ottenere l'accesso ad un sistema senza i dovuti privilegi. Sarà descritto anche il funzionamento di ognuno di essi.

Si enunciano infine gli attacchi verso IP security e le implementazione future. Questa relazione è dunque strutturata in 4 parti: cosa è IP security, come nasce e perché, studio dei protocolli, approfondimenti e in dettaglio la differenza tra di essi; trattamento delle vulnerabilità dei diversi protocolli e gli attacchi verso IP security sfruttando queste; attacchi verso IP security ma indipendentemente dai protocolli e dagli algoritmi utilizzati; ultimo capitolo è indirizzato per le implementazioni/ricerche future.

Indice

Indice	v
1 Introduzione	1
1.1 IPsec	1
1.2 Casi d'uso di IPsec	4
1.3 I protocolli di IPsec	6
1.3.1 Authentication Header	6
1.3.2 Encapsulating Security Payload	9
1.3.3 Internet Key Exchange	11
2 Vulnerabilità di IPsec	17
2.1 Vulnerabilità IKE	17
2.2 Vulnerabilità ESP	21
2.3 Vulnerabilità algoritmi usati in IPsec	26
2.3.1 Data Encryption Standard	26
2.3.2 3DES	28
2.3.3 Blowfish	30
2.3.4 Md5	31
2.3.5 Sha1	33
3 Attacchi verso IPsec	35
3.1 Attacco Man-in-the-Middle in IPsec	35
3.2 Attacco Replay	37
3.3 Attacco LogJam	38
3.4 Attacco Denial of Service in IPsec	40
3.5 Attacco Buffer Overflow	40
4 Conclusioni	43
4.1 Implementazioni future	44
Bibliografia	47
A Creazione tunnel VPN	49
B Decifratura Md5	51
C Grafici dei tempi di cifratura e decifratura	53
D IPsec in breve	55

1. Introduzione

La nascita e lo sviluppo delle reti informatiche, ha permesso agli utenti di computer differenti di comunicare tra di loro. Negli anni l'infrastruttura di Internet si è espansa in tutto il mondo creando una rete globale; ciò ha inciso progressivamente l'orientamento dell'uomo verso la rete, dove si traggono vantaggi da diversi punti di vista ad esempio il mondo della comunicazione è cambiato radicalmente, si possono scambiare informazioni e rimanere in contatto con tutto il mondo, permette inoltre l'accesso alle informazioni da qualsiasi punto del pianeta; basti pensare all'evoluzione dell'Internet Banking che ad oggi consente di effettuare operazioni bancarie da remoto mediante collegamento telematico, ma dall'altro lato si riscontrano problematiche legate alla sicurezza delle informazioni, proprio come da ultimo esempio; nasce per cui la necessità di proteggere i propri dati. Allo scopo di garantire sicurezza sono stati progettati, nel corso degli anni, diversi protocolli di sicurezza, come *IP Security* (IPsec).

1.1 IPsec

La sua nascita si deve ad una spinta della *Defence Advanced Research Projects Agency* (DARPA) e la *National Security Agency* (NSA). Il progetto Arpanet nato alla fine degli anni '60 era una rete di computer realizzata per mettere in collegamento strutture militari, ma secondo l'opinione di DARPA era necessario proteggere le connessioni, così che iniziò a sponsorizzare una serie di dispositivi per la crittografia delle connessioni tra le strutture militari. Successivamente il ruolo della NSA fu quello di promuovere, nella seconda metà degli anni '80, lo sviluppo di una serie di protocolli per la crittografia dei dati sulla neo-Internet, si trattava del programma *Secure Data Network Systems* (SDNS). Lo studio continuò ulteriormente e già dagli inizi degli anni '90 il primo nucleo di protocolli della suite IPsec era disponibile.

I pacchetti IP, gli elementi alla base dello scambio dati su Internet, sono composti da due parti: l'**area dati** tra cui rientra la lingua, il testo o le immagini e l'**header**, composto da 13 campi, di cui fanno parte gli indirizzi del mittente e del destinatario. I dati che viaggiano attraverso i diversi router per arrivare ai destinatari, non sono cifrati e possono essere pertanto, letti o manipolati da terze parti in qualunque momento, poiché il protocollo di rete in sé non possiede meccanismi di crittografia e autenticazione, dunque non sono garantiti **confidenzialità**, **autenticità**, e **integrità**.

Per questo motivo nasce *IP Security* (IPsec). IPsec è stato progettato per rendere sicure sia le comunicazioni *portal-to-portal* che *end-to-end* sia lo scambio di informazioni attraverso Internet, tuttavia l'uso predominante è quello per la creazione di *Virtual Private Network* (VPN)¹.

¹Per un'implementazione pratica di una VPN tramite IPsec si veda Appendice A.

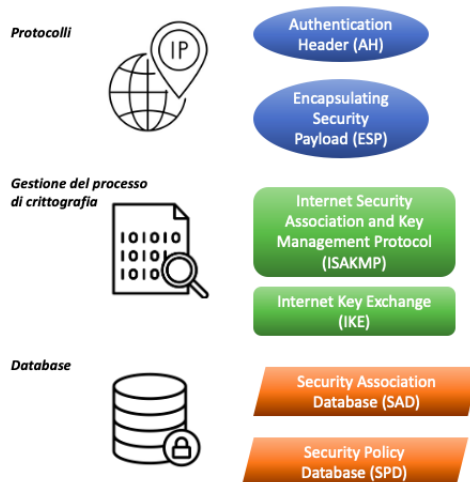


Figura 1.1: Struttura generale di IPsec

IPsec è un insieme di protocolli che garantisce sicurezza a livello IP ed ai livelli superiori, inizialmente fu sviluppato all'interno dello standard IPv6 e successivamente inserito in IPv4. La sua architettura e implementazione sono definite negli RFC² dal 2401 al 2412 ed in seguito modificate negli RFC dal 4301 al 4309. Nella Figura 1.1, troviamo la suddivisione di IPsec in base alle funzioni svolte.

IPsec può proteggere l'intero datagramma IP o solamente i protocolli di alto livello e può lavorare in due diverse modalità: **tunnel mode** e **transport mode**. Nel tunnel mode, l'intero pacchetto IP originale viene incapsulato per diventare il payload di un nuovo pacchetto e viene aggiunto un nuovo header IP. Nel transport mode, il traffico criptato viene inviato direttamente ai due host che hanno stabilito precedentemente un tunnel sicuro. Le principali differenze tra la modalità di trasporto e la modalità tunnel sono le seguenti:

- ✓ La modalità tunnel è più sicura perché i pacchetti IP originali sono completamente autenticati e crittografati. Questa modalità nasconde l'indirizzo IP, il tipo di protocollo e il numero di porta in un pacchetto IP originale.
- ✓ La modalità tunnel genera un'intestazione IP aggiuntiva, occupando più larghezza di banda rispetto alla modalità di trasporto.
- ✓ La modalità di trasporto viene utilizzata principalmente per la comunicazione tra due host o tra un host e un gateway VPN. La modalità tunnel viene utilizzata principalmente per la comunicazione tra due gateway VPN o tra un host e un gateway VPN.

La differenza più significativa tra le due modalità è che nella prima, il pacchetto trasportato da IPsec deve contenere sia l'indicazione del gateway sia quella del destinatario finale, mentre nella seconda quest'ultima non è necessaria.

IPsec può essere implementato in un host oppure in congiunzione con un router o firewall; si possono utilizzare tre approcci per l'implementazione:

- **Integrated Implementation:** Integra IPsec nell'implementazione dell'IP. Questo approccio è il migliore, ma anche difficile da implementare poiché richiede di riscrivere l'implementazione dell'IP per includere IPsec; questo approccio fornisce sicurezza nativamente ma richiede che l'intero stack venga aggiornato quando si effettua un cambiamento (Figura 1.2(a)).
- **“Bump-in-the-stack” (BITS):** Implementa IPsec “sotto” lo stack IP e sopra i driver di rete locale. L'implementazione di IPsec monitora il traffico IP in quanto viene inviato o ricevuto tramite il collegamento locale e le funzioni IPsec vengono eseguite sui pacchetti prima di passarli in alto o in basso nello stack. Questo approccio inserisce un codice IPsec speciale nello stack di rete appena sotto il software di rete esistente e sopra il software di collegamento locale. In altre parole, questo approccio implementa la sicurezza attraverso un software che, intercetta i datagrammi passati dallo stack IP esistente al livello di collegamento dell'interfaccia locale (Figura 1.2(b)).
- **“Bump-in-the-wire” (BITW):** Implementa IPsec in un processore hardware crittografico, il quale ha un proprio indirizzo IP. Questo approccio usa hardware per fornire sicurezza; il dispositivo funge da router o da gateway per tutti i datagrammi IP dietro di esso. Quando il dispositivo è usato per un singolo host, questo approccio funziona molto bene come BITS, ma la sua implementazione diventa più complicata quando si ha a che fare con più host.

²I documenti *Request For Comments* (RFC) contengono specifiche tecniche e note organizzative per Internet e sono prodotti dall'*Internet Engineering Task Force* (IETF) [1].

Per applicazioni che richiedono un alto livello di sicurezza questo approccio è quello più indicato (Figura 1.2(c)).

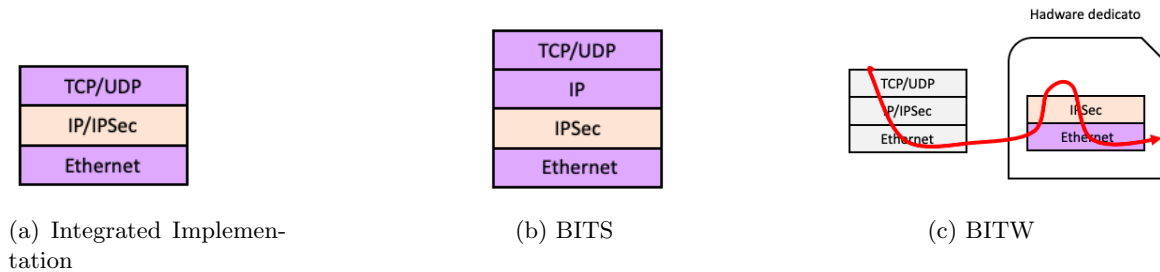


Figura 1.2: Implementazioni IPsec

> Security Association

Una *Security Association* (SA) è una negoziazione di parametri tra due *peer* comunicanti; può includere le preferenze per l'autenticazione, la cifratura ed i protocolli di IPsec che si intendono utilizzare. Ogni peer coinvolto in una comunicazione tramite IPsec deve settare due database: un *SA Database* (SAD), per gestire le proprie SA che può contenere informazioni su quali algoritmi sono utilizzati per la comunicazione, quale host ha il permesso di comunicare e su quale porta; un *Security Policy Database* (SPD), il quale mette in relazione una porzione del traffico IP ad una SA. Quindi una SA è una connessione logica tra due host, identificata dai seguenti parametri:

- Security Parameter Index (SPI)
- IP Destination Address
- Security Protocol Identifier (AH o ESP, vedi in seguito)

Nelle Figure 1.3(a) e 1.3(b) vediamo un'implementazione grafica di come funzionano i database sopra descritti.

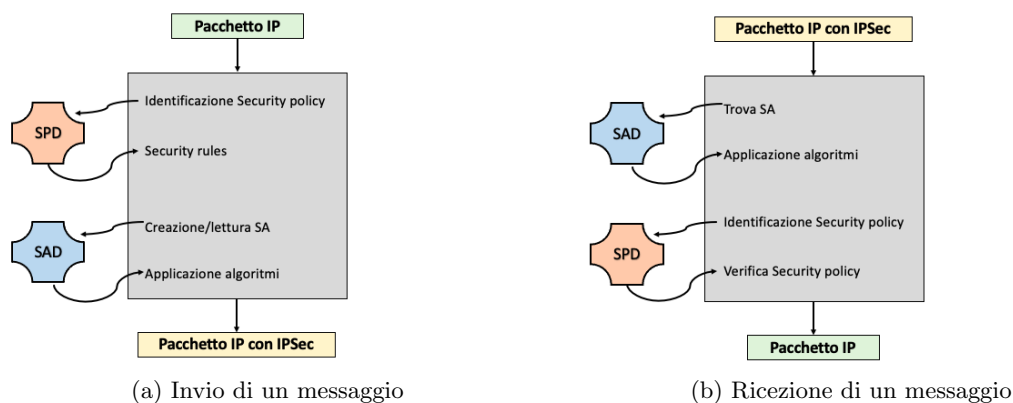


Figura 1.3: Security Association

Si può configurare IPsec con una SA predefinita, con una già condivisa in precedenza (pre-shared) oppure utilizzare IKE (vedi paragrafo 1.3.3) per stabilirne una dinamica. Quando uno tra i due peer di una SA è un *gateway*, allora la SA dovrà usare la modalità tunnel; al contrario quando i due peer sono due host la SA può usare sia la modalità tunnel che quella di transport.

1.2 Casi d'uso di IPsec

IPsec è principalmente associato alle connessioni VPN che si basano su una rete pubblica come Internet, fornendo sicurezza sotto vari aspetti. Di seguito vengono descritti casi d'uso pratici dove è possibile implementare IPsec.

➤ Accesso remoto ad infrastruttura sulla rete

I dati scambiati tra un dispositivo ed una rete interna non accessibile pubblicamente, devono essere protetti; uno dei casi d'uso di IPsec è proprio questo. IPsec è utilizzato per connettere un dispositivo (per esempio un computer) con una rete interna (per esempio di un'azienda), fornendo confidenzialità, autenticità e integrità ai dati scambiati; il tunnel è installato tra il dispositivo remoto e i dispositivi di rete, con questa configurazione il tunnel viene chiamato *Host-to-Site* (Figura 1.4) e permette ad un dispositivo di accedere alla rete in sicurezza. Il dispositivo remoto una volta connesso ad Internet, usa il client IPsec per connettersi alla rete tramite il tunnel.

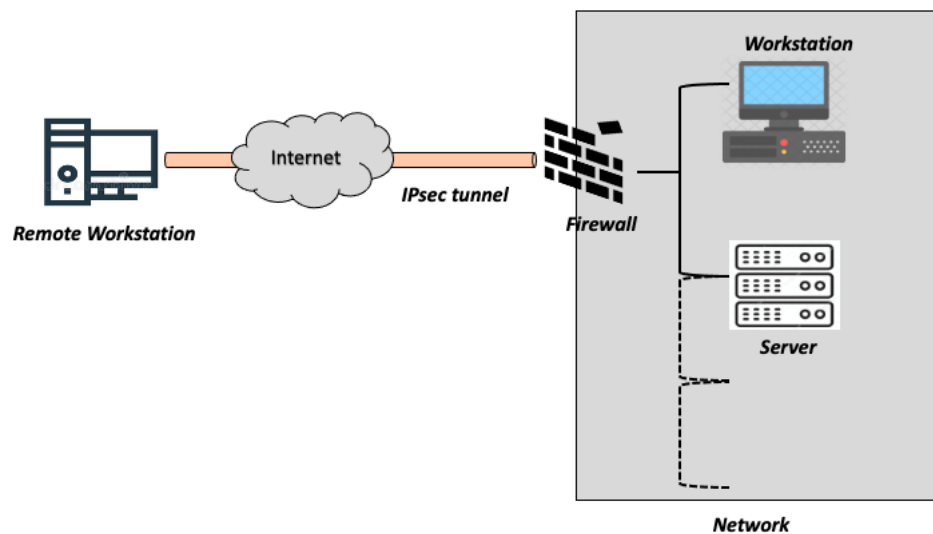


Figura 1.4: Configurazione Host-to-Site

➤ Connessione tra due reti

IPsec inoltre, può instaurare una connessione tra due o più reti locali (per esempio due sedi di un'azienda), questa configurazione è chiamata *Site-to-Site* (Figura 1.5). È usata prevalentemente quando la connessione si basa su una rete pubblica, come Internet; il

tunnel è installato tra due gateway IPsec o due firewall al confine con le due reti. Il gateway della rete remota negozia, una connessione con il gateway della rete principale; il gateway della rete principale è responsabile dell'autenticazione del gateway remoto, della cifratura e dell'integrità dei dati.

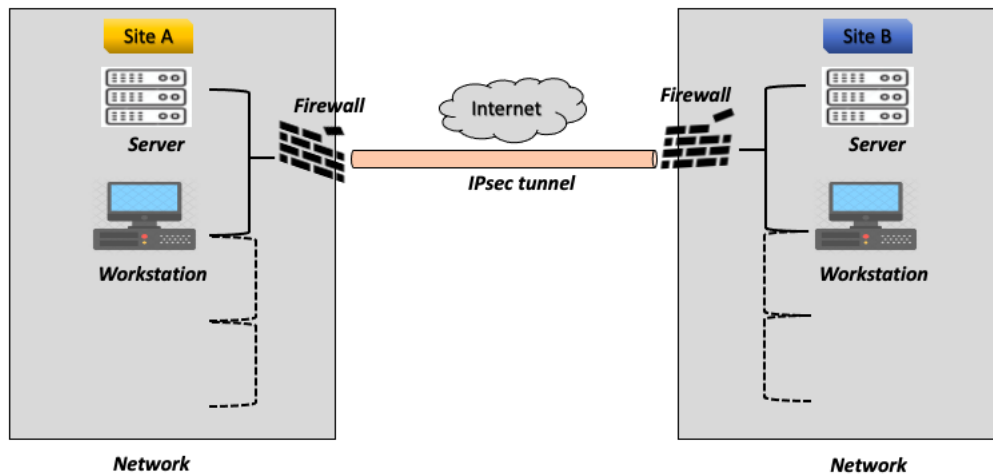


Figura 1.5: Configurazione Site-to-Site

➤ Connessione tra due dispositivi

Un altro caso d'uso di IPsec è quello di connettere due dispositivi per fornire una comunicazione sicura tra essi, questa è chiamata *Host-to-Host* (Figura 1.6). Tuttavia ci sono altri modi più performanti e sicuri di questa tipologia di tunnel. Questa configurazione sfrutta la connessione di ogni host per creare un tunnel sicuro; l'unico requisito che ogni host deve possedere per stabilire il tunnel è la sola connessione ad Internet (oltre al software per IPsec). Prima della trasmissione dei dati, gli host si autenticano con le chiavi scambiate e successivamente inizia la trasmissione.

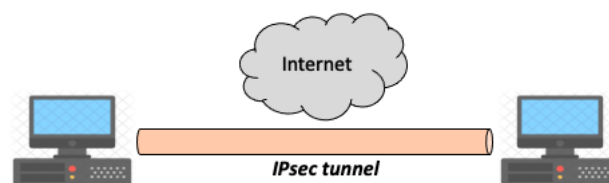


Figura 1.6: Configurazione Host-to-Host

1.3 I protocolli di IPsec

Di seguito i principali protocolli di IPsec che verranno trattati in dettaglio nei paragrafi successivi:

- Authentication Header
- Encapsulating Security Payload
- Internet Key Exchange

1.3.1 Authentication Header

L'*Authentication Header* (AH) implementato in RFC 2402 [2] e modificato successivamente nell'RFC 4302, è un meccanismo di sicurezza utilizzato per autenticare l'origine dei datagrammi e per garantire l'integrità delle informazioni inviate. Esso verifica che il payload e l'header di un pacchetto non siano stati alterati durante la trasmissione, inoltre fornisce protezione dal *replay attack*, azione volta ad ottenere una credenziale di autenticazione comunicata da un host ad un altro, e riproporla successivamente simulando l'identità del mittente. Ciò nonostante, AH non offre la confidenzialità dei dati, per cui i dati sono trasmessi in chiaro.

AH è posizionato tra l'header di un datagramma e l'header del protocollo di trasporto per assicurare autenticazione e integrità; quest'ultima stabilita mediante un messaggio digest³ generato da un algoritmo come **HMAC-MD5** o **HMAC-SHA**. L'autenticazione è garantita usando una chiave segreta condivisa per creare il messaggio digest e infine per la protezione da *reply attack* utilizza un campo con numeri sequenziali nell'header. L'Authentication Header lavora con due modalità di trasmissione: transport mode o tunnel mode.

> Transport mode

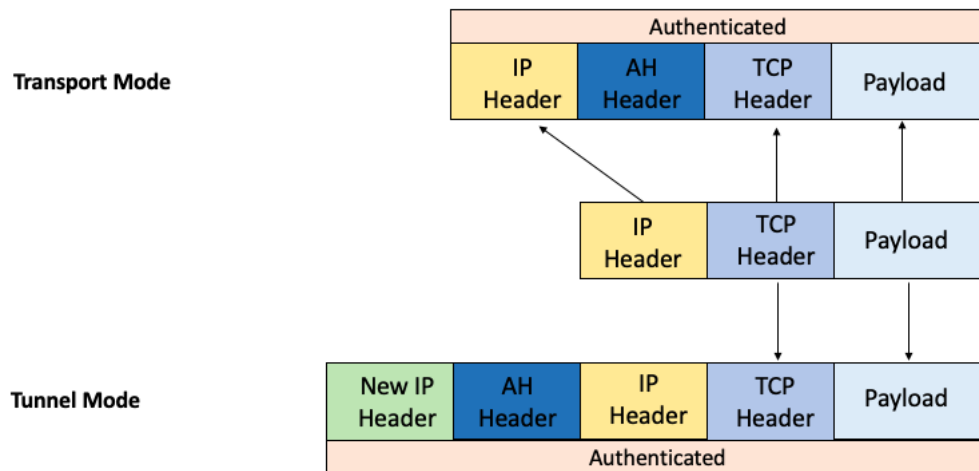
L'header del datagramma originale è l'header IP più esterno, seguito dall'header AH ed infine dal payload del datagramma. In questa modalità l'intero datagramma viene autenticato, eccetto i campi mutabili; tuttavia le informazioni sono trasmesse in chiaro ed è, quindi, soggetto ad intercettazione (Figura 1.7).

> Tunnel mode

Al contrario la modalità tunnel crea un nuovo header e lo usa come header più esterno del datagramma seguito dall'header AH, ed infine appaiono l'header e il payload originali (Figura 1.7). In questa modalità viene autenticato l'intero datagramma compreso il nuovo header e dà la possibilità di rilevare se questo è stato modificato durante la trasmissione del pacchetto.

³Un messaggio digest viene creato mediante una funzione hash generato in modo random per il quale non deve essere possibile trovare due messaggi che abbiano lo stesso digest e per il quale non deve essere possibile risalire al messaggio originale dato il digest.

Figura 1.7: AH Transport e Tunnel mode



Di seguito vengono riportati i campi di cui è composto il protocollo AH:

Campo	Lunghezza	Descrizione
Next header	8 bit	Identifica il tipo di payload che segue AH. In modalità di trasporto è il numero del protocollo di livello superiore. In modalità tunnel è il numero del protocollo IP o ESP.
Payload length	8 bit	Il valore di questo campo è la lunghezza del pacchetto AH in parole di 32 bit meno 2. Il valore di default è 4.
Reserved	16 bit	Campo riservato, il valore di default è 0.
Security Parameter Index	32 bit	Insieme all'indirizzo di destinazione e all'identificatore del protocollo identifica la Security Association per la trasmissione dei dati.
Sequence number	32 bit	È un contatore, si incrementa di un'unità per ogni datagramma inviato da un host ad una Security Association. Questo campo fornisce sicurezza contro i <i>replay attack</i> .
Authentication data	Multiplo di 32 bit, 96 bit di default	Contiene l'Identity Check Value (ICV) ed è usato dal destinatario per verificare l'integrità dei dati calcolando un valore hash e confrontandolo con il numero ICV calcolato dal mittente.

In molti casi si richiede solo l'autenticazione dei dati e ciò avviene senza influenzare le prestazioni del sistema; inoltre al contrario di altri protocolli (vedi ESP nel paragrafo successivo) non ha bisogno di algoritmi di crittografia complessi per il suo corretto funzionamento.

L'Authentication Header può essere utilizzato sia da solo sia insieme al protocollo Encapsulating Security Payload.

1.3.2 Encapsulating Security Payload

L'Encapsulating Security Payload (ESP), implementato in RFC 2406 [3] e modificato successivamente nell'RFC 4303, è progettato principalmente per fornire servizi di crittografia, autenticazione e protezione per i dati o il payload che viene trasferito in una rete IP, ma può anche garantire solo confidenzialità o confidenzialità e autenticazione; si evidenzia che l'uso della crittografia senza autenticazione è fortemente sconsigliato poichè non sicuro. In ESP entrambi i sistemi comunicanti usano una chiave condivisa per la crittazione e decrittazione dei dati scambiati. Se si vuole ottenere sia la confidenzialità sia l'autenticazione, il sistema ricevente, come prima cosa autentica il pacchetto, se l'autenticazione ha dato esito positivo, allora procede con la decrittazione. Questo tipo di configurazione riduce l'overhead di elaborazione, oltre a ridurre il rischio di attacchi *Denial-Of-Service*. Anche ESP può operare sia in transport mode che in tunnel mode.

> Transport mode

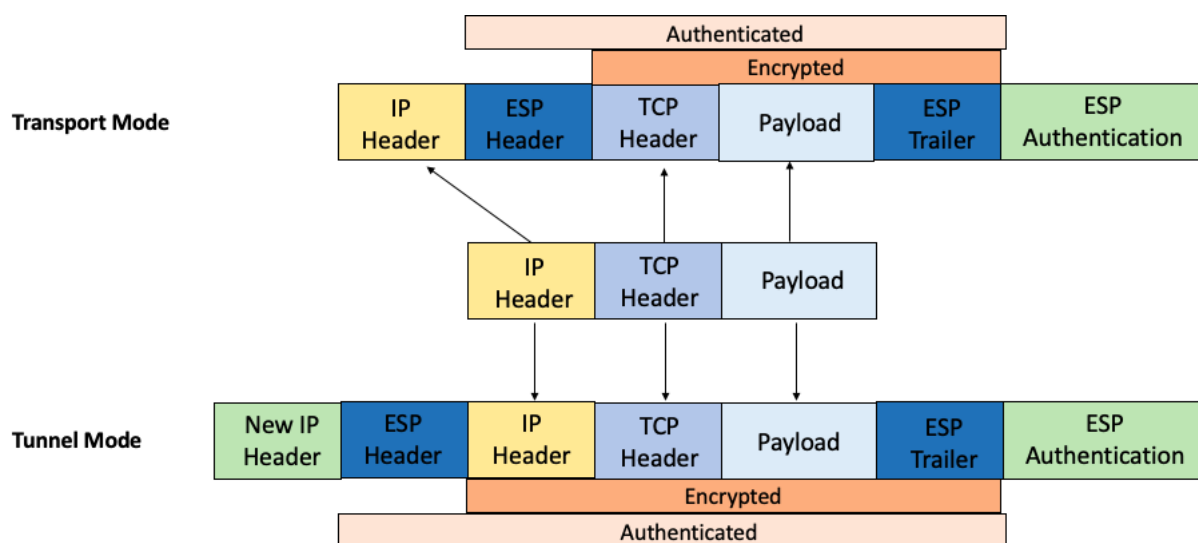
È un metodo di invio di dati su Internet in cui i dati sono crittografati ma le informazioni sull'indirizzo IP originale non lo sono. L'header ESP segue l'header IP del datagramma originale, se il datagramma originale contiene già un IPsec header allora l'header ESP va prima di questo ed infine si troveranno i dati opzionali. ESP in modalità di trasporto non fornisce integrità e autenticazione per l'intero pacchetto IP, il che potrebbe esporre le informazioni ad un potenziale attacco (Figura 1.8).

> Tunnel mode

Diversamente dal transport mode, le informazioni sull'indirizzo IP originale sono crittografate. Se l'intero pacchetto è incapsulato all'interno di un altro pacchetto come payload di dati, può crittografare l'intero pacchetto. Viene creato un nuovo IP header che viene usato come header più esterno, seguito dall'header ESP ed infine dal datagramma originale. In questa modalità ESP protegge completamente il datagramma originale ma non protegge il nuovo header IP (Figura 1.8).

In entrambe le modalità viene anche criptato il *trailer*, inserito per adeguare la grandezza del payload alla lunghezza richiesta dal cifrario a blocchi a chiave simmetrica utilizzato.

Figura 1.8: ESP Transport e Tunnel mode



ESP usa diversi algoritmi per la crittografia e autenticazione; li vediamo nell'elenco di seguito con il riferimento agli RFC in cui vengono implementati⁴:

- **Algoritmi di crittografia:**

- Data Encryption Standard (DES) [RFC 1829]
- Triple-DES (3DES) [RFC 1851]
- Advanced Encryption Standard (AES) [RFC 3602, 4106, 4309 e 4543]
- Blowfish
- Camellia [RFC 3713]

- **Algoritmi di autenticazione:**

- HMAC-MD5 [RFC 1321 e 2403]
- HMAC-SHA1 [RFC 3174 e 2404]
- HMAC-SHA2 [RFC 4868]

Di seguito vengono riportati i campi che costituiscono il protocollo ESP:

Campo	Lunghezza	Descrizione
Security Parameter Index	32 bit	Come per AH, identifica unicamente una SA
Sequence number	32 bit	È un contatore, si incrementa di un'unità per ogni datagramma inviato da un host ad una SA. Esso fornisce sicurezza contro i <i>replay attack</i>
Next header	8 bit	Identifica il tipo di payload che segue ESP. In modalità di trasporto è il numero del protocollo di livello superiore. In modalità tunnel è il numero del protocollo IP
Payload data	-	Contiene la lunghezza dei dati identificato dal campo Next Header
Padding	-	Estende la size di un header ESP. La lunghezza dipende dalla lunghezza del payload e dall'algoritmo;
Authentication data	Multiplo di 32 bit, 96 bit di default	Contiene l' <i>Identity Check Value</i> (ICV) ed è usato dal destinatario per verificare l'integrità dei dati calcolando un valore hash e confrontandolo con il numero ICV calcolato dal mittente. Le funzioni di autenticazione di ESP sono opzionali ma se sono necessarie un ICV viene appeso ai dati criptati
Pad length	8 bit	Specifica la lunghezza del campo Padding. Il valore 0 indica che non esiste padding

⁴Per la visualizzazione degli RFC, si rimanda a [4].

Nella Tabella 1.1 vengono riportate le principali differenze tra AH, ESP con solo confidenzialità e ESP con confidenzialità e autenticazione.

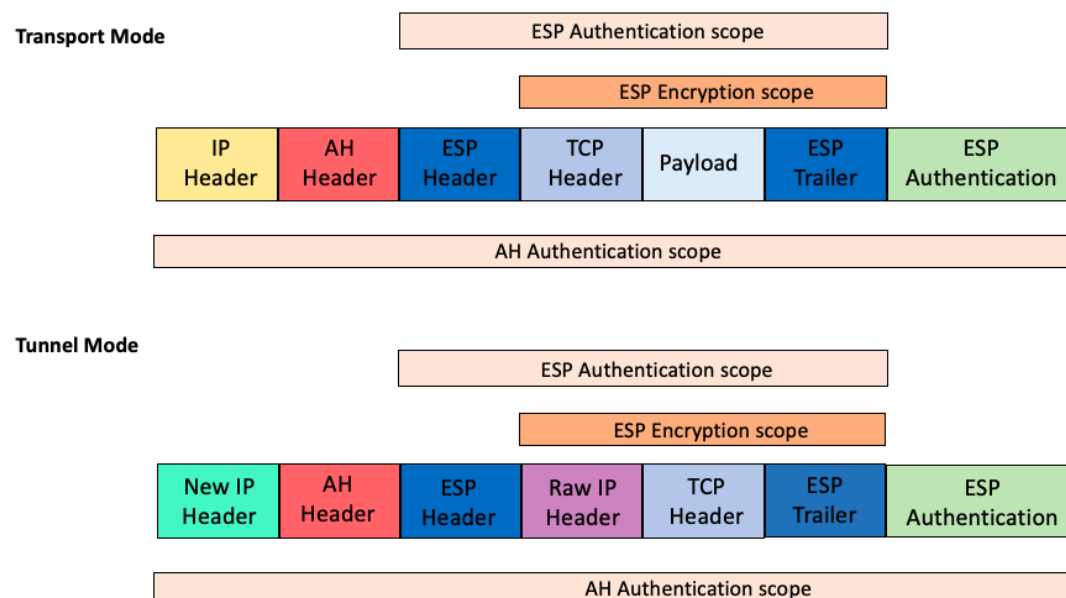
Tabella 1.1: Differenze tra AH e ESP

	AH	ESP (conf.)	ESP (conf. & aut.)
Integrità	✓		✓
Autenticazione	✓		✓
Anti-replay attack	✓	✓	✓
Confidenzialità		✓	✓

Dunque, nei paragrafi precedenti abbiamo visto come transport mode e tunnel mode lavorano con il solo protocollo AH e con il solo protocollo ESP, ma vediamo nella Figura 1.9 come possono lavorare con entrambi, in modo da autenticare anche l'indirizzo di origine contenuto nell'header IP esterno.

Quando vengono utilizzati sia AH che ESP per proteggere il traffico, devono utilizzare la stessa modalità di incapsulamento.

Figura 1.9: AH e ESP Transport e Tunnel mode



1.3.3 Internet Key Exchange

Nei sistemi di crittografia simmetrica, entrambe le parti usano la stessa chiave per la crittazione e decrittazione dei dati, perciò le chiavi devono essere scambiate in un canale sicuro. L'obiettivo dell'*Internet Key Exchange* (IKE), implementato in RFC 2409 [5] e modificato successivamente in RFC 4109, è fornire comunicazioni sicure tra due dispositivi, o *peer*, su una rete; entrambe le

parti producono indipendentemente la stessa chiave simmetrica. IKE inoltre autentica entrambe le parti e le accorda su quale metodo di crittografia e integrità usare. Il risultato di questa negoziazione è una *Security Association* (SA); anche questa deve avvenire in modo sicuro per cui IKE si compone di due fasi. L'algoritmo utilizzato per lo scambio delle chiavi è quello di *Diffie-Hellman*⁵.

➤ Fase 1

Nella fase 1 dello scambio si effettuano tre passi:

1. Negoziazione dei parametri usati per la costituzione della SA IKE
2. Scambio di informazioni relative alla chiave utilizzando l'algoritmo Diffie-Hellman
3. Autenticazione reciproca tra i due *peer*

Questa fase supporta due modalità: *main mode* e *aggressive mode*.

Main mode: effettua 3 passaggi per effettuare le attività precedenti ed ogni passaggio richiede 2 pacchetti ISAKMP pertanto sono necessari in totale 6 pacchetti ISAKMP. Prima di cominciare la negoziazione viene creato un canale sicuro, quindi cifrato, in cui i due *peer* effettuano l'autenticazione⁶. Una volta avvenuta l'autenticazione i due *peer* si scambiano i 6 pacchetti (si veda Figura 1.10):

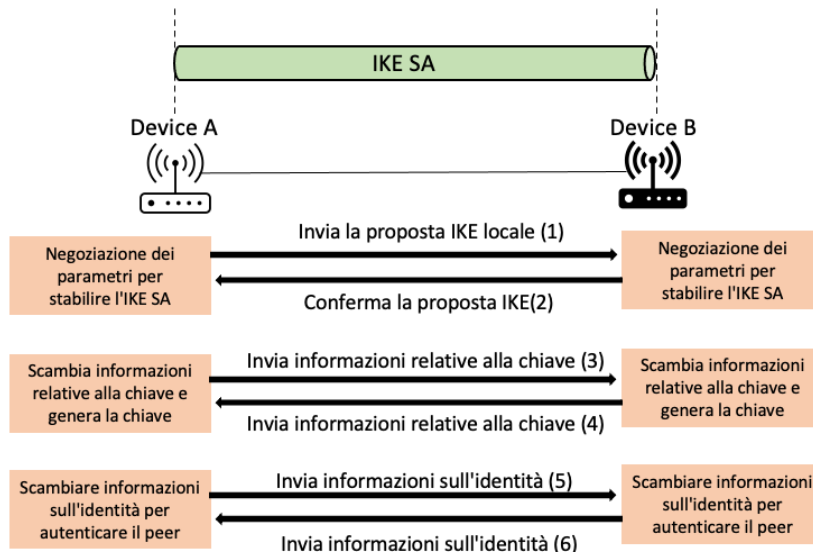
1. Nel primo pacchetto l'iniziatore (colui che ha iniziato la sessione) invia al risponditore un *cookie* e la lista degli algoritmi crittografici per l'autenticazione, l'integrità e per la confidenzialità, questi formano la ISAKMP SA;
2. Il risponditore risponde anch'esso con un proprio *cookie*;
3. L'iniziatore manda i propri parametri di Diffie-Hellman;
4. Il risponditore risponde con i propri parametri di Diffie-Hellman completando la definizione dei parametri della ISAKMP SA e da questo momento possono comunicare scambiandosi dati cifrati. Vengono generate quattro chiavi: *SKEYID*, *SKEYID_a* usata per il controllo dell'integrità, *SKEYID_e* usata per crittografare il messaggio ISAKMP e *SKEYID_d* per la derivazione delle chiavi di crittografia e autenticazione per i pacchetti IPsec. Il calcolo di *SKEYID* varia a seconda della modalità di autenticazione scelta;
5. L'iniziatore manda le proprie credenziali al risponditore;
6. Il risponditore risponde con le proprie credenziali così da assicurare l'identità reciproca. Le informazioni sull'identità sono crittografate utilizzando *SKEYID_e*, garantendo quindi la sicurezza.

Al termine di questa fase i *peer* utilizzano l'algoritmo di crittografia, l'algoritmo di autenticazione e l'autenticazione dell'identità per eseguire crittografia, decrittografia e autenticazione.

⁵L'algoritmo Diffie-Hellman risale al 1976 ed è particolarmente adatto alla generazione di una chiave segreta tra due corrispondenti che comunicano attraverso un canale non sicuro. La sua sicurezza si basa sulla complessità computazionale del calcolo del logaritmo discreto. Per l'implementazione e dettagli si rimanda a [6].

⁶Per effettuare l'autenticazione, IKE supporta due metodi: chiave condivisa in precedenza (PSK) e firma RSA.

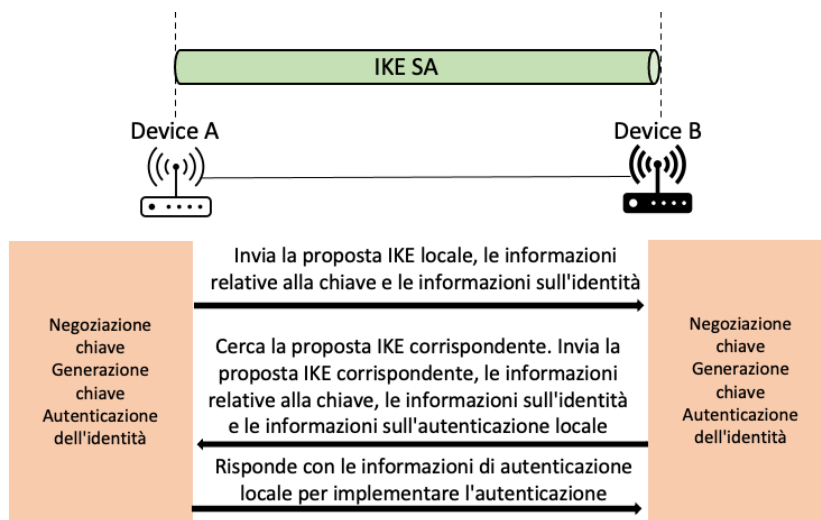
Figura 1.10: IKE Fase 1 in main mode



Aggressive mode: al contrario della main mode richiede solo tre scambi di messaggi (Figura 1.11):

1. L'iniziatore invia un messaggio ISAKMP che riporta i parametri utilizzati per creare una SA IKE, informazioni relative alla generazione di chiavi e le informazioni riguardanti l'autenticazione dell'identità;
2. Il risponditore conferma che il primo pacchetto è stato ricevuto, ricerca e invia i parametri corrispondenti, le informazioni utilizzate per la generazione della chiave e le informazioni riguardanti l'autenticazione dell'identità dell'iniziatore;
3. L'iniziatore risponde con il risultato dell'autenticazione e crea la SA IKE.

Figura 1.11: IKE Fase 1 in aggressive mode



L'aggressive mode non fornisce la protezione dell'identità ed è meno sicuro del main mode ma in termini di velocità è più performante rispetto a quest'ultimo.

Al termine della prima fase si è stabilita la ISAKMP SA, quindi esiste un canale sicuro protetto dagli algoritmi scelti, inoltre i due peer si sono autenticati a vicenda. Utilizzando il canale sicuro ISAKMP i due peer possono ora negoziare la IPsec SA.

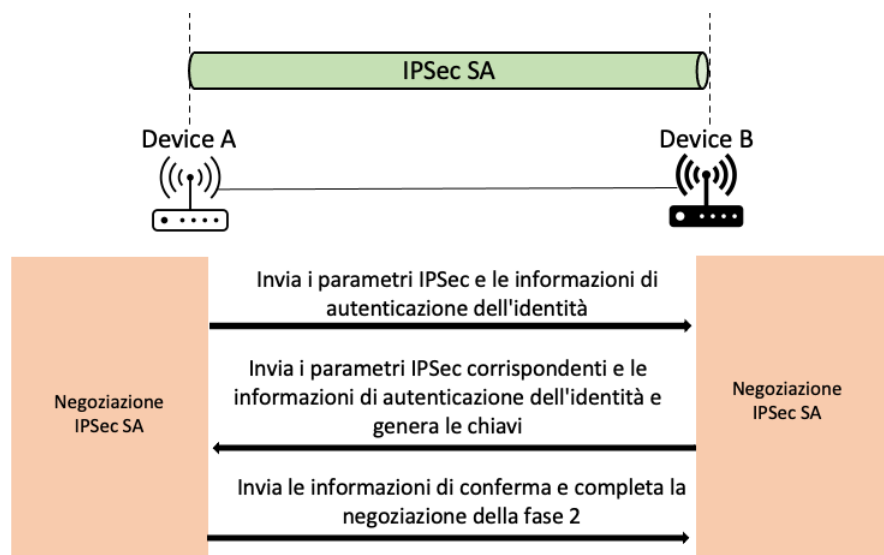
> Fase 2

La fase 2 si svolge con lo scambio di 3 pacchetti (si veda Figura 1.12):

1. Nel primo pacchetto uno dei due peer invia la propria proposta per la IPsec SA incluso il *Security Parameter Index* (SPI) di un tunnel insieme alle informazioni di autenticazione incluse nella chiave generata nella fase 1;
2. Nel secondo pacchetto, l'altro peer, accetta la proposta della IPsec SA inviata dal primo peer ed invia la propria proposta includendo lo SPI del secondo tunnel. In questo passo viene generata una nuova chiave condivisa tra i peer. Le chiavi di crittografia e autenticazione sono derivate da parametri come *SKEYID_d* e SPI assicurando che ogni IPsec SA abbia una chiave univoca;
3. Nel terzo pacchetto il primo peer accetta la proposta fatta dal secondo e concludendo la negoziazione i due peer possono ora comunicare.

Al termine della seconda fase entrambi i peer hanno tutti i parametri necessari per poter attivare i tunnel ed incominciare a scambiare pacchetti. Una volta creato un pacchetto IPsec per un particolare tunnel identificato dallo SPI, dall'IP dell'altro peer e dal protocollo ESP o AH, il pacchetto è spedito all'altro peer. Quando il secondo peer riceve il pacchetto individua la SA dai tre parametri e li utilizza per ottenere il pacchetto originale.

Figura 1.12: IKE Fase 2



IKE è implementato in due versioni: **IKEv1** e **IKEv2**. IKEv2 è il successore di IKEv1 (ormai deprecato, ma ancora ampiamente usato); le differenze tra le due versioni sono illustrate nella Tabella 1.2.

Tabella 1.2: Differenze tra IKEv1 e IKEv2

IKEv1	IKEv2
RFC 2409/4109	RFC 4306
Sviluppato nel 1998	Sviluppato nel 2005 da Cisco e Microsoft
Supporta solo la chiave pre-condivisa e i certificati	Supporta inoltre l'autenticazione <i>Extensible Authentication Protocol</i> (EAP)
Non fornisce supporto integrato per l'attraversamento di NAT	Fornisce supporto integrato per l'attraversamento di NAT (NAT Traversal o NAT-T)
Nella fase 1 ha modalità main e aggressive	Fornisce IKE_SA_INIT nella fase 1
Nella fase 2 ha il quick mode	Fornisce IKE_AUTH nella fase 2
Richiede più larghezza di banda	Richiede meno larghezza di banda
Non ha un meccanismo per la conservazione dello stato	Ha un meccanismo integrato per la conservazione dello stato e può ristabilire automaticamente una connessione interrotta (<i>keepalive-mechanism</i>)
Non ha il supporto MOBIKE	Ha il supporto <i>Mobility and Multihoming</i> (MOBIKE), il che significa che può resistere ai cambiamenti di rete, ottimale per i dispositivi mobili
É meno sicuro	É più sicuro, poiché utilizza chiavi di crittografia per entrambi i lati e supporta un numero maggiore di algoritmi di crittografia

Inoltre in IKEv2, la fase 1 e la fase 2 sono parzialmente *interleaving* e la fase 2 è ridotta allo scambio di due messaggi; in IKEv2, oltre a quelle di IKEv1, è implementata l'autenticazione con EAP⁷ che aggiunge una configurazione di implementazione all'ormai obsoleta IKEv1. Un'altra differenza tra le due versioni è che le SA in IKEv2 possono essere create, modificate e cancellate indipendentemente dalla durata del tunnel VPN; inoltre la seconda versione richiede un numero minore di SA, riducendo la banda richiesta per la creazione del tunnel.

➤ Perfect Forward Secrecy

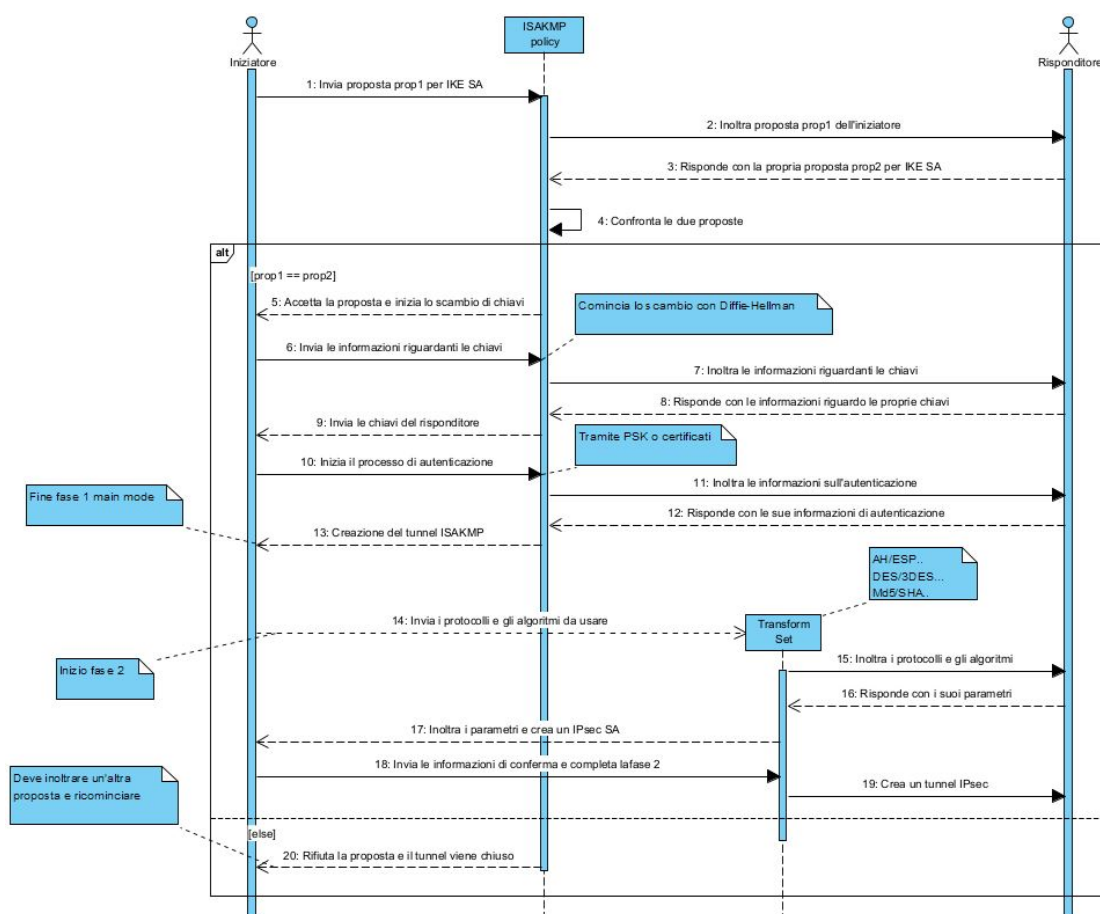
Il *Perfect Forward Secrecy* (PFS) è un meccanismo che garantisce che la stessa chiave non venga generata ed utilizzata più di una volta. Il riutilizzo delle chiavi, espone gli host e il tunnel ad attacchi di vario genere (vedere paragrafo 2.1); con il PFS, se un'attaccante dovesse scoprire la chiave, sarebbe in grado di accedere solo ai dati criptati con quella chiave e non a quelli futuri poiché il PFS forzerà IPsec a generare e utilizzare nuove chiavi.

⁷ *Extensible Authentication Protocol* (EAP) è un framework di autenticazione utilizzato spesso sugli *access point*; esso redirige la richiesta di autenticazione di un client ad uno specifico server, configurato per questo scopo.

Ogni volta, che viene negoziata una nuova SA si verifica un nuovo scambio Diffie-Hellman, con conseguente *overhead* delle risorse (sui dispositivi moderni l'overhead è trascurabile). Entrambi i peer del tunnel devono essere in grado di supportare il PFS affinché funzioni, se un solo peer lo supporta la richiesta di connessione verrà negata. La procedura migliore consiste nel configurare tutti i peer con PFS attivo (se supportato), così da non doverlo configurare ogni volta che si vuole stabilire una SA con un nuovo dispositivo che prova a connettersi al tunnel.

Di seguito vediamo un diagramma di sequenza in UML per il funzionamento delle due fasi di IKE con la prima in main mode. Al termine delle due fasi sono creati una ISAKMP SA e un'IPsec SA.

Figura 1.13: IPsec Sequence Diagram



Dopo aver illustrato i vari protocolli che compongono IPsec, come lavora e le sue modalità, nel capitolo successivo si tratteranno le vulnerabilità della suite di protocolli.

2. Vulnerabilità di IPsec

IPsec offre numerose opzioni di configurazioni che influenzano in vario modo le prestazioni e la sicurezza dei tunnel creati.

In questo capitolo verranno descritte le principali vulnerabilità scoperte, sia dei vari protocolli di IPsec sia dei vari algoritmi usati per la cifratura e autenticazione. Configurare un tunnel IPsec con uno dei protocolli/algoritmi descritti in seguito, senza le dovute precauzioni, comporterebbe un grave rischio per la sicurezza dei dati inviati attraverso il tunnel ed inoltre un rischio per tutti i *peer* collegati ad esso.

2.1 Vulnerabilità IKE

In questo paragrafo vengono descritti attacchi che sfruttano vulnerabilità nel riuso delle chiavi (*key reuse*) in IKE [7], descritte in *Bleichenbacher's attack* [8]; se un'implementazione di RSA comincia con due byte: 0x00 0x02 allora l'attacco di Bleichenbacher è possibile.

> Breve introduzione ad RSA

RSA è un cifrario a chiave pubblica che permette di cifrare un messaggio attraverso un procedimento che sfrutta le proprietà dei numeri primi. Una volta che un messaggio è stato cifrato con la chiave pubblica, può essere decifrato solo con un'altra chiave, chiamata privata.

RSA viene spesso utilizzata in combinazioni con altri schemi di crittografia o per le firme digitali; in genere non viene utilizzata per crittografare messaggi poichè meno efficiente e richiede più risorse rispetto alla crittografia a chiave simmetrica. Di seguito i vari passi per la generazione della chiave pubblica e privata:

■ Generazione della chiave pubblica

1. Il primo passo consiste nel creare le chiavi; per fare ciò abbiamo bisogno di due numeri primi p e q abbastanza grandi;
2. Si moltiplicano p e q per trovare n ;
3. Una volta trovato n si calcola:

$$\varphi(n) = (p - 1) * (q - 1) \quad (2.1)$$

Dove $\varphi(n) = (p - 1) * (q - 1)$ è la funzione di Eulero $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1)$. Trovato φ si sceglie un numero tra 1 e φ randomicamente, chiamato e coprimo con φ .

4. Dopo aver trovato e si procede a crittografare il messaggio m con la seguente formula:

$$c = m^e \bmod n \quad (2.2)$$

La chiave pubblica sarà quindi (n, e) .

■ Generazione della chiave privata

Le chiavi private sono composte da d e n , dove n è stato calcolato nella chiave pubblica e troviamo d con i seguenti passi:

$$d = e^{-1} \bmod \varphi(n) \quad (2.3)$$

dove $e^{-1} \bmod$ è l'inverso modulare di e . Una volta trovato d si procede a decifrare il messaggio m , con:

$$m = c^d \bmod n \quad (2.4)$$

La chiave privata sarà quindi (n, d) . Una volta che il mittente ha la chiave pubblica del destinatario, può utilizzarla per crittografare i dati che desidera proteggere; quando il destinatario riceve il messaggio crittografato, utilizza la propria chiave privata per accedere ai dati.

La complessità di RSA sta nella fattorizzazione in numeri primi di un numero composto da 2048 bit o superiori, per trovare p e q da n e calcolare $\varphi(n)$ nella chiave segreta; se un'attaccante entrasse in possesso di p e q sarebbe in grado di decifrare i dati molto facilmente criptati con RSA.

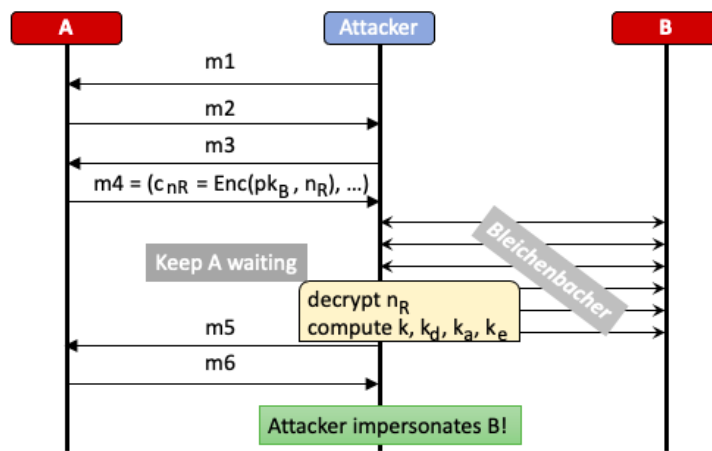
> Attacco Bleichenbacher contro PKE e RPKE

In Figura 2.1 si vede una rappresentazione grafica dell'attacco e di seguito una spiegazione nel dettaglio il cui protagonista è l'attaccante.

1. Inizia una PKE⁸ in IKEv1, basata sullo scambio di chiavi con A e aderisce al protocollo fino a che non riceve il messaggio m_4 ; estrae c_{nR} da questo messaggio, registra il valore C_I , C_R insieme al valore n_I e alla chiave privata DHKE x scelta;
2. Mantiene l'handshake con A aperto per un tot di secondi (variabili) $t_{timeout}$;
3. Inizia molte PKE parallele basate sullo scambio di chiavi con B ; dove in ogni messaggio invia e riceve i primi due messaggi, nel messaggio m_3 include la versione modificata di C_{nI} ed infine attende fino a che si riceve il messaggio m_4 oppure si ha un timeout;
4. Dopo aver completato il passo sopra si calcola n_R da DHKE condivisa da A e la loro DHKE privata x ; dopo aver fatto ciò calcolano g^{xy} ;
5. L'attaccante ora ha tutte le informazioni per completare l'attacco; può calcolare MAC_I e inviare il messaggio m_5 cifrato con la chiave k_e , e può impersonare B senza che A se ne accorga.

⁸Public Key Encryption (PKE) è un metodo di criptazione con due differenti chiavi, tra cui una pubblica e una privata. I dati criptati con la chiave pubblica possono essere decifrati solo con la chiave privata.

Figura 2.1: Bleichenbacher's attack



La descrizione teorica delle chiavi pubbliche indica che, nuove chiavi dovrebbero essere generate ad ogni nuovo utilizzo; ma spesso nella pratica questo è molto difficile e a volte non supportato. Nella realtà, è pratica comune avere un paio di chiavi RSA per tutti i protocolli (*key reuse*), che espone l'utilizzo delle chiavi ad attacchi di crittoanalisi e attacchi al dizionario.

➤ Attacco a dizionario in IKEv1 Main Mode con PSK

Il metodo di autenticazione con PSK è vulnerabile ad un attacco offline al dizionario, nella modalità aggressive; per quanto riguarda invece la modalità main è necessario effettuare un attacco online. Se l'attaccante ha intercettato la DHKE, è possibile effettuare un attacco a dizionario offline; una volta intercettato la DHKE e la PSK⁹, l'attaccante può calcolare il valore di MAC_I . Conoscendo la PSK questo può impersonare qualsiasi *peer* all'interno del tunnel IPsec. Tale attacco ha bisogno di un solo handshake, in cui l'attaccante impersona il risponditore; tramite questo handshake si apprendono abbastanza informazioni per montare un attacco al dizionario offline. Per fare ciò l'attaccante resta in attesa che la vittima e il risponditore hanno già una connessione attiva può iniziare un nuovo handshake per stabilire una nuova connessione; durante questo handshake non invia nessun pacchetto ma simula di essere il risponditore scambiando i messaggi della fase 1, ricevendo così il messaggio m_5 ; con questo messaggio, riceve ID_I e MAC_I criptati con la chiave k_e . La chiave k è derivata direttamente dalla PSK; questo permette di effettuare l'attacco a dizionario contro di essa ed in seguito è capace di decifrare il messaggio m_5 .

Tutte le versioni di IKE sono vulnerabili a questo tipo di attacco ed una PSK debole, permette ad un'attaccante con un attacco a dizionario di interpretare il risponditore; quindi una soluzione potrebbe essere quella di usare un PSK forte ed inoltre le modalità PKE e RPKE dovrebbero essere disattivate da ogni dispositivo che utilizza IKE.

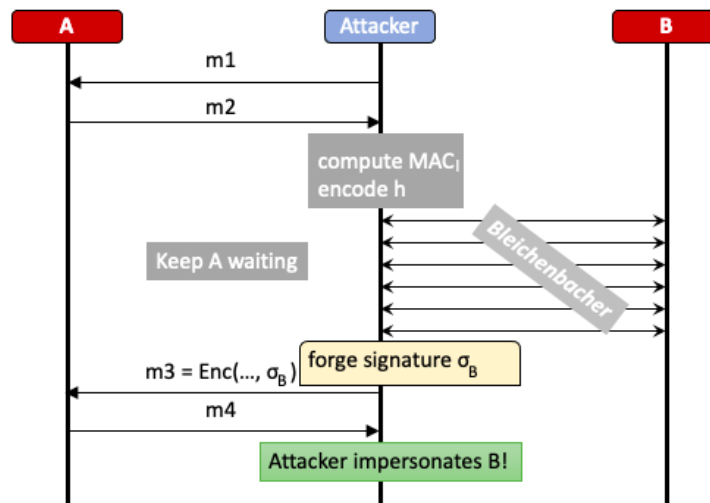
⁹La *Pre-Shared Key* (PSK) è una chiave segreta precedentemente condivisa tramite canali sicuri ed è un metodo di autenticazione.

➤ Attacco Bleichenbacher contro firma digitale basata sull'autenticazione

L'attacco contro IKEv2 con firma basata sull'autenticazione (Figura 2.2) è adattabile facilmente a IKEv1, ed è descritto dai seguenti passi:

- L'attaccante inizia uno scambio di firma basata sullo scambio di chiavi con il risponditore A e aderisce al protocollo fino a che non riceve il messaggio m_2 . Una volta ricevuto questo messaggio, possiede tutte le informazioni per derivare la chiave; da questa chiave è necessario trovare k_{pI} per calcolare $MAC_I = PRF_{k_{pI}}(ID_B)$ che è la parte dei dati firmati con la chiave privata del risponditore B ;
- Si tiene l'handshake con il risponditore A per un periodo di tempo massimo chiamato $t_{timeout}$;
- Ora l'attaccante codifica l'hash h con la PKCS #1 per la firma;
- Proceda ad iniziare molte PKE parallele basate sullo scambio di chiavi con il risponditore B ;
 - * In ognuno di questo scambi, invia e riceve i primi due messaggi;
 - * Il messaggio m_3 , include una versione modificata di c ;
 - * Attende di ricevere la risposta m_4 .
- Dopo avere ricevuto abbastanza risposte dal risponditore B , può ora calcolare la decifrazione di una firma valida;
- Infine l'attaccante completa l'handshake inviando il messaggio m_3 , includendo una firma valida al risponditore A impersonando il risponditore B .

Figura 2.2: Bleichenbacher's attack contro firma digitale



2.2 Vulnerabilità ESP

In questo paragrafo, vengono descritti attacchi contro IPsec che sfruttano vulnerabilità presenti in alcuni tipi di configurazione: quelle che utilizzano solo la crittografia senza autenticazione [9], quelle che utilizzano ESP in modalità tunnel senza funzionalità di controllo dell'integrità dei dati presente ai layer superiori ma anche quelle che utilizzano AH. In queste configurazioni vulnerabili, un'attaccante può essere in grado di modificare delle sezioni del pacchetto IPsec in modo che, una volta ricevuto dal gateway questo lo instradi (dopo averlo decrittato) verso un host controllato dall'attaccante stesso, oppure 'forzando' il gateway a generare un messaggio di errore. Tale messaggio di errore è inviato attraverso il protocollo ICMP il quale, per sua natura, conterrà l'header IP e parte del payload in chiaro. La causa principale del problema è dovuta al fatto che l'ESP usa l'algoritmo *Cipher Block Chaining* (CBC) per criptare i dati; tale algoritmo è infatti notoriamente vulnerabile ad attacchi di tipo 'bit flipping' (spiegato in seguito). Questi attacchi sono applicabili solo se un'implementazione IPsec esegue un controllo sul riempimento del padding e può estrarre il contenuto completo di un datagramma IPsec cifrato; non richiedono che un'attaccante sia in grado di monitorare tutto il traffico, ma solo il traffico nelle due direzioni di un tunnel IPsec.

■ Attacco al Padding

L'idea alla base di questo attacco è quella di manipolare il campo dell'header di un datagramma protetto usando ESP per ottenere il testo cifrato. Tutto dipende da come IPsec gestisce il *padding* e la modalità di cifratura in CBC.

➤ Funzionamento CBC in ESP tunnel mode

Il datagramma originale (inner) che deve essere protetto è trattato come una sequenza di byte. A questa sequenza viene aggiunto il padding con un particolare pattern di byte, il *Pad Length* (PL) e il *Next Header* (NH) vengono aggiunti in coda; il numero di byte presenti (incluso PL e NH) sono conformi alla lunghezza del blocco e il pattern del padding aggiunto è: o una stringa vuota oppure t byte della forma $1, 2, 3, \dots, t$ per un qualche t con $1 \leq t \leq 255$. In tunnel mode è anche permesso far precedere questo padding con una somma arbitraria di *Traffic Flow Confidentiality* (TFC)¹⁰ padding; successivamente, i dati sono criptati usando la modalità CBC. La sequenza di byte consiste di q blocchi, ognuno di n bit (64 in 3DES per esempio) denotati con P_1, P_2, \dots, P_q . Il blocco di testo cifrato è generato con questa equazione:

$$C_0 = IV, C_i = e_K(C_{i-1} \oplus P_i), (1 \leq i \leq q) \quad (2.5)$$

Dove IV è l'*Initial Vector* ovvero un bit n selezionato randomicamente, K è la chiave usata per l'algoritmo di cifratura dei blocchi e $e_K(\cdot)(d_K(\cdot))$ è per denotare la cifratura (decifratura) dei blocchi usando la chiave K . La porzione cifrata del datagramma esterno è definita per essere una sequenza di $q + 1$ blocchi C_0, C_1, \dots, C_q .

¹⁰I TFC sono tecniche ideate per nascondere/mascherare il flusso del traffico per prevenire attacchi di analisi statistica.

➤ Depadding e decifratura in CBC ESP Tunnel mode

Per la decifratura il payload del datagramma esterno può essere recuperato con la seguente equazione:

$$P_i = C_{i-1} \oplus d_K(C_i), \quad (1 \leq i \leq q) \quad (2.6)$$

A questo punto, il ricevente può ricostruire il datagramma originale (inner) criptato in precedenza; le implementazioni devono corrispondere a quelle specificate nelle policies di IPsec. Se il controllo fallisce, allora il datagramma può essere ignorato, al contrario, il datagramma è passato alle implementazioni IP per processamenti futuri.

➤ Bit Flipping in CBC

Un'attaccante che cattura un testo cifrato con CBC C_0, C_1, \dots, C_q può invertire uno specifico bit j in C_{i-1} e iniettare un testo cifrato nella rete; esso induce un'inversione del bit in posizione j nel blocco del testo in chiaro P_i come visto nella parte della decifratura. Questo tende a randomizzare il blocco P_{i-1} , se questo risulta essere C_0 (IV) non ha alcun effetto sul testo cifrato.

Supponiamo che si voglia decriptare un blocco di testo cifrato C_i , $i \geq 1$ da un testo cifrato CBC che consiste di blocchi C_0, C_1, \dots, C_q ; ogni blocco C_j ha t byte chiamati $C_{j,0}, C_{j,1}, \dots, C_{j,t-1}$ da sinistra a destra. L'attaccante crea due blocchi di testo cifrato della forma R, C_i e li sottomette per essere processati. Se il processamento conferma che il padding è corretto allora l'ultimo byte a destra di $R \oplus d_K(C_i)$ è probabile che sia 0; adesso può facilmente calcolare $(d_K(C_i))_{t-1}$ e quindi $P_{i,t-1}$. Al contrario se il processamento indica che il padding non è corretto, allora può provare con un differente valore per R_{t-1} . L'attaccante può ora estrarre il byte più a destra di P_i , usando una media di 128 interrogazioni ed un massimo di 256 interrogazioni. Una volta che il byte più a destra è stato estratto, si conosce $(d_K(C_i))_{t-1}$; per ottenere $P_{i,t-2}$ possiamo aggiustare R_{t-1} che diventa $R_{t-1} \oplus (d_K(C_i))_{t-1} = 1$ e cambiare R_{t-2} fino a che non si trova il padding corretto. Il processo di decifratura continua in questo modo fino a che ogni blocco di testo in chiaro è stato estratto; questo richiede una media di 128 interrogazioni.

Fortunatamente l'attacco descritto sopra non funziona contro un'implementazione reale di IPsec, poichè non si è a conoscenza in precedenza di quale tipo di padding viene aggiunto o quale meccanismo sia utilizzato per il calcolo del padding; inoltre l'attacco utilizza testi cifrati della forma R, C_i dove C_i è il blocco target e i dati prodotti dopo la decifratura non corrispondono ad un valido datagramma IP.

■ Attacco al testo in chiaro

Per ottenere il risultato desiderato dell'attacco si assume che i blocchi cifrati usati in ESP devono essere di 64 bit, inoltre:

- ESP è usato in modalità tunnel tra due gateway G_A e G_B ; questi forniscono sicurezza a due host comunicanti H_A e H_B localizzati dietro i gateway (Figura 2.3);
- La chiave K usata per la cifratura del traffico IP tra G_A e G_B è fissata durante l'attacco;
- L'attaccante può monitorare e catturare i datagrammi cifrati con ESP tra i due gateway;
- L'attaccante può iniettare datagrammi modificati nel network tra G_A e G_B .

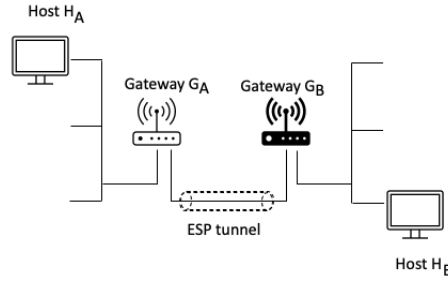


Figura 2.3: Configurazione rete

Si assume inoltre che una SA protegge il traffico tra G_A e G_B , e viceversa; inoltre una specifica ICMP-SA è usata per proteggere i messaggi ICMP mandati da G_A a G_B e viceversa. L'attaccante deve aver raccolto un testo cifrato $C = C_0, C_1, \dots, C_q$ dal payload del datagramma esterno mandato a G_B . Questo rappresenta il traffico tra H_A e H_B .

L'obiettivo dell'attaccante è quello di scoprire il testo in chiaro corrispondente a C ; per ottenere ciò, deve compiere alcuni passi descritti in seguito:

► Preparazione

Si assume che l'attaccante abbia ottenuto il testo cifrato C^j corrispondente ad un insieme di 7 testi in chiaro scelti P^j ($0 \leq j \leq 6$); P^j contiene l'inner datagramma IP con l'indirizzo di partenza, l'indirizzo di destinazione e un campo TTL¹¹ settato a 1. Si assume inoltre che P^j contenga esattamente $j + 12$ byte del payload; P^j inizia con 20 byte dell'header IP, seguito da $j + 12$ byte del payload per un totale di 32 byte. Se C^j è iniettato nel network come payload di un datagramma con l'indirizzo di destinazione di G_B , allora l'inner datagramma di G_B sarà recuperato e passato all'implementazione dell'IP di G_B . Questo decremerà il campo TTL dell'inner datagramma, producendo un campo TTL con valore 0. L'implementazione IP di G_B produrrà un messaggio ICMP di errore che indica "time to live exceeded"; il messaggio di errore avrà come destinazione l'host H_A e diventerà un inner datagramma cifrato con ESP in G_B . Il messaggio ICMP può essere rilevato dall'attaccante anche se contenuto in un datagramma cifrato con IPsec; ma se è presente un padding TFC è più difficile separarlo dal resto del traffico cifrato.

► Estrazione degli ultimi due byte di un blocco

L'attaccante seleziona un blocco arbitrario C_i , con $i \geq 1$, dal testo cifrato. Dopo l'iniezione nella rete il datagramma esterno contiene il payload come testo cifrato:

$$C' = C_0^6, C_1^6, C_2^6, C_3^6, R^6, C_i \quad (2.7)$$

dove R^6 è il blocco random, la sua posizione non disturba l'header dell'IP nel corrispondente testo in chiaro P' . G_B ignorerà l'inner datagramma fino a che il padding non sia valido e il byte NH sia uguale a 4. La lunghezza e il campo checksum nell'header IP saranno ancora corretti e il campo TTL avrà il valore 1. In questa situazione avremmo un messaggio di errore ICMP che sarà cifrato nel nono blocco del testo cifrato e l'attaccante lo potrà rilevare

¹¹Il *Time to Live* (TTL), relativo ad un pacchetto IP, è un meccanismo che determina il numero di volte al quale un router può accedervi prima che esso venga distrutto.

in base alla sua lunghezza; una volta rilevato saprà che gli ultimi due byte di $R^6 \oplus d_K(C_i)$ sono uguali a 0 e 4. Ora dovrà provare ad estrarre i byte $P_{i,6}, P_{i,7}$; il numero massimo di prove è 2^{16} con una media di 2^{15} . Potrà estrarre il byte 6 e 7 di P_i simultaneamente.

► Estrazione dei rimanenti byte di un blocco

Una volta estratti gli ultimi due byte, l'attaccante può estrarre i rimanenti byte più facilmente lavorando in sequenza da destra ($P_{i,5}$) a sinistra ($P_{i,0}$); quindi ora lavorerà con un testo cifrato della forma:

$$C' = C_0^5, C_1^5, C_2^5, C_3^5, R^5, C_i \quad (2.8)$$

dove R^5 è inizialmente settato al valore di R^6 che ha prodotto il messaggio ICMP quando estratti gli ultimi due byte, eccetto quando settiamo $R_6^5 = R_6^6 \oplus 1$. Questo assicura che P^i , la versione decifrata di C' , finisca con i byte 1 e 4. Ora l'attaccante varia R^5 nel byte 5 e inietta il testo cifrato modificato; solamente quando P' finisce con 1,1,4 possiamo guardare il blocco al contrario. Quando questo è stato rilevato sappiamo che il byte 5 di $R^5 \oplus d_K(C_i)$ è uguale a 1. Da questo possiamo dedurre il valore di $P_{i,5}$. Continuando in questo modo si possono estrarre i rimanenti byte di P_i con una media di 128 tentativi ed un massimo di 256, usando una versione modificata di C^j ed estendendo il padding un byte alla volta.

La complessità dell'attacco è $2^{16} + 6 \cdot 2^8$ tentativi per blocco di testo cifrato, con una media di quasi la metà.

■ Attacco basato sulle elaborazioni delle opzioni

Questo attacco è basato sulle elaborazioni delle opzioni. L'idea alla base di questo attacco è quella di prendere un testo cifrato esistente e trasformarlo in un testo cifrato con dentro un inner datagramma, il quale produce un messaggio ICMP di errore per il fallimento delle elaborazioni delle opzioni. Si suppone che l'attaccante ha raccolto un numero di testi cifrati della forma C_0, C_1, \dots, C_q dal payload del datagramma esterno diretto a G_B .

► Preparazione

Chiamiamo C' uno del testo cifrato target dal tunnel. La fase di preparazione può essere condotta solamente in C' dopo il quale, ogni blocco non-IV, da qualsiasi altro testo cifrato può essere decifrato efficacemente. La fase di preparazione può essere descritta così:

1. Modifica del blocco C'_0 invertendo i bit 6 e 7 per ottenere un nuovo blocco C''_0 ;
2. Settare un contatore i a 0;
3. **Ripeti:**
 - Modifica del blocco C''_0 così che i byte 4 e 5 contengano la rappresentazione binaria di i ;
 - Preparare un datagramma esterno diretto a G_B con il payload identico a C' eccetto per C'_0 che è rimpiazzato con i blocchi modificati nel passo sopra. Inietta questo datagramma modificato nella rete;
 - Incrementa i .
4. **Until** un testo cifrato ICMP appare nel tunnel tra G_B e G_A .

Il calcolo del checksum usato dall'IP include tutti i byte dell'header, inclusi ogni byte opzionale. Dopo aver modificato i byte 4 e 5 di C'_0 , possiamo iniettarli nella rete come payload del datagramma esterno diretto a G_B ; dopo la decifratura e depadding, il risultante datagramma inner sarà passato nell'implementazione IP di G_B per processamenti futuri. Alla fine di questa fase di preparazione, l'attaccante possiede un testo cifrato C^* con $r + 1$ blocchi.

► Estrazione degli ultimi due byte di un blocco

L'attaccante è ora pronto a usare il blocco $r + 1$ del testo cifrato C^* con i blocchi denotati con $C_0^*, C_1^*, \dots, C_r^*$ per attaccare un blocco arbitrario C_i ; inietta nella rete un datagramma esterno contenente il payload di $r + 3$ blocchi di testo cifrato:

$$C^\# = C_0^*, C_1^*, \dots, C_r^*, R^6, C_i \quad (2.9)$$

dove R^6 è un blocco random del blocco target C_i . Ora l'attaccante varia R^6 in byte 6 e 7, inietta il testo cifrato modificato e controllo il messaggio ICMP nella direzione opposta. Quando è rilevato, l'attaccante sa che gli ultimi due byte di $R^6 \oplus d_K(C_i)$ sono uguali a 0 o 4; da qui prova ad estrarre i byte 6 e 7 con un massimo di tentativi di 2^{16} ed una media di 2^{15} .

► Estrazione dei rimanenti byte di un blocco

Si procede all'estrazione dei rimanenti byte; ora lavora con un testo cifrato della forma:

$$C^\# = C_0^*, C_1^*, \dots, C_r^*, R^5, C_i \quad (2.10)$$

dove R^5 è uguale al valore di R^6 che produce un messaggio ICMP quando estratti gli ultimi due byte. Questo assicura che $P^\#$ la versione decrittata di $C^\#$ finisca con 1,4. Ora l'attaccante modifica R^5 nel byte 5 e inietta il testo cifrato modificato. Solamente quando P' finisce con 1,1,4 rivedremo il messaggio ICMP nella direzione opposta. Da questo possiamo dedurre il byte 5 di P_i , continuando così tutti i byte di P_i saranno estratti con una media di 128 tentativi ed un massimo di 256.

La complessità dell'attacco è di 2^{15} tentativi.

Per implementare questi attacchi si devono trovare implementazioni di IPsec che ispezionano il padding e scartano il datagramma inner nel caso in cui non sia correttamente formattato. Questi attacchi hanno dimostrato che usare IPsec con la sola cifratura non è sicuro e quindi sconsigliato; per ovviare a questo problema si può usare la modalità di cifratura combinata con l'autenticazione e integrità.

2.3 Vulnerabilità algoritmi usati in IPsec

In questo capitolo vengono descritte le vulnerabilità dei vari algoritmi usati, per la cifratura e per l'autenticazione, in IPsec. Ogni algoritmo descritto in seguito presenta una o più vulnerabilità note che, rendono vulnerabili le implementazioni di IPsec che usano questi algoritmi, permettendo ad un'attaccante di intercettare comunicazioni crittate tra due interlocutori (hosts, gateway, ecc..) e scoprire informazioni che dovrebbero rimanere cifrate.

2.3.1 Data Encryption Standard

L'algoritmo Data Encryption Standard (DES) [10], è un cifrario a blocchi a chiave simmetrica creato negli anni '70 da ricercatori di IBM e certificato dal *National Institute of Standards and Technology* (NIST) nel 1977. Questo algoritmo suddivide il testo in chiaro in blocchi da 64 bit e li converte in testo cifrato utilizzando chiavi a 64 bit, inclusi 8 bit di controllo che rendono la lunghezza della chiave di 56 bit. Siccome è un algoritmo a chiave simmetrica, utilizza la stessa chiave sia nella cifratura che nella decifratura dei dati. Si basa sul cifrario a blocchi di Feistel¹², dove il messaggio alfanumerico viene convertito in una sequenza di 0 e 1 e suddiviso in blocchi da 64 bit.

Vediamo la struttura generale di DES in Figura 2.4 e di seguito il funzionamento in dettaglio di come funziona l'algoritmo:

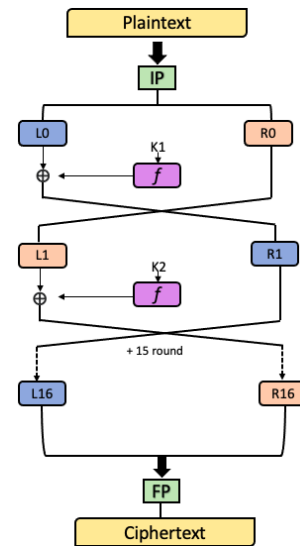


Figura 2.4: Struttura generale di DES

- Il blocco del testo in chiaro inizialmente viene convertito in una permutazione iniziale (IP);
- La permutazione iniziale produce due metà del blocco permutato, chiamate *Left Plain Text* (LPT) e *Right Plain Text* (RPT) in figura L e R;
- Ogni blocco LPT e RPT eseguono i 16 passi per il processo di crittografia;
- Alla fine dei 16 passi, LPT e RPT sono ricombinati insieme e formano una Final Permutation (FP), anche chiamata IP^{-1} poiché inverte le operazioni svolte da IP, la quale viene eseguita sui blocchi combinati;
- Il risultato di questo processo produce un testo cifrato di 64 bit.

Il DES utilizza due funzioni per generare l'output richiesto; queste funzioni vengono richiamate all'interno della struttura generale dell'algoritmo; di seguito è descritto il loro funzionamento nel dettaglio. L'algoritmo (Figura 2.4), è composto da 16 fasi identiche chiamate *round* o cicli; sono inoltre presenti la permutazione iniziale IP e quella finale FP, l'una l'inversa dell'altra, le quali non hanno nessuna importanza per la cifratura. Inizialmente il blocco è suddiviso in due metà di 32 bit e processato alternativamente; procedendo in questo modo il processo di cifratura e

¹²Il cifrario di Feistel, detto anche rete di Feistel è stato creato dal crittologo Horst Feistel nel 1973; si tratta di un cifrario che mescola trasposizioni e sostituzioni a livello di bit, non di carattere. Non ha mai avuto applicazioni pratiche ma è stato il precursore di cifrari reali come il DES.

decifratura risultano simili, con la sola differenza sulle sottochiavi che sono applicate nell'ordine inverso nella fase di decifratura. Il simbolo \oplus denota l'operazione di OR esclusivo (XOR). La Round Function (f nella Figura 2.4, spiegata in dettaglio nel seguito), viene messa in \oplus con la metà sinistra del blocco mescolandolo con una parte della chiave. Il risultato della Round Function è poi combinato con l'altra metà del blocco, e le due metà sono scambiate prima del ciclo successivo; dopo l'ultimo ciclo, le due metà non vengono scambiate.

■ Round Function

La Round Function (Figura 2.5(a)) è composta dai seguenti blocchi:

- Expansion Box che realizza una permutazione ed espansione, opera su mezzo blocco di 32 bit che espande fino a 48 duplicando alcuni bit;
- L'output del blocco precedente viene messo in XOR con la sottochiave K_I , creando quindi sedici sottochiavi di 48 bit (una per ogni ciclo) dalla chiave principale;
- Dopo la combinazione con la sottochiave, il blocco viene diviso in 8 parti di 6 bit, prima del processamento con le *S-box*¹³, che realizzano una sostituzione e compressione da 6 bit a 4;
- Straight Box che realizza, un'ulteriore permutazione con i 32 bit risultanti dalle S-box riordinandoli in base a delle permutazioni fisse.

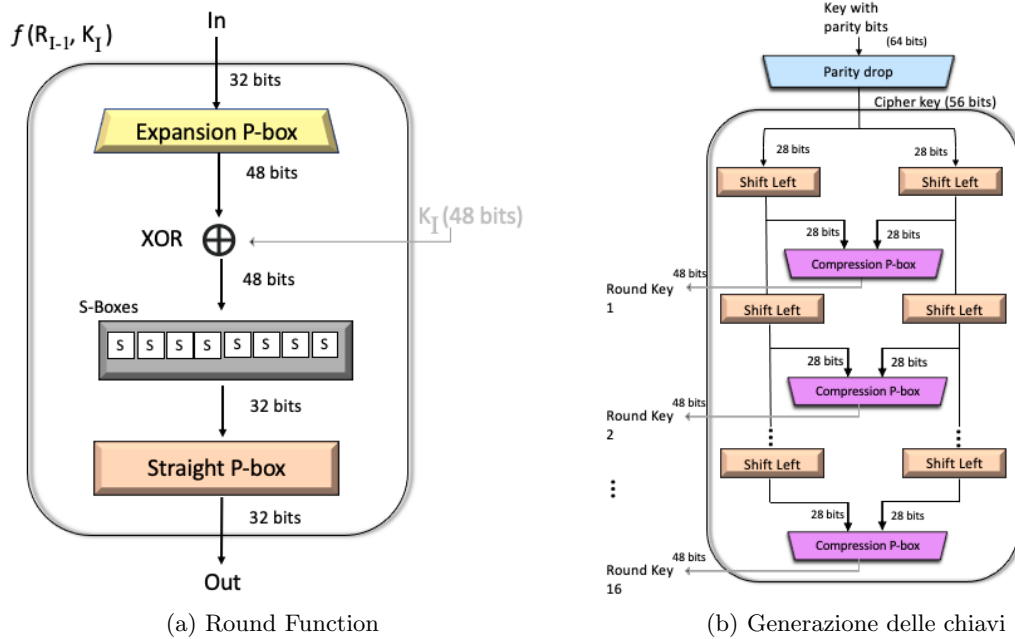


Figura 2.5: Round Function e Generazioni delle chiavi in DES

¹³Le S-box o Substitution box vengono utilizzate negli algoritmi a chiave simmetrica per oscurare relazioni tra il testo in chiaro e il testo cifrato seguendo il principio della confusione [11]. Le S-box forniscono il cuore della sicurezza del DES, senza le quali la cifratura sarebbe lineare e quindi facilmente violabile.

■ Generazione delle chiavi

Le sottochiavi sono ottenute tramite operazioni di permutazione e shift di 1 o 2 bit a sinistra. La chiave iniziale a 56 bit viene modificata e portata a 64 bit, tramite l'aggiunta di un bit di parità ogni 7 bit. La generazione delle chiavi per la decifratura è simile ma deve generare le chiavi nell'ordine inverso quindi lo shift è verso destra invece che sinistra (Figura 2.5(b)). L'operazione di decodifica, segue gli stessi passi; ma in tal caso i dati in ingresso sono costituiti da un blocco di 64 bit di testo cifrato.

Come descritto le chiavi di questo algoritmo hanno una lunghezza di 56 bit, quindi 2^{56} possibili combinazioni; inizialmente la lunghezza della chiave era sufficiente per essere considerato un algoritmo sicuro e non-violabile, avendo anche altre caratteristiche desiderate di un cifrario a blocchi, come l'*effetto valanga* dove un piccolo cambiamento nel testo in chiaro si traduce in un cambiamento sostanziale nel testo cifrato, e la *completezza* dove un bit di testo cifrato dipende da molti bit di testo in chiaro. Con l'avanzare delle tecnologie il calcolo delle possibili combinazioni delle chiavi, inizialmente incalcolabile in tempi utili, divenne sempre più semplice da calcolare. Già dalla fine degli anni '90 l'*Electronic Frontier Foundation* (EFF) costruì un dispositivo chiamato **Deep Crack**, che conducendo un attacco a forza bruta (*Brute Force attack*)¹⁴, riuscì a recuperare la chiave segreta di un messaggio cifrato con il DES in 56 ore. L'attacco brute force di Deep Crack mostrò definitivamente che il DES è facilmente violabile; tale che nel 2005 il NIST lo dichiarò un algoritmo *non* sicuro e violabile.

Esistono attacchi più rapidi di quello a forza bruta in grado di violare DES, che sono: la crittanalisi differenziale, la crittanalisi lineare e l'attacco di Davies; ad ogni modo questi attacchi sono solo teorici e non applicabili nella realtà.

2.3.2 3DES

Il Triple DES o 3DES è un cifrario a blocchi creato utilizzando tre volte il DES. La lunghezza della chiave del DES, non sufficiente a prevenire attacchi di forza bruta, ha creato il bisogno di avere un algoritmo crittografico più forte contro questo tipo di attacco; così nel 1999 il NIST ha approvato il Triple DES. Il 3DES ha 3 chiavi, che chiameremo K_1 , K_2 e K_3 , ognuna con una lunghezza di 64 bit inclusi gli 8 di controllo, con una lunghezza totale di 168 bit ($3 \times 64 - 3 \times 8$); portati a 192 con l'aggiunta di bit di parità. Ma la sicurezza garantita è di soli 112 bit.

Il funzionamento del 3DES è il seguente:

1. Cripta i blocchi del testo in chiaro utilizzando il DES con la chiave K_1 ;
2. Decripta l'output del passo 1 utilizzando il DES con la chiave K_2 ;
3. Cripta l'output del passo 2 utilizzando il DES con la chiave K_3 ;
4. L'output del passo 3 è il testo cifrato.

Possiamo riassumere la cifratura di 3DES in questa formula: $C = E(K_3, D(K_2, E(K_1, M)))$ dove M è il messaggio in chiaro, $E(K_1, M)$ indica la cifratura di M mediante la chiave K_1 , $D(K_2, E(K_1, M))$ indica la decifratura del messaggio precedentemente criptato utilizzando la chiave K_2 ed infine $E(K_3, D(K_2, E(K_1, M)))$ indica la cifratura del messaggio appena decriptato, però con la chiave K_3 . Per la decifratura si utilizzerà la stessa formula, ma utilizzando le chiavi in ordine inverso.

¹⁴Un attacco di forza bruta consiste in un algoritmo che verifica tutte le soluzioni teoricamente possibili fino a che, trova quella effettivamente corretta. Il suo punto di forza principale è che riesce sempre a trovare la soluzione corretta, ma è sempre la soluzione più lenta e/o dispendiosa.

La procedura di cifratura e decifratura è la stessa di DES (descritta nel paragrafo 2.3.1), ma viene ripetuta tre volte (Figura 2.6).

Esiste anche un'altra versione del 3DES in cui le chiavi K_1 e K_3 sono uguali, ma questo riduce la lunghezza complessiva delle chiavi arrivando a 128 bit di cui 16 di controllo. L'algoritmo 3DES è significativamente più sicuro di DES contro attacchi di tipo brute force e più robusto contro attacchi basati sulla crittoanalisi ma, la vulnerabilità trovata nell'uso di blocchi da 64 bit (*Sweet32 attacks*, spiegato nel seguito di questo capitolo), la complessità di implementazione a livello software e la velocità dell'algoritmo (3 volte più lento di DES), hanno fatto sì che nel 2018 il NIST ha classificato il 3DES non sicuro e dal 2023 non sarà più possibile utilizzarlo lasciando il posto al più sicuro *Advanced Encryption Standard* (AES).

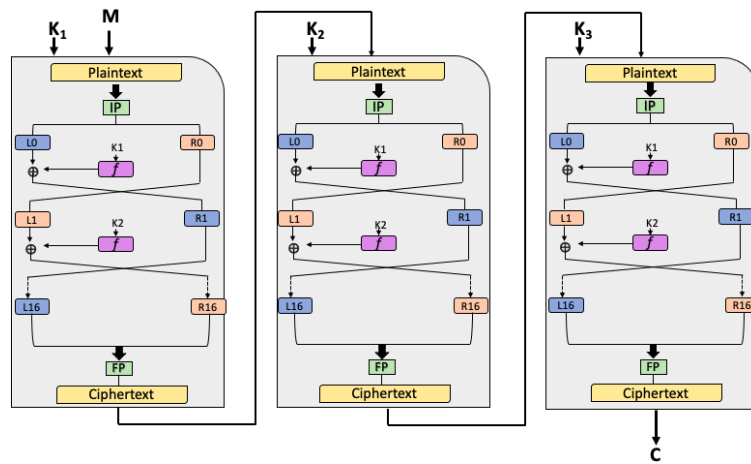


Figura 2.6: Triple DES

➤ Sweet32 attacks

L'attacco *Sweet32* [12] [CVE-2016-2183], sfrutta una vulnerabilità dei cifrari con blocchi di 64 bit (come DES/3DES e Blowfish); è stata scoperta da ricercatori del *Institut National de Recherche en Informatique et en Automatique* (INRIA) e permette ad un'attaccante di recuperare piccole porzioni del testo in chiaro. Questo algoritmo divide i dati in blocchi di piccole dimensioni, i quali sono vulnerabili al *birthday attacks* (attacco del compleanno)¹⁵ dove ci sarà una situazione in cui due blocchi avranno la stessa chiave. Un'attaccante può accedere alle informazioni utilizzando un'operazione di XOR sui blocchi per rilevare il testo in chiaro. Per ottenere la stessa probabilità statistica del *paradosso del compleanno* è necessario catturare almeno **785 Gigabyte** di traffico per individuare i blocchi del cookie di autenticazione da utilizzare per la decodifica delle comunicazioni. Questo attacco prevede la visita forzata della vittima designata su un sito malevolo (che nel seguito chiameremo *A*), su cui dovranno essere registrate le credenziali di accesso necessarie a impostare il *cookie*; dal sito *A* la vittima verrà indirizzata a quello originario (che chiameremo *B*), e l'attaccante dovrà restare in ascolto per catturare tutti i blocchi di traffico sufficienti (765 GB) per identificare una *collisione*. A quel punto si dovrà procedere alla decodifica

¹⁵Un attacco del compleanno [13], è un tipo di attacco crittografico basato sui principi matematici del paradosso del compleanno nella teoria delle probabilità. Data una funzione f , lo scopo dell'attacco è quello di trovare 2 numeri x_1, x_2 tali che $f(x_1) = f(x_2)$; tale coppia di valori x_1, x_2 è chiamata *collisione*.

del cookie di login. La messa in pratica di questo attacco nel mondo reale è resa difficile dal tempo di esecuzione, poiché bisogna mantenere attiva la connessione ad un server per periodi di tempo molto lunghi.

Possono essere prese alcune misure per prevenire questo tipo di attacco:

- Usare cifrari con blocchi di 128 bit;
- Usare il più sicuro AES e mantenere il 3DES come seconda opzione;
- Limitare la durata della sessione dove vengono usati i cifrari con 64 bit.

2.3.3 Blowfish

L'algoritmo Blowfish è un algoritmo a chiave simmetrica, il che vuol dire che viene utilizzata la stessa chiave per criptare e decriptare i messaggi; è anche un cifrario a blocchi, basato sulla funzione di Feistel, la cui lunghezza di ogni blocco è di 64 bit (come DES/3DES) ed utilizza chiavi di lunghezza variabile tra i 32 bit e i 448 bit. Nella Figura 2.7(a) vediamo una rappresentazione grafica dell'algoritmo. Un messaggio in chiaro di 64 bit è prima diviso in due metà ognuna con 32 bit, la metà sinistra è messa in XOR con il primo elemento di P (P è un array di 18 interi di 32 bit ognuno) creando così P' processata in F (il funzionamento di F è spiegato nel seguito di questo paragrafo), quindi messo in XOR con la metà destra del messaggio in chiaro che produce un nuovo valore F' . F' rimpiazzerà la metà sinistra originale e P' rimpiazzerà la metà destra del messaggio originale. Questo processo è ripetuto 15 volte con i successivi valori di P ; i risultanti P' e F' sono messi in XOR con gli ultimi due valori di P e ricombinati per produrre il testo cifrato di 64 bit.

➤ Funzione F

La funzione F di Blowfish (Figura 2.7(b)), divide l'input di 32 bit in quattro parti da 8 bit ognuno e li usa come indici delle *S-box*, che vengono sommati e messi in XOR per produrre l'output finale di 32 bit.

La stessa procedura viene utilizzata per la decriptazione con la sola differenza che il testo in input è quello criptato invece che il testo in chiaro.

Proprio come 3DES, Blowfish è vulnerabile all'attacco *Sweet32* (vedere paragrafo 2.3.2 per la spiegazione) poiché utilizza blocchi da 64 bit; quindi perché usare Blowfish invece di optare per altri algoritmi più sicuri? Blowfish è uno dei pochi algoritmi di cifratura che hanno una licenza aperta, ovvero senza la necessità di chiedere alcuna autorizzazione, inoltre è un algoritmo open-source con codice sorgente disponibile per lo studio della sicurezza o per scovare eventuali vulnerabilità. Blowfish è anche uno dei cifrari a blocchi più veloci per uso generale, il che lo rende ideale per una vasta gamma di processori. Di contro Blowfish non fornisce autenticazione e non garantisce l'originalità delle chiavi, quindi è possibile che la stessa chiave venga riutilizzata.

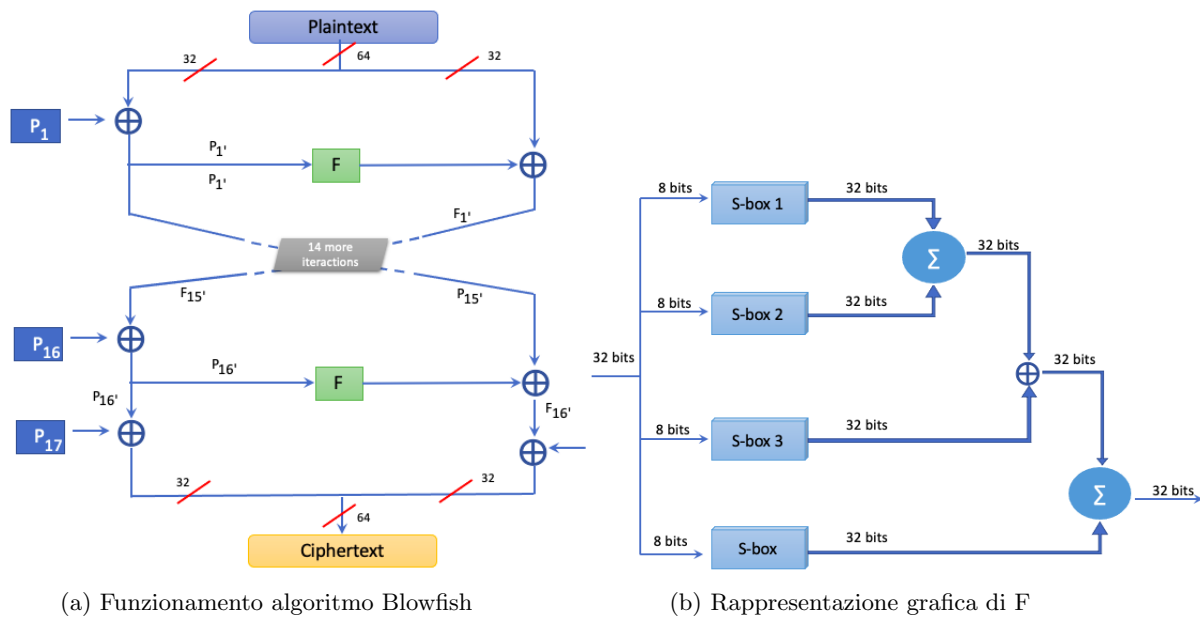


Figura 2.7: Rappresentazione grafica di Blowfish

Nell'Appendice C sono stati messi a confronto gli algoritmi appena descritti avvalendosi di grafici a barre e a torta, sulla base dei dati raccolti in [15].

2.3.4 Md5

L'algoritmo Md5 è la quinta versione dell'algoritmo *Message-Digest* e produce un messaggio *digest* di 128 bit; questa versione è molto più veloce delle versioni precedenti. Md5 produce il messaggio *digest* in cinque passi: padding, aggiunta della lunghezza, inizializzazione del buffer MD, procesamiento del messaggio in blocchi di 512 bit ed infine l'output. Questo algoritmo è stato sviluppato per prendere un input di qualsiasi taglia e produrre un hash in output di 128 bit.

➤ Padding

Il messaggio originale è esteso ("padded") così da avere la lunghezza (in bit) congrua a 448 modulo 512. Per prima cosa viene aggiunto un bit "1" e poi zero o più bit "0" per portare la lunghezza del messaggio fino a 64 bit in meno di un multiplo di 512.

➤ Aggiunta della lunghezza

I 64 bit, di cui sopra, sono aggiunti alla fine del messaggio esteso ("padded") per indicare la lunghezza del messaggio originale in byte. Le regole per l'esecuzione di questo passo sono:

1. La lunghezza in byte del messaggio originale è convertita in binario. Se abbiamo un *overflow*, vengono aggiunti solo i primi 64 bit.
2. Divisione dei blocchi di 64 bit in 2 parole di 32 bit ciascuna.
3. La prima parola è aggiunta per prima e seguita dalla seconda.

➤ Inizializzazione del buffer MD

L'algoritmo Md5 richiede un buffer di 128 bit con uno specifico valore iniziale. Le regole per l'inizializzazione del buffer sono:

- Il buffer è diviso in 4 parole di 32 bit ognuna, chiamate *A*, *B*, *C* e *D*;
- La parola *A* è inizializzata con: 0x67452301;
- La parola *B* è inizializzata con: 0xEFCDAB89;
- La parola *C* è inizializzata con: 0x98BADCFE;
- La parola *D* è inizializzata con: 0x10325476.

➤ Processamento del messaggio in blocchi di 512 bit

Questo è il passo principale dell'algoritmo; per ogni blocco in input, 4 round di operazioni sono processate con 16 operazioni per ogni round. Vengono usate funzioni ausiliarie, chiamate *F*, *G*, *H*, *I*, che prendono in input numeri di 32 bit e producono un output di 32 bit usando gli operatori logici OR, XOR e NOR. Il contenuto dei 4 buffer, è mischiato con i buffer ausiliari e i 16 round sono eseguiti usando 16 operazioni.

➤ Output

Dopo che tutte le operazioni sono state eseguite, i buffer *A*, *B*, *C*, *D*, contengono l'output di Md5 cominciando con i primi bit di *A* e finendo con gli ultimi bit di *D*.

Nella Tabella 2.1 possiamo vedere i vantaggi e gli svantaggi di Md5.

Vantaggi	Svantaggi
È un algoritmo ampiamente utilizzato per gli hash unidirezionali utilizzati senza fornire il valore originale	È incline agli attacchi di <i>collisione</i>
È utilizzato per verificare l'integrità dei file	È piuttosto lento
Lunghezza massima del messaggio illimitata	Hash da 128 bit

Tabella 2.1: Vantaggi e svantaggi Md5

Questo algoritmo è considerato vulnerabile, poiché ha alcune criticità che lo rendono **non** sicuro. Di seguito possiamo vedere vulnerabilità per questo algoritmo:

- Il *Brute force attack* è molto veloce contro l'algoritmo Md5, poiché un hash di 128 bit non è sufficiente per garantire sicurezza contro questo attacco¹⁶.
- È stato così ampiamente utilizzato negli anni che sono state creati enormi database con cui poter decifrare molto velocemente un hash Md5. In Internet si trovano database con oltre 1.150 miliardi di password da poter confrontare per decifrare un hash Md5¹⁶.

¹⁶Appendice C

- L'algoritmo Md5 si è dimostrato anche vulnerabile alle *collisioni*. Una collisione si incontra quando due parole hanno lo stesso hash generato; nel caso di una collisione con un attacco di crittoanalisi si possono decifrare facilmente molti hash.

Per ovviare a queste vulnerabilità (oltre a scegliere altri algoritmi più sicuri), si possono usare alcuni accorgimenti che migliorano la sicurezza: si può utilizzare il “salt” ovvero una parola da inserire prima e/o dopo il testo da cifrare così da avere un hash più difficile da decifrare; per cifrare password, utilizzare password complesse che utilizzino caratteri maiuscoli, minuscoli e caratteri speciali.

2.3.5 Sha1

Secure Hash Algorithm 1 (Sha1) è una funzione di hash crittografica che trasforma una stringa di lunghezza indefinita in una stringa lunga 20 byte esadecimali (160 bit); è stata creata dal NIST nel 1993 e dal 1996 è diventata lo standard mondiale. Questo algoritmo è stato implementato per essere unidirezionale, ovvero una volta ottenuto l'hash non si può ottenere il testo in chiaro di partenza; inoltre è presente l'effetto valanga, ovvero che, un cambio di lettere nel testo in chiaro, diviene un cambio sostanziale nel testo cifrato. In Figura 2.8 possiamo vedere l'effetto valanga in pratica, dove è stata cambiata solamente una lettera, che da maiuscola è diventata minuscola e come l'output risulti completamente diverso. Esistono due metodi per cifrare i messaggi utilizzando SHA1, sebbene uno dei metodi risparmi l'elaborazione di sessantaquattro parole a 32 bit, è più complesso e richiede più tempo per l'esecuzione; per questo di solito si usa il secondo metodo descritto in dettaglio nel seguito.

```
>>> import hashlib
>>> m = hashlib.sha1("Università La Sapienza".encode('utf-8'))
>>> n = hashlib.sha1("università La Sapienza".encode('utf-8'))
>>> m.hexdigest()
'461cee3c8ed50c9971073a404d71f704873dc080'
>>> n.hexdigest()
'91314aa9f658c1017c85c97f49002c69e10e0b5c'
```

Figura 2.8: Effetto valanga in SHA1

Di seguito vediamo i passi nel dettaglio del funzionamento di criptazione di SHA1:

1. Il primo passo è inizializzare cinque stringhe casuali di caratteri esadecimali che serviranno come parte della funzione di hash;
2. Il messaggio viene riempito (“padded”) aggiungendo un “1”, seguito da un numero sufficiente di “0” fino a quando il messaggio raggiunge la dimensione di 448 bit. La lunghezza del messaggio rappresentata da 64 bit, viene quindi aggiunta alla fine, producendo un messaggio lungo 512 bit;
3. L'input M , ottenuto sopra, viene suddiviso in blocchi da 512 bit e ciascun blocco viene ulteriormente suddiviso in sedici parole da 32 bit denominate W_0, \dots, W_{15} ;
4. Per ogni parola, si effettuano 80 iterazioni i e si eseguono i seguenti passi per ciascun blocco:
 - Per le iterazioni da 16 a 79 con $16 \leq i \leq 79$ esegue la seguente operazione:

$$W(i) = S^1(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(I-16)) \quad (2.11)$$

5. Si memorizzano i valori hash definiti nel passaggio 1 nelle seguenti variabili:

- $A = H_0$
- $B = H_1$
- $C = H_2$

- $D = H_3$
- $E = H_4$

6. Per tutte le 80 iterazioni, con $0 \leq i \leq 79$, calcola:

$$TEMP = S^5(A) + f(i, B, C, D) + E + W(i) + K(i) \quad (2.12)$$

Dove $S^n(X)$ è lo spostamento circolare (a sinistra o destra) di X di n posizioni. Nel paragrafo seguente sono mostrati i dettagli sulla funzione logica f .

7. Si registra il risultato dell'hash del blocco sul valore hash complessivo di tutti i blocchi, come mostrato di seguito

- $H_0 = H_0 + A$
- $H_1 = H_1 + B$
- $H_2 = H_2 + C$
- $H_3 = H_3 + D$
- $H_4 = H_4 + E$

8. Il messaggio finale è rappresentato come la stringa di 160 bit composta in OR (\vee), tra i 5 valori di hash:

$$HH = S^{128}(H_0) \vee S^{96}(H_1) \vee S^{64}(H_2) \vee S^{32}(H_3) \vee H_4 \quad (2.13)$$

➤ Funzioni utilizzate nell'algoritmo

In SHA1 viene utilizzata una sequenza di funzioni logiche, a seconda del valore di i :

$$\begin{array}{ll} f(i, B, C, D) = (B \wedge C) \vee ((\neg B) \wedge D) & \text{per } 0 \leq i \leq 19 \\ f(i, B, C, D) = B \oplus C \oplus D & \text{per } 20 \leq i \leq 39 \\ f(i, B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{per } 40 \leq i \leq 59 \\ f(i, B, C, D) = B \oplus C \oplus D & \text{per } 60 \leq i \leq 79 \end{array}$$

Sebbene SHA1 sia ancora ampiamente utilizzato in IPsec, nel 2005 sono state trovate vulnerabilità che hanno compromesso la sua sicurezza. Queste vulnerabilità sono state scoperte da un algoritmo in grado di trovare velocemente collisioni con input diversi, quindi anche SHA1 come Md5 è vulnerabile all'attacco delle collisioni. I primi a trovare collisioni in SHA1 sono stati ricercatori del *CWI Institute in Amsterdam* [14] i quali, hanno presentato due file PDF diversi ma che mostravano lo stesso hash; la collisione trovata ha richiesto un grande quantitativo di risorse per essere realizzata ma è risultata essere molto più veloce di un attacco a forza bruta. Questo attacco (per la mole di risorse computazionali richieste) non è attuabile nella realtà con le risorse di computazione ad oggi disponibili; ma vista l'esponenziale progressione delle capacità di calcolo negli ultimi anni, SHA1 verrà rimpiazzato, entro il 2023, da funzioni più sicure come SHA256 e SHA3.

3. Attacchi verso IPsec

In questo capitolo vengono descritti degli attacchi verso IPsec, che possono essere sfruttati per entrare in possesso di dati ed informazioni criptati nel tunnel creato; questi attacchi sfruttano vulnerabilità delle configurazioni di IPsec indipendentemente dagli algoritmi/protocolli usati (vulnerabilità degli algoritmi/protocolli già descritti nel Capitolo 2).

3.1 Attacco Man-in-the-Middle in IPsec

Man-in-the-Middle (MITM) è un tipo di attacco che consente agli attaccanti di intercettare le comunicazioni tra due o più host con lo scopo di rubare, spiare o modificare i dati che si stanno scambiando.

IPsec usa chiavi per l'autenticazione degli host nel tunnel; un malintenzionato potrebbe recuperare una PSK debole, decifrarla e violare la sicurezza del tunnel. Questo attacco prende di mira l'handshake in IKE, il protocollo preposto per lo scambio di chiavi, usato nella modalità "aggressiva". In questa modalità la PSK e altre informazioni sono trasmesse in un hash non criptato e possono essere intercettate e deciptate da terzi; inoltre anche gli id e gli IP degli utenti sono scambiati in chiaro ed è quindi, possibile capire quali host e con quale IP sono presenti nel tunnel. Nel seguito vediamo i passi con cui è possibile attuare questo attacco¹⁷.

1. La prima informazione che un'attaccante deve recuperare è l'IP del gateway; questo può essere attuato in diversi modi come per esempio uno scan di rete sulla porta 500 (UDP);
2. Con l'indirizzo IP del gateway è possibile effettuare degli scan più approfonditi e scoprire se il gateway è configurato in modalità "aggressiva";
3. Se il gateway sta utilizzando la modalità "aggressiva", si possono ricavare le informazioni descritte sopra come l'IP o la PSK; è possibile intercettare anche il traffico ISAKMP inviato da un client al gateway;
4. A questo punto si procede a catturare la risposta dal gateway durante un tentativo di connessione, nel quale, l'attaccante deve essere in possesso di un indirizzo IP che il gateway accetta così da poter stabilire la connessione e ricevere una risposta; questo non è sempre necessario poiché, in scenari reali può capitare che i gateway sono configurati per accettare le connessioni da ogni dispositivo che ne fa richiesta;
5. Una volta che la PSK è stata ottenuta è possibile inserirla per aver accesso alla connessione.

Con la PSK un'attaccante può intercettare il traffico tra il client e il server VPN; una volta fatto ciò lo può alterare oppure impersonare il client e concludere la fase 2 con il server, con conseguente accesso alle risorse interne.

Per cercare di limitare questo tipo di attacco possono essere effettuati alcuni passi:

- Disabilitare IKE in aggressive mode;
- Preferire l'autenticazione con i certificati, invece che l'autenticazione con Pre-Shared Key;
- Se si usa una PSK, che sia una password complessa e non facilmente decifrabile;

¹⁷Per la spiegazione teorica dell'attacco si rimanda al paragrafo 2.1.

- Limitare gli accessi ai sistemi e servizi che sono realmente richiesti utilizzando dei firewall dietro al server VPN, così da minimizzare gli accessi non autorizzati.

Tutti i tunnel creati con IKE in aggressive mode possono essere vulnerabili a questo attacco indipendentemente dagli algoritmi di cifratura o autenticazione usati, sfruttando configurazioni sbagliate da parte degli utenti che creano il tunnel.

■ Attacco alla PSK in pratica

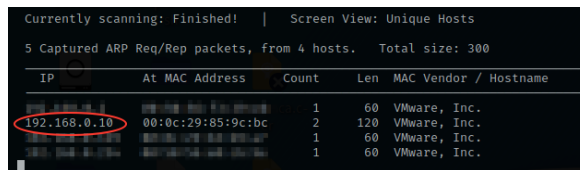
Di seguito vediamo come un attacco ad IKE in modalità “aggressive” permetta di recuperare la PSK di un tunnel VPN. Per sviluppare questo attacco sono stati utilizzati alcuni tool open-source come “ike-scan” e “psk-crack” inoltre è stato utilizzato un server VPN in una macchina virtuale con indirizzo IP: 192.168.0.10/24, la macchina attaccante utilizza Linux come sistema operativo.

Il primo passo è quello di effettuare una scansione sulla rete (Figura 3.1(a)) per trovare gli host presenti e cercare l’indirizzo IP del server (di cui già si era a conoscenza, effettuato per completezza) con il comando:

```
# netdiscover -r 192.168.0.0/24
```

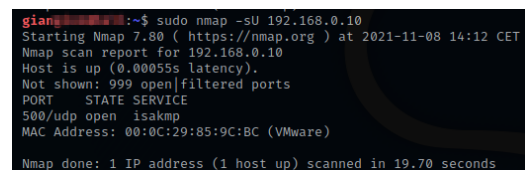
dove *netdiscover* è un ARP scanner usato per cercare gli host attivi su una determinata rete determinata dall’opzione *-r*. Trovato l’indirizzo IP del server, si effettua un port scanner sulla porta 500 (UDP, utilizzata da IKE) per cercare i servizi in esecuzione, confermando che sulla porta 500 è in esecuzione IKE (Figura 3.1(b)); questo risultato è stato ottenuto con il comando:

```
# nmap -sU 192.168.0.10
```



IP	At	MAC Address	Count	Len	MAC Vendor / Hostname
192.168.0.10	00:0c:29:85:9c:bc	2	120	VMware, Inc.	
192.168.0.10	00:0c:29:85:9c:bc	1	60	VMware, Inc.	
192.168.0.10	00:0c:29:85:9c:bc	1	60	VMware, Inc.	

(a) Host presenti sulla rete



```
Starting Nmap 7.80 ( https://nmap.org ) at 2021-11-08 14:12 CET
Nmap scan report for 192.168.0.10
Host is up (0.00055s latency).
Not shown: 999 open|filtered ports
PORT      STATE SERVICE
500/udp   open  isakmp
MAC Address: 00:0C:29:85:9C:BC (VMware)
Nmap done: 1 IP address (1 host up) scanned in 19.70 seconds
```

(b) Port scanner sulla porta 500

Figura 3.1: Ricerca e scansione del server

Confermato che sulla porta 500 è in esecuzione IKE, possiamo sfruttare il tool “ike-scan” per scoprire alcune informazioni importanti sul server in questione come per esempio gli algoritmi di crittografia usati, gli utenti attivi e la PSK. Troviamo l’hash della PSK (Figura 3.2(a)) con il seguente comando:

```
# ike-scan -M -A -Pike-hash -d 500 192.168.0.10
```

dove l’opzione *-A* è per la modalità “aggressive”, *-P* indica il file dove scrivere l’hash recuperato in output e *-d* è la porta di destinazione.

Trovato l’hash della PSK, usiamo il tool “psk-crack” per cercare di decriptarla usando una wordlist; con questo tool è possibile anche provare un attacco brute force per cercare di decriptare la PSK provando tutte le combinazioni di password possibili, ma il fine di questo caso pratico è quello di mostrare quanto questo attacco sia efficace nel recuperare una PSK di un server con attiva la modalità “aggressive” (si sconsiglia dunque la modalità “aggressive”, se si ha necessità di usare questa modalità la scelta della PSK deve essere una password complessa e non comune). La decifratura dell’hash della PSK (Figura 3.2(b)), è stata ottenuta con il comando seguente:

```
# psk-crack -d /usr/share/wordlists/rockyou.txt ike-hash
```

```

(a) ike-scan per trovare hash
Starting ike-scan 1.9.4 with 1 hosts (http://www.nta-monitor.com/tools/ike-scan/)
192.168.0.10 Aggressive Mode Handshake returned
HDR=(CKY-R=800848d01883e0b5)
SA=(Enc=3DES Hash=SHA1 Auth=PSK Group=2:modp1024 LifeType=Seconds LifeDuration(4)=0x00007080)
KeyExchange(128 bytes)
Nonce(16 bytes)
ID(Type=ID_IPV4_ADDR, Value=192.168.0.10)
Hash(20 bytes)
VID=afcad71368a1fc96b8696fc77570100 (Dead Peer Detection v1)
Ending ike-scan 1.9.4: 1 hosts scanned in 0.017 seconds (58.06 hosts/sec); 0 returned notify

(b) Psk-crack per password in chiaro
Starting psk-crack [ike-scan 1.9.4] (http://www.nta-monitor.com/tools/ike-scan/)
Running in dictionary cracking mode
key 423456 matches SHA1 hash a95c0126b041b7cf8000b11b3fd84ca0c6532d7d
Ending psk-crack: 1 iterations in 0.006 seconds (156.35 iterations/sec)

```

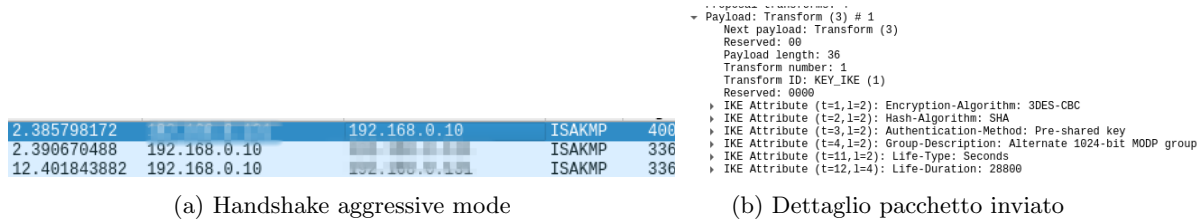
(a) Ike-scan per trovare hash

(b) Psk-crack per password in chiaro

Figura 3.2: Hash e password in chiaro

In alternativa al tool “psk-crack” avremmo potuto utilizzare qualsiasi altro tool per la decrittazione di hash, tra i più gettonati si citano *Hashcat* e *John the Ripper*. Con questa procedura è possibile decriptare l’hash della PSK in modalità offline e quindi non impattare sui log del tunnel VPN.

Una volta trovata la PSK, può essere utilizzata per stabilire una connessione VPN con il server con conseguente accesso a tutte le risorse che il tunnel proteggeva. Nella Figura 3.3(a) vediamo l’handshake della modalità “aggressive” con i tre messaggi scambiati e nella Figura 3.3(b) vediamo dettagli implementativi di IKE, come gli algoritmi di cifratura usati per la costruzione del tunnel.



(a) Handshake aggressive mode

(b) Dettaglio pacchetto inviato

Figura 3.3: Sniffing di pacchetti con Wireshark

3.2 Attacco Replay

In questo paragrafo viene descritto cos’è il Replay attack e come IPsec lo previene.

Il Replay attack è un tipo di MITM attack, dove un’attaccante intercetta una comunicazione in transito su una rete e ritrasmette i dati intercettati nel tentativo di prendere il controllo del sistema. Viene effettuato in tre passi:

1. In primo luogo, si intercettano i flussi di comunicazione di rete per ottenere informazioni sui tipi di attività in corso;
2. successivamente, si intercettano le informazioni dell’utente legittimo;
3. infine, l’attaccante riproduce o reinvia in modo massiccio le informazioni raccolte per indirizzare erroneamente il destinatario e ingannarlo facendo credere che sia un utente autentico.

Per effettuare questo attacco non è necessario che l'attaccante decrypti le informazioni intercettate, ma è sufficiente reinviarle così come sono; quindi la cifratura non basta a prevenire questo attacco.

IPsec protegge la rete dai replay attack utilizzando uno “sliding window mechanism”¹⁸ chiamata anti-replay window. Questa funzione controlla il numero di sequenza di ciascun pacchetto IPsec ricevuto, rispetto all'insieme dei numeri di sequenza correnti. Se il numero di sequenza ricevuto non è nell'insieme dei numeri correnti, il pacchetto è considerato ripetuto e viene scartato; sia AH che ESP hanno un meccanismo integrato di anti-replay. La anti-replay window è una “finestra” (di solito di 64 bit) che scorre lungo il flusso di dati in arrivo e i bit corrispondono al numero di sequenza dei pacchetti; quindi ogni pacchetto in arrivo viene autenticato, se presente all'interno dell'insieme della finestra corrente viene esaminato ponendo contemporaneamente il relativo bit a 0, se è a sinistra della finestra corrente il pacchetto viene scartato ed infine se è a destra fa scorrere la finestra a destra del numero di celle che superano il limite destro della finestra stessa (Figura 3.4).

In alcune situazioni, però può capitare che i pacchetti di servizio sono ricevuti in ordine differente da quello originale e l'anti-replay può considerarli ripetuti; per ovviare a ciò, il meccanismo anti-replay può essere disattivato (rendendo vulnerabile il tunnel).

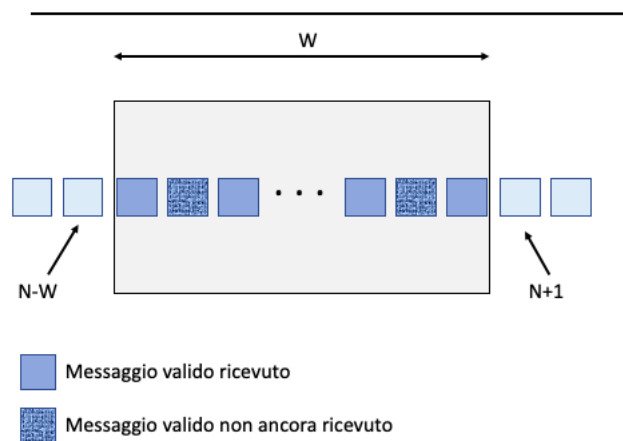


Figura 3.4: Sliding window mechanism

3.3 Attacco LogJam

Questo attacco sfrutta delle vulnerabilità nello scambio di chiavi con Diffie-Hellman, usate per stabilire le chiavi della sessione in IPsec.

➤ Breve introduzione allo scambio di chiavi con Diffie-Hellman

Lo scambio di chiavi Diffie-Hellman (DH) è usato per scambiare chiavi in modo sicuro su un canale **non** sicuro. Nel seguito chiameremo *A* e *B* i due interlocutori che si vogliono scambiare le chiavi.

¹⁸Lo sliding window mechanism è un protocollo a livello dati per la consegna affidabile e sequenziale di frame di dati.

1. I due interlocutori scelgono due numeri primi g e p randomicamente;
2. A e B generano la loro chiave privata con valori random che chiameremo sk_a e sk_b rispettivamente;
3. A genera la sua chiave pubblica pk_a dalla sua chiave segreta sk_a con la seguente formula:

$$pk_a = g^{sk_a} \bmod p \quad (3.1)$$

4. B genera la sua chiave pubblica pk_b dalla sua chiave segreta sk_b con la seguente formula:

$$pk_b = g^{sk_b} \bmod p \quad (3.2)$$

5. A invia la sua chiave pubblica a B e viceversa;
6. A genera la chiave segreta con:

$$sk = pk_b^{sk_a} \bmod p \quad (3.3)$$

7. B genera la chiave segreta con:

$$sk = pk_a^{sk_b} \bmod p \quad (3.4)$$

L'attacco prevede l'acquisizione del valore g^{sk} da cui è possibile calcolare la chiave segreta sk ; il problema di trovare questo valore è noto come il problema del *calcolo del logaritmo discreto*¹⁹; matematicamente intrattabile quando p è di dimensione maggiori o uguali a 2048 bit, ma calcolabile facilmente quando p è piccolo come, per esempio, di 512 bit. Il calcolo di $g^a \bmod p$ con p di 512 bit, diviene computazionalmente possibile in tempi brevi. Quindi la difficoltà principale di questo attacco è quello di calcolare il logaritmo discreto di 512 bit prima che il client e il server terminino il loro scambio; il modo più veloce di fare ciò è usare un algoritmo chiamato *Number Field Sieve* (NFS), il quale impiega qualche anno con un solo *core* ma una settimana con poche centinaia di *core*. Nella pratica questo ancora non basta, poichè bisogna calcolarlo prima che il client e server finiscano il loro scambio; così NFS viene diviso in due diverse fasi:

- La prima è quella del pre-calcolo (per un numero primo p). Questa fase include il processo di selezione polinomiale che dipende da p . L'output di questa fase è una tabella da utilizzare nella seconda fase;
- La seconda calcola a (dato $g^a \bmod p$). Questa fase utilizza la tabella creata nella fase del pre-calcolo.

Così facendo si riesce a calcolare il logaritmo discreto di un numero con 512 bit in qualche minuto, scoprendo la chiave privata ed invalidando la sicurezza dello scambio con Diffie-Hellman.

Quindi la soluzione (per ora), è quella di usare Diffie-Hellman con almeno 2048 bit per lo scambio di chiavi in IPsec.

¹⁹Il problema del calcolo del logaritmo discreto è molto simile a quello della fattorizzazione dei numeri primi, in quanto entrambi sono considerati difficili (non esistono algoritmi che li risolvono in tempo polinomiale).

3.4 Attacco Denial of Service in IPsec

Il Denial of Service (DOS) è un attacco che consiste nell'inviare molteplici richieste ad un dispositivo target (host, gateway, sito, ecc.) fino a che il dispositivo in questione non risulti saturo e non riesca più a rispondere ad alcuna richiesta, rendendo il dispositivo irraggiungibile. Esiste anche un'altra versione, che è il Distributed Denial of Service (DDOS) dove le false richieste arrivano allo stesso momento da più fonti. Ci sono due tipi di DOS, uno che consiste nell'inviare pacchetti mal formati nella speranza di causare mal funzionamenti del sistema e l'altro che consiste nell'invio di pacchetti formati come gli originali da sembrare tali. In IPsec si userà il secondo tipo di DOS.

Assumiamo che due dispositivi comunichino attraverso due gateway *A* e *B* usando solo ESP con solo crittografia senza autenticazione. Un'attaccante può intercettare un datagramma legittimo, modificare l'indirizzo sorgente (tramite le vulnerabilità descritte nel paragrafo 2.2) e reiniettarlo nella rete. Il gateway *B* accetterà il datagramma modificato, estrarrà l'IP header, il payload e lo invierà a destinazione. Una volta fatto ciò, per ogni datagramma legittimo, l'attaccante può generare 2^{32} falsi source address e quindi lo stesso numero di datagrammi i quali avranno lo stesso contenuto di un datagramma legittimo. Tutti verranno accettati dal gateway *B* e inviati a destinazione; in questo modo sia il gateway che il dispositivo di destinazione saranno sovraccaricati di datagrammi non validi e resi irraggiungibili. Questo ribadisce che ESP solo con crittografia è vulnerabile ad una grande varietà di attacchi e quindi ne è scongiato l'uso.

Con lo sviluppo della versione 2 di IKE (IKEv2), questo attacco verso IPsec è stato parzialmente eliminato, ma comunque attuabile. Per prevenire questo attacco bisogna limitare il traffico IKE/ISAKMP, permettendolo solo dai siti o dispositivi conosciuti. Di seguito vediamo alcuni passi per limitare ulteriormente l'attuazione di questo attacco:

- Quando viene ricevuta una richiesta *IKE_SA_INIT*, è possibile avviare un timer. Se il tempo scade prima che venga ricevuto un messaggio *IKE_AUTH*, la *IKE_SA* dovrebbe essere cancellata;
- Se la decrittografia *IKE_AUTH* fallisce più volte consecutivamente durante la creazione della sessione, la *IKE_SA* deve essere cancellata;
- Definire la dimensione massima del certificato per *IKE_SA*, che impedirà il download di certificati di grandi dimensioni;
- Se i messaggi IKE in entrata per *IKE_SA* superano un certo limite, tutti i messaggi oltre il limite devono essere eliminati.

3.5 Attacco Buffer Overflow

Un altro attacco verso IPsec è quello del *Buffer Overflow*; il buffer non è altro che uno spazio di archiviazione temporaneo. Il Buffer Overflow è la sovrascrittura di spazi di memoria e l'immissione nel buffer di dati oltre la sua capacità; questi possono essere salvati ovunque nel buffer e quindi sovrascrivere spazi di memoria già allocati, permettendo l'aggiunta di codice dannoso per "saturare" il buffer. Questo può comportare: arresti anomali del sistema, perdita del controllo dell'accesso ed esecuzione di codice arbitrario.

Un Buffer Overflow, di solito, è causato da errori nel codice sorgente che permette di allocare grandi quantità di memoria nel buffer; per ovviare a questo problema si può inserire un controllo dei limiti che impedisca il sovraccarico del buffer controllando che i dati scritti siano entro limiti accettabili.

Per sfruttare un Buffer Overflow, un'attaccante deve inviare pacchetti UDP al servizio ISAKMP (porta UDP/500); i pacchetti UDP inviati al buffer, il quale non riuscirà più a gestire le sovra-scritture, permetterebbero di impersonare un peer legittimo o un dispositivo ed inoltre consentirebbero di eseguire codice arbitrario e ottenere il pieno controllo del server, poiché non sarà richiesta alcuna forma di autenticazione. Nel 2016 è stato rilevato che un'ampia gamma di router e firewall di *Cisco* risultano vulnerabili a questo attacco [CVE-2016-1287]²⁰ e l'azienda ha dovuto rilasciare aggiornamenti del software per riparare questa vulnerabilità.

Tutti i router e i firewall potrebbero essere vulnerabili a questo attacco, infatti è sufficiente del codice sorgente errato; per prevenire questo, vanno effettuati controlli sul codice, aggiornati i software ed effettuati test specifici per far sì che un dispositivo non sia vulnerabile.

²⁰Nel presente link viene spiegato nel dettaglio come questa vulnerabilità può essere sfruttata: <https://research.nccgroup.com/2017/06/15/a-warcon-2017-presentation-cisco-asa-exploiting-the-ikev1-heap-overflow-cve-2016-1287/>.

4. Conclusioni

Nei capitoli precedenti, dopo un'introduzione ai protocolli sono state descritte vulnerabilità note ed attacchi ad IPsec; in questo capitolo si vedranno le considerazioni finali e le implementazioni future.

Nella presente relazione è stato possibile appurare di come IPsec tutt'oggi è ancora uno dei metodi più sicuri per creare un tunnel tra due dispositivi comunicanti nella rete, anche se non privo di vulnerabilità. È ancora difficile definire un elevato livello di protezione e allo stesso tempo la validità. È stato approfondito il tema delle vulnerabilità e i diversi metodi di attacco verso IPsec per appropriarsi di dati criptati. Per la scrittura/sviluppo di questa tesi è stato effettuata un'implementazione di un tunnel di IPsec verso un server virtuale (VPS). Per fare ciò è stato utilizzato un server con Ubuntu 18.04 LTS e un client con Debian 10 (Figura 4.1(a) e 4.1(b)); dopo numerosi tentativi ed errori di configurazione è stato finalmente possibile creare il tunnel infatti uno degli svantaggi dell'utilizzo di IPsec è proprio la difficoltà di configurazione. Si è appurato inoltre che creare un tunnel con IPsec richiede un grande sforzo in termini di risorse di calcolo poichè i dati devono essere crittografati e decifrati costantemente, il che porta facilmente ad un overhead della CPU. Altri svantaggi nell'utilizzo di IPsec possono essere: la compatibilità data dal fatto che non si ha uno standard per la configurazione e ciò crea problemi tra i dispositivi; l'ampia gamma di accessi forniti da IPsec che aumentano la possibilità di concedere privilegi ad altri dispositivi sulla rete; ed infine le restrizioni nei firewall che potrebbero negare la connessione, per esempio molti firewall in uso ad oggi bloccano per default la porta 500 (UDP) utilizzata da IKE.

Durante la scrittura di questa relazione studiando le vulnerabilità presenti nei protocolli/algoritmi utilizzati in IPsec, gli svantaggi nell'uso di IPsec descritti sopra e lo sforzo in termini di ore per creare un tunnel, mi chiedevo sempre più spesso se non ci fossero metodi più sicuri e "facili" per la creazione di un tunnel VPN; ma andando sempre più in dettaglio di come IPsec e i suoi componenti funzionassero è stato facile capire del perché, ad oggi, sia ancora uno dei migliori metodi per creare un tunnel. La sicurezza della rete che fornisce (se configurato correttamente), l'indipendenza dalle applicazioni poichè implementato a livello di rete, la sicurezza dei dati scambiati e la varietà dei metodi di autenticazione e cifratura forniti, rendono IPsec un'ottima soluzione per costruire un tunnel su qualsiasi tipo di rete che sia WAN o LAN.

```
gi:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    link/ether 00:0c:29:1d:a7:be brd ff:ff:ff:ff:ff:ff
    inet 102.27.1.1/24 scope global dynamic eth0
        valid_lft 1591sec preferred_lft 1591sec
    inet6 fe80::208c:291d:a7be:ff:fe scope link
        valid_lft forever preferred_lft forever
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
    link/noarp 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 102.27.1.2/24 scope global tun0
        valid_lft forever preferred_lft forever
    inet6 fe80::208c:291d:a7be:ff:fe scope link stable-privacy
        valid_lft forever preferred_lft forever
```

(a) Connessione al tunnel

```
gi:~$ ping 10.0.9.7
PING 10.0.9.7 (10.0.9.7) 56(84) bytes of data.
64 bytes from 10.0.9.7: icmp_seq=1 ttl=64 time=28.7 ms
64 bytes from 10.0.9.7: icmp_seq=2 ttl=64 time=29.5 ms
64 bytes from 10.0.9.7: icmp_seq=3 ttl=64 time=30.0 ms
64 bytes from 10.0.9.7: icmp_seq=4 ttl=64 time=28.3 ms
64 bytes from 10.0.9.7: icmp_seq=5 ttl=64 time=29.3 ms
64 bytes from 10.0.9.7: icmp_seq=6 ttl=64 time=28.6 ms
64 bytes from 10.0.9.7: icmp_seq=7 ttl=64 time=29.2 ms
^C
--- 10.0.9.7 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6016ms
rtt min/avg/max/mdev = 28.268/29.082/29.980/0.554 ms
gi:~$ ping 10.0.9.7
PING 10.0.9.7 (10.0.9.7) 56(84) bytes of data.
^C
--- 10.0.9.7 ping statistics ---
117 packets transmitted, 0 received, 100% packet loss, time 118770ms
```

(b) Ping con e senza tunnel

Figura 4.1: Tunnel IPsec con server virtuale

4.1 Implementazioni future

Le implementazioni di IPsec e gli algoritmi utilizzati da esso, sono in costante sviluppo e progresso; all'inizio del nuovo millennio si aveva la sicurezza che sia MD5 sia SHA1 fossero inviolabili, ma a solo 20 anni di distanza ciò non è più vero (come descritto nei precedenti capitoli) ed ormai sono diventati obsoleti. Per questo motivo negli ultimi anni, per la creazione di tunnel IPsec, è sempre più uso comune utilizzare il più sicuro SHA3 creato dal NIST nel 2015. Ad oggi SHA3 non è ancora uno standard per la creazione di tunnel IPsec e su molti dispositivi ancora non è supportato, ma la velocità di implementazione di nuove tecnologie e le vulnerabilità in MD5 e SHA1 porteranno SHA3, in futuro, ad esserlo. Inoltre lo stack IPv6 non è ancora integrato con l'implementazione IPsec.

Con l'avvenire dei computer quantistici, la presenza degli algoritmi attualmente esistenti, diverrebbero inefficaci, per cui si pensa allo sviluppo di nuovi algoritmi da implementare al fine di garantire sicurezza. Lo scambio di chiavi con DH diverrebbe vulnerabile per la facilità con cui si calcolerebbe il logaritmo discreto; quindi si dovrà utilizzare uno scambio di chiavi che resista a tali calcoli. Le proposte attuali per tali algoritmi richiedono tutti l'utilizzo di chiavi pubbliche di grandi dimensioni che devono essere scambiate in IKE durante la fase IKE_SA_INIT. Ma durante questa fase dello scambio la frammentazione di IKEv2 non può ancora essere utilizzata poiché non è stato ancora creato un canale sicuro in grado di identificare i frammenti legittimi; per ovviare a questo problema verrà inserito uno scambio intermedio tra IKE_SA_INIT e IKE_AUTH che possa supportare la frammentazione. Questo lavoro è descritto in [16].

Una volta che tali algoritmi saranno implementati, anche il protocollo IKEv2 dovrà essere modificato per supportarli; si sta studiando, un'implementazione di IKEv2 dove si manterrà lo scambio di chiavi DH esistente e si aggiungerà uno o più scambi di chiavi “quantum-safe”, così da avere maggiore sicurezza nel caso uno di questi divenga vulnerabile; questo studio è descritto in [17]. Questo scambio di chiavi è chiamato *Quantum Key Distribution* (QKD) ed è uno scambio basato su delle leggi quantistiche della fisica piuttosto che sulla complessità computazionale.

Lo sviluppo e implementazione di computer quantistici porterà anche altri problemi alle implementazioni di IPsec correnti, uno su tutti (oltre quelli descritti sopra) è la decifrazione di RSA in tempi molto brevi. La fattorizzazione di interi in numeri primi ha una complessità computazionale esponenziale, ma tramite l'algoritmo di Shor²¹, implementato in un computer quantistico, questo diverrà polinomiale; è stato ipotizzato che un computer quantistico con 4099 qubit perfettamente stabili, riuscirebbe a violare la crittografia RSA-2048 in neanche un minuto. Ancora non è stata trovata una soluzione a questo problema ed il NIST ha lanciato un progetto chiamato “Post-Quantum Cryptography”, destinato a trovare implementazioni di algoritmi “quantum-safe” che prendano il posto di RSA entro l'anno 2029, quando si presuppone che i computer quantistici diverranno “realtà”.

²¹L'algoritmo di Shor è stato sviluppato da Peter Shor nel 1994, per risolvere il problema della fattorizzazione di numeri interi in numeri primi.

5. Ringraziamenti

Per la scrittura di questa relazione sono tante le persone che vorrei ringraziare, che direttamente o indirettamente hanno contribuito alla scrittura di essa. In primo luogo ringrazio il Professore **Francesco Parisi-Presicce** che mi ha dato l'opportunità di svolgere un tirocinio su un argomento che ho trovato molto interessante, il quale mi ha dato molti spunti di riflessione e che vorrei approfondire nel futuro. Un ringraziamento particolare va ai miei genitori *Maurizio* e *Cinzia* che mi hanno permesso di arrivare fino alla fine di questo percorso, a mio fratello *Alessio* e a mia sorella *Martina* ed ai miei cognati *Claudia* e *Andrea*; ai miei nonni *Anna* e *Ferruccio* e ai miei zii *Sabrina* e *Massimo* e *Tiziana* e *Matteo*, i quali ognuno a modo suo ha contribuito e aiutato a farmi arrivare fino a qui.

Ringrazio anche la famiglia *Sbarra*, *Simone* e *Veronica* con la quale ho condiviso tanto negli anni; a quest'ultima va un'ulteriore ringraziamento per aver portato nelle nostre vite la piccola *Gresia* che ha fatto riscoprire a tutti noi cosa vuol dire tornare bambini, anche solo per il tempo di un balletto. Non posso non ringraziare i miei fratelli de l'*Allegra Brigata* con i quali nel bene e nel male ho condiviso tutto ed insieme a *Camilla*, *Federica* e *Flaminia* hanno reso la mia adolescenza fantastica e costruito momenti che rimarranno per sempre dentro di me, con la speranza che ce ne siano molti altri. Ringrazio *Daniele*, amico da una vita e compagno di tante "avventure" che hanno reso meno difficili alcuni periodi della mia vita e *Chiara* che da qualche anno sopporta le nostre storie sempre con il sorriso portando felicità e spensieratezza a chiunque le sia vicino.

Inoltre ringrazio chiunque abbia passato, negli ultimi anni, anche solo una serata con me poiché anche loro hanno contribuito a questo risultato.

Infine ringrazio *Melissa*, con la quale ho condiviso gioie e dolori di questo percorso e non solo; la ringrazio inoltre per tutto quello che mi ha trasmesso ed insegnato in questi anni, soprattutto per avermi spronato ad andare avanti anche nei momenti peggiori con la consapevolezza che senza di lei tutto questo non sarebbe mai stato possibile; sei e sarai sempre un pezzo fondamentale della mia vita, quindi grazie di tutto.

6. Bibliografia

- [1] **Security Architecture for the Internet Protocol.**
Stephen Kent; Randall Atkinson (November 1998).
<https://datatracker.ietf.org/doc/html/rfc2401>
- [2] **IP Authentication Header**
Stephen Kent; Randall Atkinson (November 1998).
<https://datatracker.ietf.org/doc/html/rfc2402>
- [3] **IP Encapsulating Security Payload**
Stephen Kent; Randall Atkinson (November 1998).
<https://datatracker.ietf.org/doc/html/rfc2406>
- [4] **RFC Editor**
<https://www.rfc-editor.org/>
- [5] **The Internet Key Exchange**
Dan Harkins; Dave Carrel (November 1998).
<https://datatracker.ietf.org/doc/html/rfc2409>
- [6] **Diffie-Hellman Key Agreement Method**
Eric Rescoria (June 1999).
<https://datatracker.ietf.org/doc/html/rfc2631>
- [7] **The Dangers of Key Reuse: Practical Attacks on IPsec IKE**
Dennis Felsch; Martin Grothe; Jörg Schwenk; Adam Czubak; Marcin Szymanek (August 2018)
SEC'18: Proceedings of the 27th USENIX Conference on Security Symposium
- [8] **Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1**
Daniel Bleichenbacher (August 1988)
CRYPTO '98: 18th Annual International Cryptology Conference
- [9] **Attacking the IPsec Standards in Encryption-only Configurations**
J. P. Degabriele; Kenneth G. Paterson (June 2007).
Conference: Security and Privacy, 2007. SP '07.
- [10] **Data Encryption Standard (DES)**
National Institute of Standards and Technology (December 1993).
<https://web.archive.org/web/20040410171758/http://www.itl.nist.gov/fipspubs/fip46-2.htm>
- [11] **Communication theory of secrecy systems**
C. E. Shannon (October 1949).
The Bell System Technical Journal, vol. 28, no. 4, pp. 656-715
- [12] **On the Practical (In-)Security of 64-bit Block Ciphers**
Karthikeyan Bhargavan; Gaëtan Leurent (August 2016)
https://sweet32.info/SWEET32_CCS16.pdf

- [13] **Another Birthday Attack**
Don Coppersmith (August 1985).
5th Crypto 1985. 14-17
- [14] **The first collision for full SHA-1**
Marc Stevens; Elie Bursztein; Pierre Karpman; Ange Albertini; Yarik Markov (February 2015).
<https://shattered.it/static/shattered.pdf>
- [15] **A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish**
Priyadarshini Patila; Prashant Narayankarb; Narayan D. G.; Meena S. M.; (December 2015)
International Conference on Information Security & Privacy
- [16] **Intermediate Exchange in the IKEv2 Protocol**
Smyslov V; (August 2021)
<https://datatracker.ietf.org/doc/draft-ietf-ipsecme-ikev2-intermediate/>
- [17] **Multiple Key Exchanges in IKEv2**
Tjhai C; Tomlinson M; Bartlett G; Flurher S; Van Geest D; Garcia-Morchon O; Smyslov V
(September 2021)
<https://datatracker.ietf.org/doc/draft-ietf-ipsecme-ikev2-multiple-ke/>

A. Creazione tunnel VPN

Di seguito vediamo un caso pratico di VPN con IPsec implementato con GNS3, un software open source per la simulazione di reti.

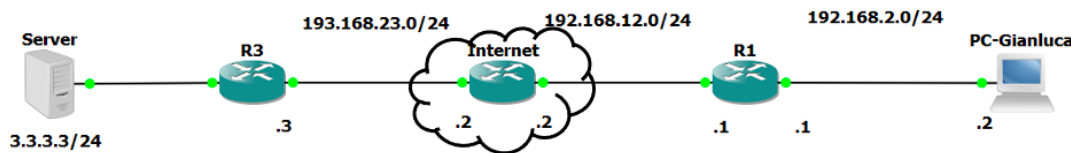


Figura A.1: Topologia iniziale

Nella Figura A.1 vediamo la topologia iniziale costituita da un client, un server, due router collegati in Internet i quali instraderanno i pacchetti inviati dal client al server e viceversa. Nella Figura A.2 possiamo vedere che inizialmente non si ricevono risposte dal server né dal client (Figura A.2(a)), né dal router (Figura A.2(b)) poiché situati in reti diverse. Per riuscire a far comunicare il client e il server è stato implementato un tunnel IPsec creando così una VPN tra il router *R1* e il router *R3*. Dopo aver implementato il tunnel riceviamo correttamente risposte dal server (Figura A.3).

```
C:\Users\...>ping 3.3.3.3 -t
Pinging 3.3.3.3 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

(a)

```
[admin@R1] > ping 3.3.3.3
SEQ HOST                                SIZE TTL TIME STATUS
0 192.168.12.2                          84 64 0ms net unreachable
1 192.168.12.2                          84 64 1ms net unreachable
2 192.168.12.2                          84 64 0ms net unreachable
3 192.168.12.2                          84 64 2ms net unreachable
4 192.168.12.2                          84 64 2ms net unreachable
5 192.168.12.2                          84 64 1ms net unreachable
6 192.168.12.2                          84 64 1ms net unreachable
sent=7 received=0 packet-loss=100%
```

(b)

Figura A.2: Risposta prima del tunnel IPsec

```
C:\Users\...>ping 3.3.3.3 -t
Pinging 3.3.3.3 with 32 bytes of data:
Reply from 3.3.3.3: bytes=32 time=3ms TTL=63
Reply from 3.3.3.3: bytes=32 time=10ms TTL=63
Reply from 3.3.3.3: bytes=32 time=5ms TTL=63
Reply from 3.3.3.3: bytes=32 time=12ms TTL=63
Reply from 3.3.3.3: bytes=32 time=7ms TTL=63
```

Figura A.3: Risposta dopo la creazione del tunnel IPsec

B. Decifratura Md5

```
1 827ccb0eea8a706c4c34a16891f84e7b
2 e10adc3949ba59abbe56e057f20f883e
3 25f9e794323b453885f5181f1b624d0b
4 5a105e8b9d40e1329780d62ea2265d8a
5 5f4dcc3b5aa765d61d8327deb882cf99
6 25d55ad283aa400af464c76d713c07ad
7 18aa20d71f1624aac4c14fe5cf972d56
8 5b217538ef3e4199db6d3c3889848344
9 912ec803b2ce49e4a541068d495ab570
10 d8578edf8458ce06fbc5bb76a58c5ca4
...
90 2ee2abccdda7424657a2fc26873f5a5ed
91 b3b7eca0709c3c25e677677937e8f609
92 6fb42da0e32e07b61c9f0251fe627a9c
93 01cfdc4f6b8770febfb40cb906715822
94 608f0b988db4a96066af7dd8870de96c
95 dc647eb65e6711e155375218212b3964
96 263bce650e68ab4e23f28263760b9fa5
97 67881381dbc68d4761230131ae0008f7
98 061fba5bdfc076bb7362616668de87c8
99 6988ec3aba1eaddf2435141bf10487ca
100 1b53230416980fce41adbac1cba86956
```

Figura B.1: Hash di 100 password

tempo totale per decriptare 97 hash, di password comuni, su 100 è di quasi 6 ore con modalità mista, sia wordlist sia brute force. In Figura B.2(b) vediamo 17 hash decriptati (i primi 10 da wordlist, gli ultimi 7 da brute force). Per raggiungere il 100% di hash decriptati si potrebbe provare ad utilizzare una wordlist più grande, lasciare che il brute force sia eseguito per più tempo oppure provare con un attacco delle collisioni; ma l'obiettivo non era raggiungere la totalità di hash decriptati ma dimostrare quanto password deboli e comuni, criptate con Md5, fossero vulnerabili ad un'attaccante che prova a decriptarle.

Per dimostrare le vulnerabilità dell'algoritmo Md5, si è preso l'elenco delle 100 password più usate nel 2020 [fonte NordPass] e convertiti in hash con Md5 (Figura B.1). Per decifrare gli hash è stato utilizzato "Hashcat", un software open-source per la decriptazione di hash, installato in ambiente Cloud; la scelta di utilizzare un'ambiente Cloud è data dal fatto che l'hardware fornito per la decifratura degli hash, avesse capacità computazionali molto più elevate rispetto a quelle di un computer in locale. Inoltre abbiamo usato una wordlist con quasi 60000 password comuni da confrontare con gli hash, per cercare corrispondenza tra gli hash e le password. Come possiamo vedere nella Figura B.2(a) si impiegano pochi secondi per decifrare 90 hash su 100; per i restanti 10 si utilizza un attacco Brute Force che prova tutte le combinazioni tra lettere minuscole, maiuscole, numeri e caratteri speciale.

L'attacco Brute Force impiega quasi 6 ore per decriptare 7 password su 10. Quindi il

```
Started: Mon Oct 25 13:17:02 2021
Stopped: Mon Oct 25 13:17:04 2021
```

(a) Tempo di esecuzione con wordlist

```
1 e19d5cd5af0378da05f63f891c7467af:abcd1234
2 dc647eb65e6711e155375218212b3964:Password
3 81dc9bdb52d04dc20036dbd8313ed055:1234
4 6eea9b7ef19179a06954edd0f6c05ceb:qwertyuiop
5 97db1846570837fce6ff62a408f1c26a:1q2w3e4r5t
6 b2fe440cb7a0b127f1a90ffea296313b:shopping
7 01cfdc4f6b8770febfb40cb906715822:54321
8 6fb42da0e32e07b61c9f0251fe627a9c:0987654321
9 c44a471bd78cc6c2fea32b9fe028d30a:asdfghjkl
10 432f45b44c432414d2f97df0e5743818:12345678910
...
1 098f6bcd4621d373cade4e832627b4f6:test
2 18aa20d71f1624aac4c14fe5cf972d56:zinch
3 5a105e8b9d40e1329780d62ea2265d8a:test1
4 2ee2abccdda7424657a2fc26873f5a5ed:animoto
5 1b53230416980fce41adbac1cba86956:Chegg123
6 cd61a16f33a745d361b355bf1d627da9:livetest
7 e084a4d403be5343520a951f33090d07:dubsmash
```

(b) Risultato dopo la decifratura

Figura B.2: Tempo e output dopo la decifratura

C. Grafici dei tempi di cifratura e decifratura

Vediamo alcuni grafici che rappresentano il tempo di cifratura (Figura C.1(a)), decifratura (Figura C.1(b)) e la media tra vari algoritmi (Figure C.2(a) e C.2(b)). Sono stati confrontati DES, 3DES, AES, Blowfish e RSA utilizzando file di dimensioni 25KB, 50KB, 1MB, 2MB e 3MB composti da testo e immagini come input per la crittografia. L'output crittografato di ciascun file viene salvato come file, che a sua volta viene immesso per la decrittazione. Per motivi di confronto sono stati utilizzati gli stessi file di input per tutti gli algoritmi durante l'esperimento e lo stesso sistema per tutte le implementazioni e il lavoro di analisi, in modo che le condizioni della memoria e del processore rimangano le stesse per tutti gli algoritmi di confronto.

L'analisi viene eseguita con le seguenti metriche in base alle quali i crittosistemi possono essere confrontati:

■ Tempo di crittografia

Il tempo impiegato per convertire il testo in chiaro in testo cifrato è il tempo di crittografia. Il tempo di crittografia dipende dalla dimensione della chiave, dalla dimensione del blocco di testo in chiaro e dalla modalità; si è misurato il tempo di crittografia in millisecondi.

■ Tempo di decodifica

Il tempo per recuperare il testo in chiaro dal testo cifrato è chiamato tempo di decodifica. Si desidera che il tempo di decrittazione sia meno simile al tempo di crittografia per rendere il sistema reattivo e veloce; in questo contesto è stato misurato in millisecondi.

■ Memoria utilizzata

Diverse tecniche di crittografia richiedono diverse dimensioni di memoria per l'implementazione. Questo requisito di memoria dipende dal numero di operazioni che l'algoritmo deve eseguire, dalla dimensione della chiave utilizzata, dai vettori di inizializzazione utilizzati e dal tipo di operazioni.

■ Effetto valanga

In crittografia, una proprietà chiamata diffusione riflette la forza crittografica di un algoritmo. Se c'è un piccolo cambiamento in un input, l'output cambia in modo significativo; questo è anche chiamato effetto valanga. È stato misurato l'effetto valanga utilizzando la distanza di hamming. La distanza di Hamming nella teoria dell'informazione è una misura della dissomiglianza; calcolata come somma di bit XOR bit considerando il valore ascii. Si desidera un alto grado di diffusione, ovvero un elevato effetto valanga. $\text{Effetto valanga} = (\text{distanza di hamming} / \text{dimensione del file})$.

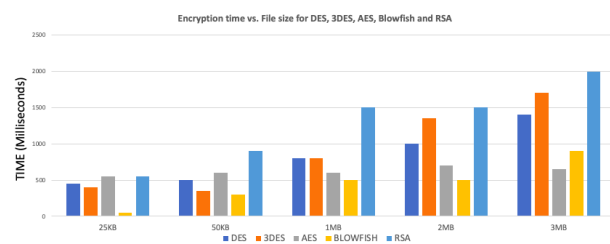
■ Entropia

La casualità è una proprietà importante nei processi crittografici perché le informazioni non dovrebbero essere in grado di essere indovinate da un utente malintenzionato; l'entropia è la misura della casualità nell'informazione. Misura l'incertezza nelle informazioni. Si richiede che

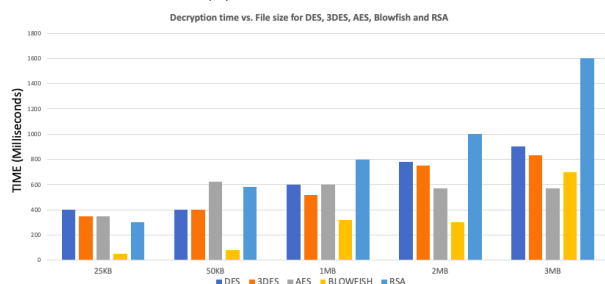
gli algoritmi forniscano un'elevata casualità nei messaggi crittografati, in modo che ci siano meno o nessuna dipendenza tra chiave e testo cifrato. Con un'elevata casualità, la relazione tra chiave e testo cifrato diventa complessa. Questa proprietà è anche chiamata confusione. Un alto grado di confusione è auspicabile per rendere difficile da indovinare ad un attaccante. Calcoliamo l'entropia usando la formula di Shannon.

■ Numero di bit necessari per la codifica ottimale

Il bit crittografato verrà trasmesso su una rete dopo la codifica, questa metrica ci indica la larghezza di banda richiesta per la trasmissione. Se un bit crittografato viene codificato con un numero inferiore di bit, consumerà una larghezza di banda inferiore e anche uno spazio di archiviazione inferiore.

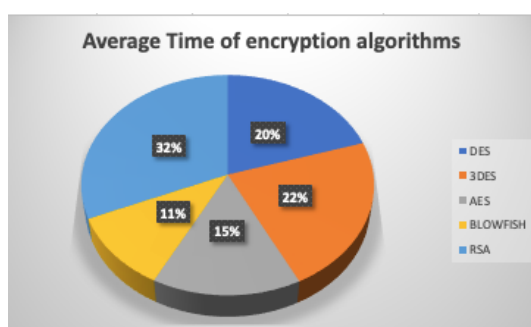


(a) Encryption time

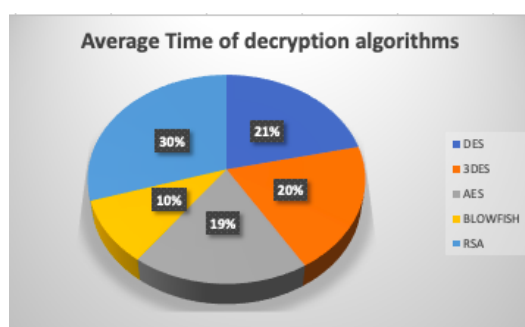


(b) Decryption time

Figura C.1: Tempi di cifratura e decifratura



(a) Average encryption time



(b) Average decryption time

Figura C.2: Media dei tempi di cifratura e decifratura

D. IPsec in breve

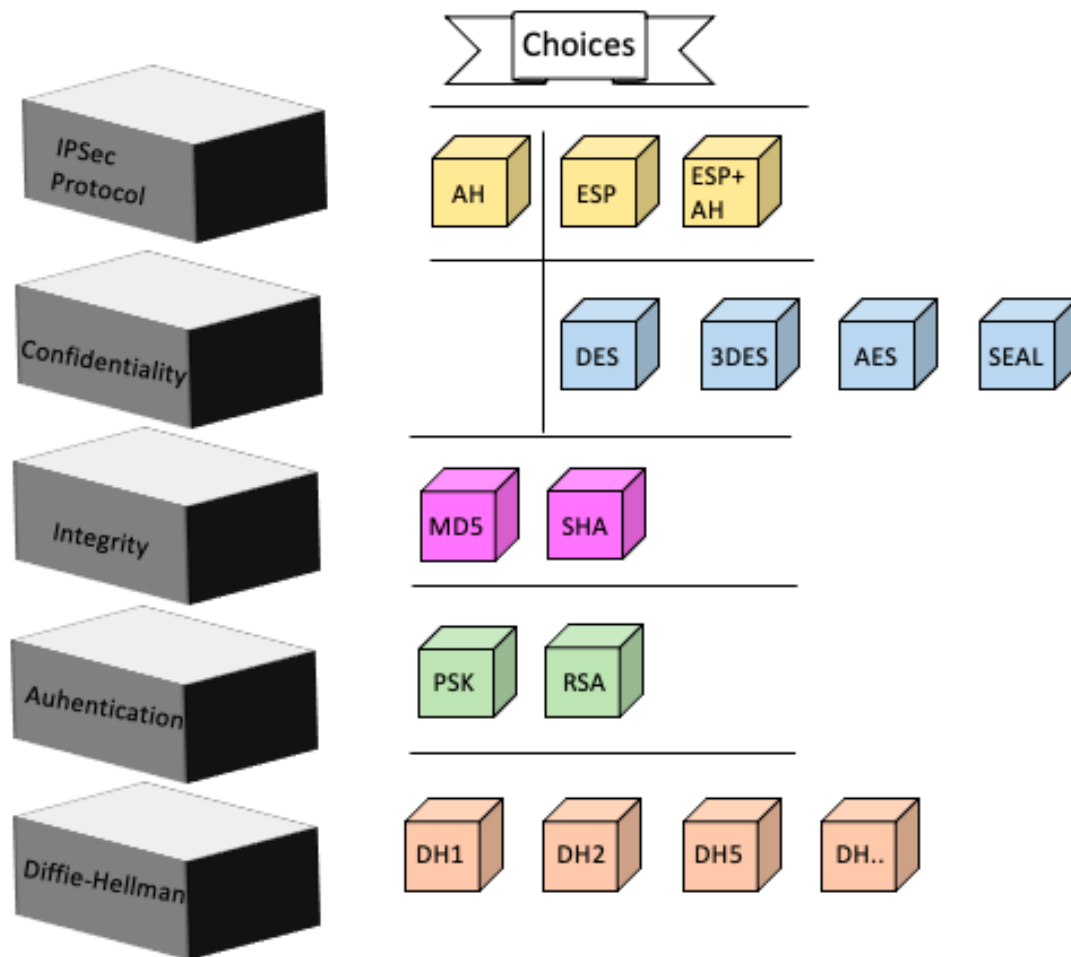


Figura D.1: Composizione di IPsec