Computer and Network Security

Lorenzo Rossi

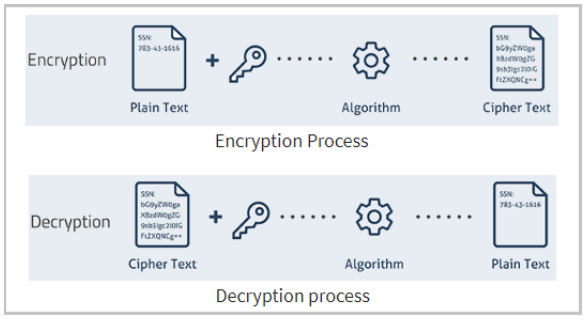
October 24, 2021

Contents

1 Definizioni e basi sulla crittografia simmetrica

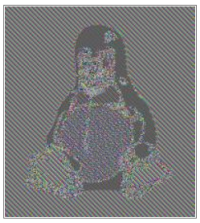
L'obiettivo della crittografia è quello di trasformare i dati in un qualcosa di incomprensibile al fine di garantire confidenzialità e sicurezza. Questa trasformazione deve essere **reversibile**. Ne è un esempio la crittografia simmetrica in cui si utilizza una chiave che viene utilizzata sia per cifrare che decifrare i dati. La chiave, come appena detto, serve a cifrare e decifrare i dati e perciò deve essere privata, cioè conosciuta solo da coloro che sono impegnati nella trasmissione. Tuttavia, per permettere la cifratura/decifratura, si necessitano di algoritmi che devono essere noti, cioè pubblici. L'agoritmo che cifra e decifra viene detto **chiper**. L'iter che viene eseguito è il seguente:

- Il testo non cifrato **plaintext** P viene cifrato tramite l'algoritmo pubblico (chiper);
- Il chiper è una funzione del tipo $C = ENC(K, P)$ che produce il messaggio cifrato **chipertext** C ed inviato al destinatario;
- Il chipertext viene decifrato dal chiper,che ricordiamo esser pubblico, dalla funzione $P = DEC(K, C)$ producendo così il plaintext **originale**.



1.1 Substitution cipher

Il **substitution cipher** è un algoritmo di cifratura che, preso un plaintext, sosituisce ad ogni lettera un'altra lettera dell'alfabeto. Questo algoritmo è facile da decifrare e poco sicuro poiché è possibile capire la chiave utilizzata (lo schema utilizzato per sostituire le lettere) a mano, ma anche tramite un *frequency analysis* che conta la frequenza delle lettere in una parola. Una volta ottenuta questa frequenza, la si confronta con quella delle parole dell'alfabeto utilizzato e, così facendo, risulta immediato risalire al testo originario. Un'altra problematica è che questo cipher non preserva la confidenzialità, poiché lo **stesso** plaintext presenta lo **stesso** cipher text. Ne sono un esempio le immagini:



1.2 One time pad - Vernam cipher

Il **one time pad** (Vernam cipher) è un algoritmo di cifratura che consiste nella creazione del ciphertext effettuando uno XOR[1] tra il plaintext e una chiave casuale lungua quanto il messaggio da inviare. Definendo M il plaintext e K la chiave, possiamo definire l'operazione di cifratura:

$$CT = ENC(K, M) = M \bigoplus K$$

A questo punto abbiamo ottenuto il messaggio cifrato CT, per decifrare il cipher text si utilizza l'operazioen di XOR fra la chiave e ciphertext:

[1] l'operazione di XOR funziona come segue:

XOR Truth Table		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

$$CT \bigoplus K = (M \bigoplus K) \bigoplus K = M \bigoplus (K \bigoplus K) = DEC(K, CT) = M$$

Il **Vernam Cipher** è incondizionatamente sicuro, poiché rispetta il **teorema di Shannon** per cui un cipher è detto sicuro se:

- Vi sono tante chive quanti messaggi;
- Le chiavi devono essere lunghe quanto il plaintext;
- Se le chiavi sono **casuali**;

Anche se questo cipher rispetta il teorema di Shannon per la crittografia, risulta:

- Assente di integrità (vedere l'esempio sul ciphertext dell'immagine);
- Insicuro se le chiavi vengono riusate: l'operazione di XOR cancella le chiavi e lascia i plaintext concatenati in XOR;
- Le chiavi **Random** sono difficili da ottenere poiché sono ben diverse da quelle pesudorandomcihe.

Riguardo all'ultimo punto, per ottenere una chiave pseudorandomica si utilizza un **PRNG** (Pseudo Random Number Generator) che è a tutti gli effetti un algoritmo e può risultato buono o sbagliato su diversi aspetti:

- Risulta una scelta buona poiché si ottengono proprietà statistiche sull'uscita;
  AGGIUNGI ROBA
- Può essere previsto;
- Essendo pseudo randomicomo, le chiavi che vengono generati si ripetono con una certa periodicità;
- Tutti questi primi 3 aspetti, nella crittografica, sono assolutamenti da evitare, se possibile.

Al contrario per ottenere una chiave veramente casuale si utilizzano i **TRNG** (Truly Random Number Generator) in cui si prendono fenomeni fisici per estrarre la casualità. Ne possono essere un esempio il rumore atmosferico e il rumore termico dei resistori.

1.3 One Time Pad - The Attacker

Ipotizziamo di avere un **One Time Pad** con un PRNG. Finché l'algoritmo utilizzato per generare le chiavi non è noto sembra essere tutto abbastanza sicuro tuttavia ciò non è possibile poiché, per ipotesi, è pubblico. Inoltre, se ricevessimo due messaggi *M1* e *M2* la situazione cambierebbe:

$$M1 \bigoplus M2 = (S \bigoplus K_i)(S \bigoplus K_{i+1}) = K_i \bigoplus K_{i+1}$$

Quindi, sapendo che il chipertext è costante, che si utilizza un PSEUDO random generator e che PRNG risulta noto. Per ottenere il plaintext basterebbe effettuare le seguenti operazioni:

Algorithm 1 Break Vernam Cipher - PRNG Version

Run: for$(x_i = 0; x_i < 2^{16}; x_i++)$
$Z_i = x_i \bigoplus PRNG(x_i)$
if $Z_i == M1 \bigoplus M2 = K_i \bigoplus K_{i+1}$ **then** $K_i = PRNG(x_i)$

1.4 Definizioni di Sicurezza

Per valutare la qualità di un cipher occorre dare delle definizioni chiave che non possono essere di carattere "assoluto", ma che definiscano con precisione la sicurezza rispetto ad un modello di avversario. Questo perché un cipher può essere sicuro contro un attaccante che è abilitato ad accedere solo al chipertext ma non contro quelli che possono vedere sia porzioni di plain/cipher text. Come convenzione si utilizza un **modello** rappresentante lo scenario in cui si deve raggiunge l'**obiettivo**. A livello internazionale si utilizza come baseline di sicurezza:

IND-CPA

In cui:

- **IND**: INDistinguishability
  Under
- **CPA**: Chosen PlainText Attack

1.5 IND-CPA Game

Andiamo ora a definire il modello e l'obiettivo di IND-CPA. Assumiamo la presenza di una vittima ed di un avversario. L'avversio è a conoscenza del cipher utilizzato e del plaintext.

- **IND-PHASE**:
  1. Scegli M_0, M_1 della stezza lunghezza;
  2. L'avversario invia M_0, M_1 alla vittima;
  3. La vittima prende casualmente uno dei due messaggi con un lancio di una moneta $M_b, b \leftarrow rand(0, 1)$;
  4. La vittima genera il messaggio cifrato $C = ENC(K, M_b)$ e lo invia all'avversario;
- **CPA-PHASE**:
  5. A questo punto l'avversario non sa neanche quale chipertext ha ricevuto e quindi invia i plaintext M_0, M_1 ad un oracolo che conosce sia la chiave e sia il cipher utilizzato.
  6. L'oracolo risponde con due ciphertext $C_0 = ENC(K, M_0)$ e $C_1 = ENC(K, M_1)$
- **GOAL**: Se l'avversario deve inviare ulteriori messaggi alla vittima per capire quale sia il plaintext, allora il cipher è sicuro. Altrimenti, il cipher viene detto **broken**.

1.6 IND-CPA: Conseguenze

Come abbiamo già detto, un avversario non deve essere in grado di calcolare qualsiasi informazione su un plaintext anche se potessa aver accesso ad oracolo di cifratura. Riformulando:

Un avversario, dati due plaintexts di stessa lunghezza e dato un ciphertext il quale contiene un messaggio scelto randomicamente tra i due, non dovrebbe essere in grado di distinguere quale dei due sia.

Le conseguenze di IND-CPA sono che:

- La cifratura deve essere randomizzata. Quindi uno stessa messaggio deve restituire diversi ciphertext e quest'ultimo deve essere casuale e quindi indistinguibile.
- Se si ripete una sottostringa, deve essere cifrata di due ciphertext differenti.

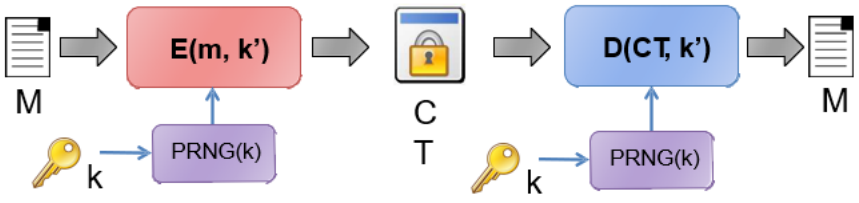
2 Stream-Cipher e WEP

Nel mondo reali i ciphers possono essere divisi in due categorie:

- **STREAM ciphers**:
  Deboli: RCA;
  Forti: Salsa20,ChaCha20;
- **BLOCK Ciphers**:
  Deboli: DES,3DES;
  Forti: AES;
- **BLOCK ciphers in STREAM MODE**:
  AES-CTR;

2.1 Stream Ciphers

Lo **stream cipher** richiede una chiave simmetrica di lunghezza finita scelta casualemente (K), mentre la chiave utilizzata per la cifratura e la decifratura viene scelto dalla base della chiave simmetrica casuale tramite un algoritmo (**PRNG(K)** pseudorandom (**keystream** K'). Per questi motivi, viene detto un cipher one time pad "approssimato".



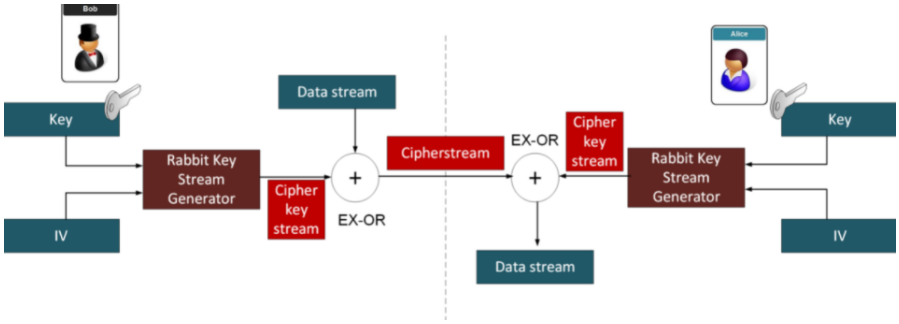
Le funzioni di cifratura e decifratura rimangono invariate:

$$CT = ENK(M, M) = M \bigoplus keystream = M \bigoplus PRNG(K)$$
$$M = DEC(K, CT) = CT \bigoplus keystream = CT \bigoplus PRNG(K)$$

Se si ripete una sottostringa, il chipertext cambia. Tuttavia, se il messaggio viene cifrato due volte, il chipertext si ripeterà dato che il cipher utilizzato è deterministico. Ne è un esempio RC4.
Quindi la problematica è nel fatto che la chiave è statica.

2.2 Initialization Vectors (IV) in Strem Ciphers

Al fine di garantire **sicurezza semantica** si introducono gli **initialization vectors IV**. Esso è un valore casuale differente per ogni messaggio che garantisce l'unicità del ciphertext. In particolare, si IV viene compinato con la chiave dal cipher PRNG(K||V); ne viene fatto lo XOR con il plaintext, ottenendo il chipertext a cui si aggiunge l'initialization vector IV. Quest'ultimo risulterà sempre diverso e la non conoscenza di IV **NON** è un requisito di sicurezza.



2.3 WEP

L'esempio di WEP è utile a capire perché un buon algoritmo di cifratura non è abbastanza a rendere sicuro un sistema.Inanzitutto, iniziamo con il definire gli obbiettivi preposti da WEP:

- Autenticazione:
- Integrità:
- Confidenzialità;

Al fine di garantire questi tre aspetti in Wi-Fi, si scelse di utilizzare come WEP cipher RC4 per generare il keystream. **RC4** funziona simile a stream con PRNG:

- $ENC(KEY, MSG) = MSG \bigoplus RC4(IV, KEY)$
- $Keystream = RC4(IV, KEY)$

Quindi, WEP **per ogni** frame invia un IV differente poiché se avessero inviato un IV per unico keystream si poteva essere poco efficienti in caso di perdita di pacchetti. Inoltre,come anticipato in precedenza, la trasmissione in WEP di IV in chiaro non è un problema perché da esso non si dovrebbe rompere la sicurezza se si utilizza un buon stream cipher. Avendo, quindi assunto che RC4 è forte, allora WEP è sicuro finché IV non si ripete mai. Tuttavia vi erano queste problematiche:

- Stesso keystream se stessa (Key,IV) e quindi assenza di sicurezza semantica;
- Se IV si ripete, la debolezza è pratica.
- utilizza di un IV di 24bits;
- Generazione di IV lasciato al programmatore.

Le conseguenze di queste scelte hanno portato ad una ripetizione dello stesso IV in 2^{24} cicli,corrispondenti a 8 ore se ogni frame è di 1500bytes e la rete di 7mbps. Questa soluzione si rivelò disastrosa e anche se IV veniva scelto in maniera casuale e anche se si sceglieva una chiave di dimensioni maggiori perché il problema era che la lunghezza di IV era troppo corta.

2.4 Attacchi pratici verso un IV di piccolo dimensioni

2.4.1 Attacco Passivo

Un attacco che si può effettuare in presenza di IV (initialization vectors) di piccolo dimensioni è quello di creare un **dizionario** formato da:

$$IV \rightarrow RC4(IV, Key)$$

Si utilizzano messaggi noti per recuperare il Keystream nel seguente modo:

$$(MSG \bigoplus RC4(IV, key)) \bigoplus MSG \rightarrow RC4(IV, key) \tag{1}$$

I messaggi noti possono essere raccolti anche tramite l'invio di email contenenti grandi allegati. Per proteggersi da questi attacchi si necessita di una **procedura di autentitcazione**

2.4.2 IV ripetuti

Un'altra tecnica è quella di aspettare che un IV sia ripetuto ed in particolare:

$$(M' \bigoplus RC4(IV, key)) \bigoplus RC4(IV, key) = M' \tag{2}$$

Come già discusso in precedenza, l'aumento della grandezza della chiave non risolve il problema poiché l'IV mantiene la stessa lunghezza di 24 bits, insufficiente per mantenere un certo grado di sicurezza.

2.4.3 Attacco ad RC4

RC4 è l'algoritmo utilizzato da WEP per generare il keystream utilizzato per cifrare i messaggi. Fluhrer, Mantir e Shamir riuscirono a rompere questo algoritmo notando che nel 5% dei casi la chiave generata da RC4 faceva intuire il valore della chiave utilizzata come argomento di questafunzione. Quindi dimostrarono che, accumulando traffico, era possibile ottenere la chiave segreta simmetrica. Questo attacco era molto pratico poiché bastava utilizzare il traffico già noto (es. traffico su internet) per ottenere componenti della chiave segreta. Inoltre, si è dimostrato che questo attacco è lineare con la dimensione della chiave.

2.5 User Authentication

Non bisogna confondere il concetto di **autenticazione** con il concetto di **identificazione**: nel primo, si vuole dimostrare la propria identità digitale (es. email,ID); nel secondo, si vuole mostrare che la propriaidentità digitale è controllata dal proprietario. Nel pratico il concetto di autentitcazione utente non è facilmente raggiungibile poiché non è possibile collegare l'ID all'identità nella vita reale. Quindi un organo internazionale chiamato NIST SP, fornisce una definizione di autenticazione utente.

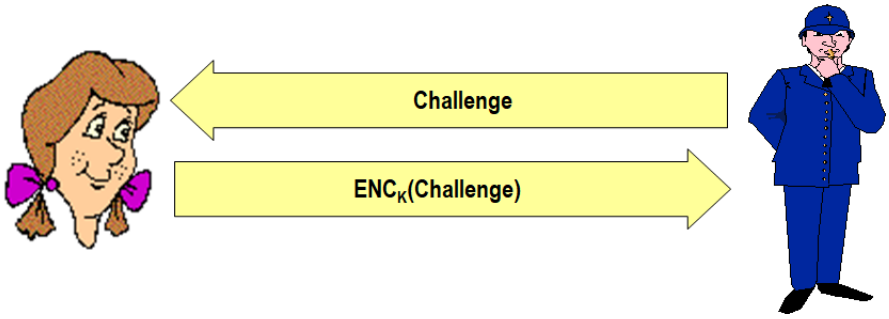
Definition 2.1 (Autenticazione utente)**.** *L'autenticazione utente è il processo per stabilire la fiducia nelle identità degli utenti presentate elettronicamente a un sistema informativo.*

I mezzi che possono essere utilizzati per verificare l'autenticazione, possono essere i seguenti:

- Qualcosa che solo tu conosci: Password,PIN, risposte a domande già note;
- Qualcosa che solo tu hai: Smart card, dispositivi fisici come token;
- Qualcosa per cui tu sei: dati biometrici come retina, impronta digitale;
- Qualcosa che tu fai: come la scrittura e i movimenti del mouse sul computer;

2.5.1 Autenticazione Wep

L'idea dietro al Wep (**Wired Equivalent privacy**) non era quella di autenticare gli utenti, ma quella di redigere una lista di chi poteva utilizzare la rete. Tutti gli autorizzati nella rete avrebbero potuto comunicare senza ausilio crittografico, mentre per quelli al di fuori avrebbero osservato solamente i messaggi cifrati. Quindi, la conoscenza di un segreto comune rendeva un utente fidato. **L'idea di WEP**



L'idea è quella di utilizzare la stessa chiave del frame anche per l'autenticazione.

Remark. Questa pratica è **fortemente** sconsigliata poiché per servizi diversi si devono utilizzare chiavi differenti.

L'access point invia un valore numerico non cripato come plain text (challenge). L'utente che vuole accedere cripta il valore ricevuto in un modo da dimostrare la sua autenticità. Per ogni utente che arriva cambia la challenge(*sequenza da 128bits*).
Dato che la chiave è pre-condivisa, l'access point sarà in grado di riconoscere se il modo in cui l'utente ha cripato la challenge sia corretto o meno. UNa volta criptata la challenge, l'utente invia all'access point un IV scelto da lui unita alla challenge criptata in XOR con l'RC4(IV,Key). Gli errori nel fare ciò sono i seguenti:

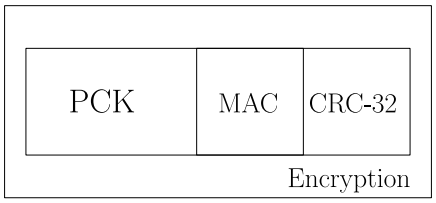
- L'autenticazione WEP è basata sul fatto che la challenge non venga criptata quando inviata dall'access point, ovvero si lascia che il planitext rimanga visibile. Facendo *Chipertext*$\bigoplus$ *Challenge* si ottiene la keystream. Il grande errore fu usare la stessa chiave nell'encryption e nella challenge. Il WEP in questo modo finisce per implementare un KPA (*Known Plaintext Attack*), in quanto chiunque conosce la challenge può costruire un dizionario composto da *IV-RC4(IV,Key)*. L'autenticazione costituisce un problema per la condifenzialità.
- Solo ascoltando una persona fidata per il sistema che conosce la chiave si può riuscire ad aggirare l'autenticazione. L'errore sta nel fatto che IV è deciso dalla macchina di chi manda la challenge criptata, ovvero da chi vuole entrare nel sistema. Cosi facendo, se si esegue:

$$Challenge \bigoplus (Challenge \bigoplus RC4(IV, Key)) = RC4(IV, Key)$$ **si ottiene lo stream.**

A questo punto, il sistema è sensibile ad attacchi. Posto un MITM(*man in the middle attack*, se esso ascolta IV di un messaggio e si ricava l'RC4 corrispondente, è in grado di entrare nella rete; se a quel punto chiede una challenge all'access point, può scegliere un IV precedentemente intercettato ed entrare nel sistema.
La soluzione a questo problema poteva essere quella di comunicare oltre alla challenge anche l'IV.

2.5.2 Integrità WEP

L'idea di WEP per l'integrità era quella di utilizzare CRC-32 come un valore di controllo di integrità così da evitare eventuali manomissioni dei messaggi in WEP. L'algoritmo **CRC-32** è un algoritmo di **error detection** che tiene conto dei bit di partenza tramite bit di parità posti alla fine del messaggio. Quindi, in pratica:



decisero di includere CRC-32 al pacchetti per poi criptarlo ed inviarlo in rete. Tuttavia, questa scelta si rivelò fallimentare poiché l'algoritmo CRC-32 è lineare rispetto all'operazione $\bigoplus$ e quindi ha permesso attacchi di due tipi:

- Modifica dei messaggi;
- Iniezione di messaggi.

2.5.3 Modifica del emssaggio

Supponiamo che un hacker volesse modificare qualche bit del plaintext in WEP.
Quando il messaggio viene mandato, vengono criptati tutti i bit:

$$C = M|CRC32(M) \bigoplus RC4(IV, Key)$$

In questo attacco prendiamo un vettore di bit δ della stessa lunghezza del messaggio in cui sono settati ad 1 i soli bit che l'attaccante vuole modificare. Quindi, calcoliamo il ciphertext di δ e calcoleremo il nuovo messaggio:

$$\begin{aligned} C' &= C \bigoplus \{\delta, c(\delta)\} = \\ &= [RC4(IV,K) \bigoplus M, c(M)] \bigoplus \{\delta, c(\delta)\} = \\ &= RC4(IV,K) \bigoplus \{M \bigoplus \delta, c(M) \bigoplus c(\delta)\} = \\ &= RC4(IV,K) \bigoplus \{M', c(M \bigoplus \delta)\} = \\ &= RC4(IV,K) \bigoplus \{M', c(M')\} \end{aligned}$$

Quindi siamo riusciti ad ottenere un messaggio modificato M' con RC4 valido.

Remark. Risulta importante concatenare a δ anche il suo CRC per ragioni di coerenza con il controllo dei bit di parità.

2.5.4 Iniezione del messaggio

In questo tipo di attacco, l'attaccante necessita ci una coppia IV,RC4(IV,K). Quindi:

- Calcola $c(M)$;
- Invia $IV, RC4(IV,K) \bigoplus [M|c(M)]$

2.6 Conclusioni

Il controllo di integrità deve essere non lineare per non consentire modifiche dei messaggi e deve includere esplicitamente una chiave diversa dato che un crittografica esterna non fornisce garanzie sull'integrità.

3 Message Authentication: Hash Function

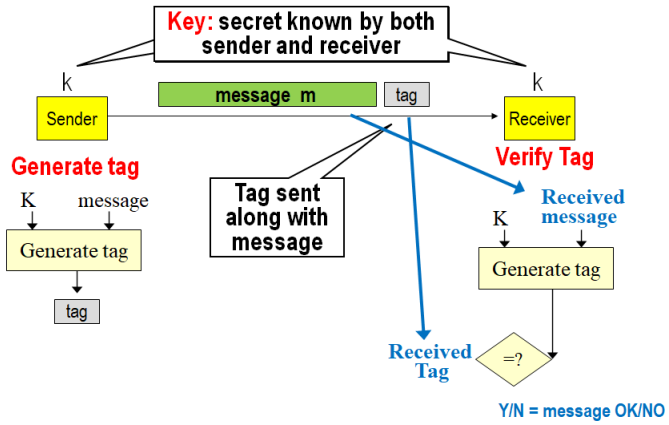
Occorre evidenziare che **confidenzialità** $\neq$ **integrità**. Iinfatti:

- **confidenzialità**: messaggi nascosti. Solo il leggittimo destinatario dovrebbe essere in grado di vedere il contenuto dei emssaggi;
- **Integrità**: autenticità del messaggio. Nessuno dovrebbe essere in grado di modificare un messaggio finché è in trasmissione e la sorgente leggittima dovrebbe essere il solo a forgiare un messaggio.

Questi predicati sono corretti finché non si parla di algoritmi AEAD (*authenticatted encryption associated data.*

3.1 Autenticazione del messaggio con chiave simmetrica

Il mittente e il destinatario conoscono una chiave k dettao **integrity key**. Questa è una chiave diversa da quella per criptare il messaggio già visto. L'obiettivo è quello di aggiungere un qualcosa al messaggio che serva da check per l'integrità. Questo qualcosa è il **TAG**.Il mittente invia al destinatario il messaggio e il tag. Quest'ultimo è una sequenza di bit di lunghezza fissata, di solito più corto del messaggio stesso, generata da un funzione che prende come argomenti la chiave K e il messaggio. In seguito, quando il destinatario dovrà ricevere il messaggio e quindi verificarne l'autenticità, confronterà il tag associato al messaggio e quello che il destinarario del messaggio avrà generato sulla base della chiave K ed il messaggio. Se questo confronto ha buon fine, allora il messaggio è integro.



3.2 Attacco MITM : attacco in cui l'attaccante modifica il messaggio in uno diverso

L'autenticazione di messaggio protegge dagli attacchi man in the middle. L'attaccante non può modificare il messaggio considerando le seguenti assunzioni:

- $TAG = F(K, m)$ la funzione F è crittograficalmente forte;
- L'attaccante vede sia il TAG che il messaggio m, ma non deve essere in grado di risalire a K;
- Non può cambiare il TAG senza conoscere la chiave;
- Non può cambiare il messaggio utilizzandone un altro che possiede lo stesso TAG.

Non è possibile prevenire un man in the middle attack, ma ci si può facilmente accorgere dell'attacco in quanto il tag non corrisponderà. L'attaccante ha solo 2^{-100} di probabilità di ricreare il tag giusto dopo aver modificato il messaggio di partenza affinché il destinatario non si accorga dello scambio.

3.3 Attacco message sppofing

L'autenticazione di messaggio protegge dal message spoofing,cioè si è protetti dalla ricezioni di messaggi la cui identità non è quella effettiva. Si basa sull'assunzione che la funzione che genera il tag sia crittogaficamente forte e non si può calcolare $F(K, X)$ senza conoscere la chiave segreta.

3.4 Replay attack

L'autenticazione di messaggio non protegge dai replay attack: quando si inviano due messaggi uguali, si otterrà lo stesso tag. Questo tipo di attacco non porta problematiche se si è certi che non ci possano essere messaggi uguali. Per permettere l'invio di due messaggi uguali leggettimamente si utilizzano i **nonce**. Questi sono delle informazioni che non devono rimanere segrete e che permettono la differenziazione di tutti i messaggi.
Queste informazioni aggiuntive vengono inserite nel messaggio così da inglobarle nel TAG e possono essere di tre tipi: **sequence number**,**random number**,**time stamp**.

3.5 Nonce: Pro vs Cons

- **Sequence Number**: la loro problematica riguarda il come gestire i reboot del sistema poiché in ogni avvia del sistema la sequenza ricomincia da capo, ma al contempo si è certi del messaggio che si sta ricevendo;
- **Random Number**: con 128 bit avrei $\frac{1}{2^{64}}$ probabilità di collisione. Tuttavia, non posso tenere traccia dei numeri che sono stati generati. Per risolvere questo aspetto, si potrebbe avere un database in cui si controllano tutti i numeri casuali che sono stati generati, tuttavia le linee da controllare potrebbero essere elevate;
- **Time step**: La problematica è capire come definire l'istante attuale e come prevenire modifica nel gestore dell'orario. Dato che l'orario è può essere impostato manualmente, da un Network Time Protocol (da cui è possibile effettuare attacchi di spoofing) o anche da sistemi di GPS.

3.6 MAC & Digital Signature

- Nella DS, nessuno può modificare il messaggio ad eccezione del creatore.
- Nessuno eccetto il destinatario e il mittente pososno modificare un messaggio autenticato;
- MAC e DS hanno l'obiettivo di garantire integrità dei dati/messaggi;
- Il MAC (message authentication code) è una forma più debole di una digital signature.

3.7 Definizioni di sicurezza per MAC

Per la crittografia si era parlato di modello INDCPA per definire la sicurezza. Ora per i messaggi occorre dare una definizione simile. La sicurezza è imperdonabile e quindi un attaccante non deve essere in grado di creare o modificare i messaggi e ciò implica che l'attaccante nopn dovrebbe essere in grado di estrarre la chiave dalla coppia $\{M, TAG(K, M)\}$. Formalmente, definiziona la sicurezza contro un modello di attaccante (*Known Message Attack, Chosen Message Attack, Adaptively Chosen Message Attack* a cui gli vengono dati un numero di coppie passate formate da messaggio e tag e un messaggio m. Quindi, l'attaccante non dovrebbe essere in grado di generare un tag valido. La "bontà" della sicurezza viene data in basse alla probabilità di generare una coppia valida e deve essere pressocché nulla. L'idea di tag minimo per garantire sicurezza è di 96 bit.

3.8 MAC: Hash function

Una **funzione hash** è una funzxione che sintetizza qualsiasi messaggio di lunghezza X in un digest di lunghezza fissa Y. Queste funzioni non sono invertibili. Per avere un MAC sicuro si ha bisogno di una buona funzione hash (come **SHA-256**) e di includere il segreto nella hash. A meno che non si possegga una funzione has perfetta (**random oracle**), ha importanza la posizione in cui si mette il messaggio nella sequenza di bit di ingresso alla funzione hash. Definiamo il **random oracle** come un modello black box tale che, dato X in puti, $H(X)$ è un valore veramente randomico. Data la proprietà di ripetibilità, a due input uguali corrisponodno due output uguali. Quindi è impossibile avere una funzione has perfetta poiché è è impossibile avere un digest veramente randomico. Le proprietà di una funzione hash sono le seguenti:

PREIMAGE RESISTANCE (ONE WAY)

- Una funzione hash non è invertibile: dato Y, risultato di una funziona hash, è difficile risalire a X; **AGGIUNGERE DA APPUNTI**

Second Pre Image Resistance

- Una funzione hash possiede la proprietà di resistenza alle collisioni debole: dato X, è difficile trovare un altro X' con lo stesso digest;

Collision Resistance

- Una funzione hash possiede la proprietà di resistenza alle collisioni forte: è difficile trovare due generici messaggi che hanno lo stesso digest.

3.8.1 Paradosso del compleanno

Il paradosso del compleanno ci aiuta a capire la probabilità di collisione, cioè di ripetizione, di due numero:

n bit digest → N=2ⁿ **K = n. messages** **p = collision probability**

$$P(\text{no collision}) = 1 - p = \frac{N!/(N-K)!}{N^K} =$$

$$= \frac{N}{N} \cdot \frac{N-1}{N} \cdot \frac{N-2}{N} \cdot \dots \cdot \frac{N-(K-1)}{N} =$$

$$= 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdot \dots \cdot \left(1 - \frac{K-1}{N}\right) =$$

Remember: for x small, 1-x approx e^(-x)

$$= \prod_{i=1}^{K-1} \left(1 - \frac{i}{N}\right) \approx \prod_{i=1}^{K-1} e^{-i/N} = e^{-\sum_{i=1}^{K-1} i \,/\, N}$$

$$= e^{-\frac{K(K-1)/2}{N}} \approx e^{-\frac{K^2}{2N}}$$

So, we concluded that: $1 - p \approx e^{-\frac{K^2}{2N}}$

Let's solve in K: $\log(1-p) \approx -\frac{K^2}{2N} \Rightarrow K = \sqrt{2N} \cdot \sqrt{\log \frac{1}{1-p}}$

50% (p=1/2) collision probability: $K \approx \sqrt{2} \cdot \sqrt{\log 2} \sqrt{N} \approx 1.177\sqrt{N}$

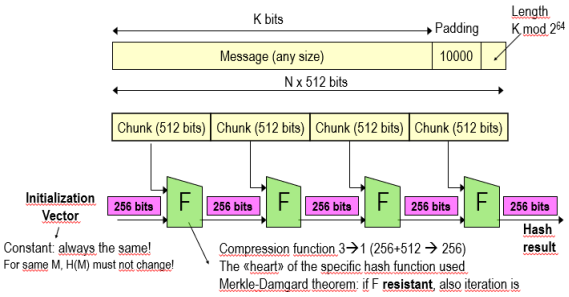
And since N=2ⁿ, using n bits: $K \approx 1.177\sqrt{2^n} \approx 2^{n/2}$

Conclusion: n-bit digest → security level is NOT 2^n but $2^{n/2}$

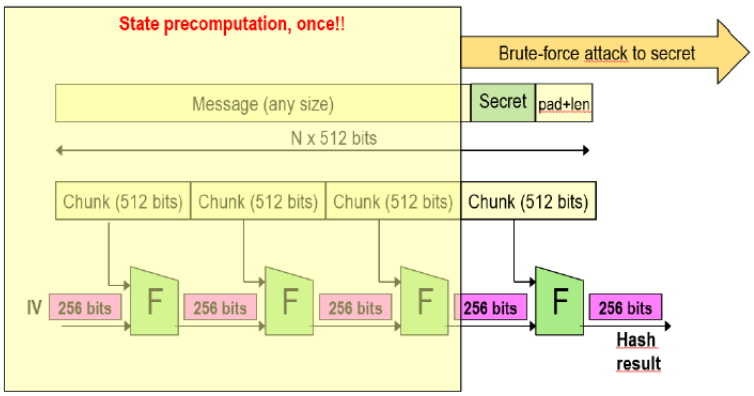
3.8.2 Costruzione di una funzione Hash

- **Segreto come suffisso** Si utilizza la costruzione di **Merkle-Damgard** per trasformare un set di k bit in un set di dimensione fissata. Ne è un esempio **SHA-256** che prende k bit arbitrari, li padda aggiungendo 10000

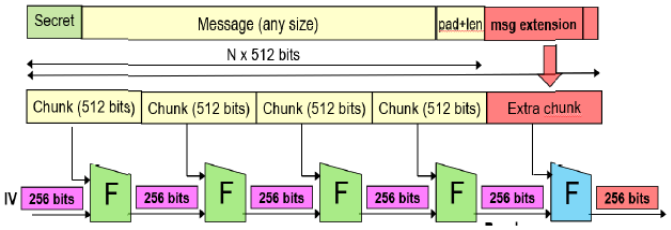
fino ad avere $N \cdot 512bit$; negli utlimi 64 viene messo un numero che costituisce la lunghezza del messaggio. In generale, si ha un messaggio ed usando il padding e last value si arriva ad avere un messaggio complessivo lungo un multiplo di 512 bit. Questi vengono tagliati in porzioni (chuncks) di 512bit. A questo punto viene preso un IV di 256 bit che è **DIVERSO** da quello utilizzato nella crittografia e lo si affianca con il primo chunck e , sommandoli, diventano da 768 bit; si applica F e si riottengono 256 bit. questi vengono riattaccata al secondo chunk e così via iterativamente fino alla fine. La **funzione di compressione** F è il cuore della costruzione della funzione di hash. Se F resiste, allora anche l'iterazione.



Un attacker potrebbe essere in grado di ridurre la complessità se si pre-calcolasse tutta la parte del messaggio, in modo tale da dover comprimere con F solo l'ultimo blocco. Questo ridurrebbe la complessità dell'attacco in quanto applicherebbe F una sola volta. Ma comunque è un **brute force attack**, quindi si dovrebbe comunque indovinare la combinazione corretta di bit.



- **Segreto come prefisso** Se mettessimo il segreto all'inizio, l'attaccante riesce ad indovinare pad+len e quindi ha violato il sistema. Questo perché oltre il pad+len si aggiunge anche un msg extension alla fine costituito da plaintext arbitrario. In questo caso si crea un extra chunck tale che riapplicando F si ottiene un messaggio valido ma non scritto dall'utente. Non vi è più integrità.**Message Expansion Attack**

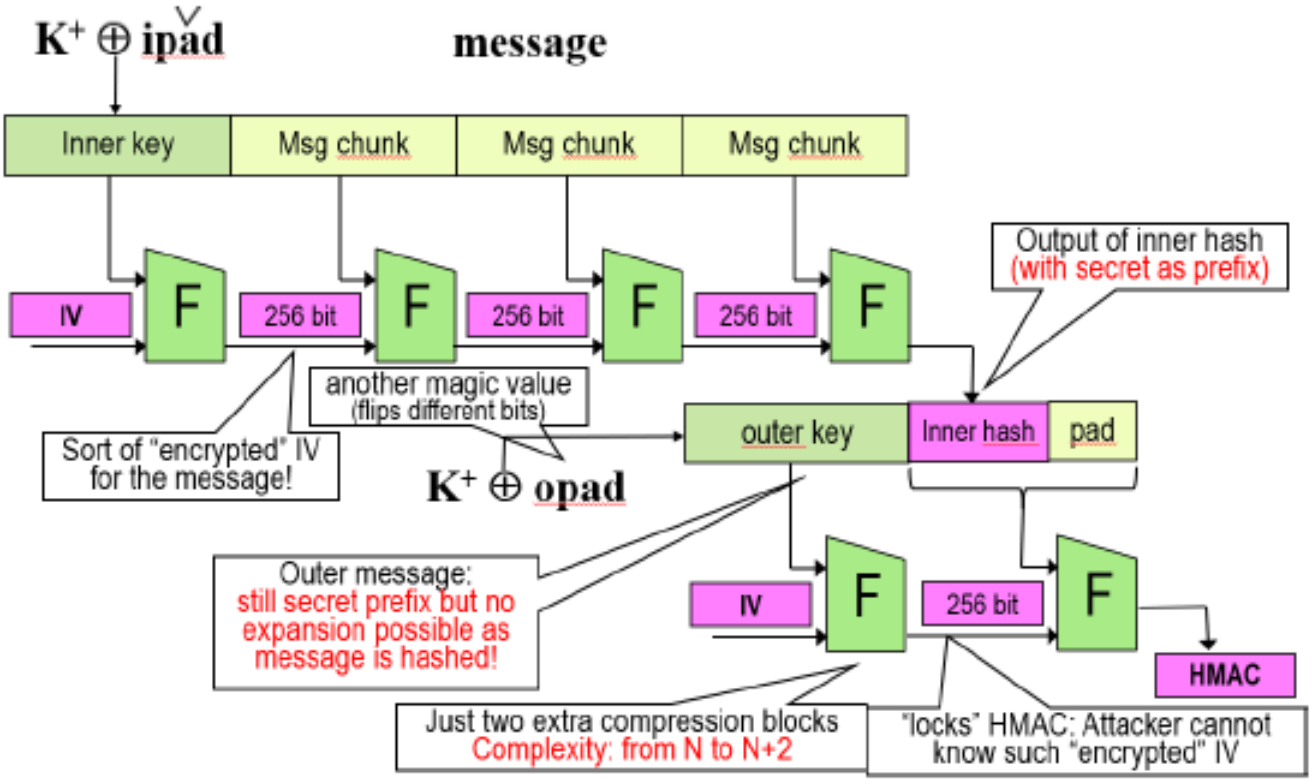


3.8.3 HMAC (Hash Based Message Authentication Code)

L'**HMAC** costituisce un modo sicuro di aggiungere il segreto nella funzione hash. In particolare, si vuole creare un MAC a partire da un messaggio e un segreto; si applica una funzione hash H sceltra tra quelle esistenti e si calcola:

$$HMAC_K(M) = H(K^+ \bigoplus opad || H(K^+ \bigoplus ipad || M))$$

In cui K^+ è la chiave precondivisa, estesa alla dimensione di un block grazie all'acquisizione di una serie di 0 di padding (completamento). Si ottenogno così 512 bit di chunk. Fatto ciò si necessita di un secondo segreto che però non viene chiesto all'utente. Per ottenere il primo segreto si prende K^+ esteso e si fa lo XOR con opad (*outerpad, serie di bit composta da 00110110 ripetuti*). Per ottenere il secondo segreto si prende la stessa chiave estesa ma si fa lo XOR con ipad(*inner pad formato da 010111100 ripetuti fino a diventare un chunk.*



Quindi all'utente viene richiesto solo un segreto che viene gestito come segue: si attacca all'inizio del messaggio $K^+ \bigoplus ipad$; si prende un IV conosciuto e si applica F, ottenendo 256bit che verranno messi in XOR alle porzioni dei messaggi uno dopo l'altro ogni volta riottenendo 256bit e riapplicando F. Infine, si ottiene un inner hash (vulnerabile) che sarà attaccato con $(outerkey||innerhash||pad)$. Ora, si esegue la F di IV e outerkey ottenendo 256 bit i quali saranno ricompressi con $innerhash||pad$.

Remark. • Inner hash richiede N compressioni;

• outer hash richiede 2 compressioni;

La complessità non è aumentata rispetto a prima, ma il risultato è sicuro.

Definition 3.1 (Messaggio Sicuro)**.** *Un messaggio risolta sicuro se ha un'uscita pseudo randomica e non vi sono collisioni.*

4 User Authentication: Password

Concettualmente, l'obiettivo dell'autenticazione è quello di dimostrare di sapere una password o un segreto. Tuttavia, si riscontra una differenza tra password e segreto:

- **secret key**: è una stringa (true) random di N bit. La probabilità di indovinare una chiave segreta(segreto) è pari a $\frac{1}{2^N}$ con N:= numero di bit;
- **Passowrd** è una stringa di N bit in cui i caratteri scelti per comporla non si basano su tutti i caratteri possibili. La conseguenza è che la probabilità di indovinare una stringa è molto minore di quella che si avrebbe con una chiave segreta. Per questo motivo, le password sono stringe a **bassa entropia**, cioè "poco" casuali.

4.1 Problematiche delle Password

Le problematiche delle **passowrd** sono le seguenti:

1. **Password Overload**: è il riutilizzo della stessa password su siti differenti.
2. **Restricted Charset**: l'insieme dei caratteri utilizzati non è completo; Per esempio:

$$1\ bit \to 256\ caratteri\ possibili; \quad Chiave\ segreta\ da\ 8\ bytes \to 2^8 = 64\ bit$$

La probabilità di indovinare un carattere è:

$$P_{guess}(s) = \frac{1}{256}$$

➔ **Secret key = 8 bytes**
⇨ 1 byte = 256 possible values
→Probability of guess = 1/256
⇨ Probability of guessing all 8 bytes
→1/256 * 1/256 * … (8 times) = 1 / 256^8
= 1 / 18.446.744.073.709.551.616
⇨ Assuming 66M guesses/s
» We will see later on why this number!
→Average guess time = $1.8x10^{19}$ / $6.6x10^7$ / 2 seconds = **4431 years**
➔ **Passwd = 8 bytes**
⇨ But if you use only low case letters and numbers?
⇨ 1 byte = 36 possible values!
⇨ Probability of guessing all 8 bytes password
→1 / 36^8 = 1 / 2.821.109.907.456
→Average guess time = $2.8x10^{12}$ / $6.6x10^7$ / 2 seconds = **5.9 hours!!**

Ripetuta per 8 caratteri:

$$P_{totalGuess}(s) = \frac{1}{256^8}$$

Ora assumiamo che il nostro computer abbia una potenza di calcolo di $Rate = 66\frac{Mguesses}{s}$. Il tempo medio speso per indovinare la password è pari a :

$$T_{avarege} = \frac{1}{2}\frac{2^N}{Rate} = \frac{1}{2}\frac{1.8 \times 10^{18}}{6.6 \times 10^8} = 4431\ anni$$

Nel caso di una Password di 8 bytes, ipotizziamo che l'utente possa utilizzare letteree numeri lowercase.

$$1\ byte = 36\ possibili\ valori$$

La proboabilità di indovinare una password è pari a:

$$P_{Totalguess}(p) = \frac{1}{N^K} = \frac{1}{36^8}$$

Quindi il tempo medio impiegato è:

$$T_{avarege} = \frac{2.8 \times 10^12}{6.6 \times 10^6}\frac{1}{2} = 5.9\ ore$$

Come si può vedere la decrescita è esponenziale.

3. **Low Entropy**: Le password devono essere ricordate e quindi non sono randomiche poiché associate ad algoritmi che incosciamente facciamo. L'entropia è la misura di quanto è casuale un qualcosa.

   Remark. In un **Brute Forse Attack** per indovinare k lettere in cui ogni lettera appartienze ad un alfabeto di n lettere. Occorre provare n^k passowrd differenti **SE** la stringa è stata generata *casualmente*.

   Sia X una variabile discreta casuale tale che:

   $$X = \{x_1, .., x_n\} con P = p_i i = 1, .., n\ Probabilit\ uniforme$$

   Definition 4.1 (Entropia)**.**

   $$H(X) = -\sum_i p_i log_2(p_i)$$

   Remark. Viene rappresentato in bit e definisce la quantità di informazione osservando l'evento X; Per esempio:

   - *lancio di una monenta*:
   - $x_1 = \ faccia\ A \quad x_2 = \ faccia\ B \quad p_1 = xp_2 = \frac{1}{2}$

     $$H(x) = -2(\frac{1}{2}log_2(\frac{1}{2})) = -log_2(\frac{1}{2}) = log_2(2) = 1$$

     Quindi l'informazione contenuta in una moneta è di 1 bit. Nel lancio di una moneta truccata:$x_1 = faccia\ A \quad x_2 = \ faccia\ B \quad p_1 = 1p_2 = 0$

     $$H(x) = -1log_2(\frac{1}{2}) = 0$$

     Quindi vi è total,ente assenza di informazione. Il risultato del prossimo evento è totalmente predicibile.
   - PIN Code formato da numeri. Ogni numero può assumere 10 valori (0,1,2,..,9) per quattro cifre. Quindi:

     $$p_i = \frac{1}{10} \rightarrow P_{Tot} = \prod_{i=1}^{4} p_i = \frac{1}{10^4}$$

     L'entropia vale:

     $$H(x) = -log_2(\frac{1}{10^4})10^4\frac{1}{10^4} = log_2(10^4)bit = 13.28bit$$

     Nel caso di giorni e mesi:

     $$H(x) = -log_2(\frac{1}{366})366\frac{1}{366} = 9bit$$

   Altri Esempi Si trovano sui fogli

Riassumendo, il **valore informativo** di x_i dipende da quanto l'evento x_i è inatteso: minore è la proabilità p_i, maggiore è il valore informativo di quell'evento.

Definition 4.2 (Information Content)**.**

$$Information\ content = funzione\ di\ \frac{1}{p_i}$$

Definition 4.3 (Entropia)**.** *L'**entropia** è il valor medio dell'**information content**.*

$$H(X) = E[IC(X)] = \sum_i p_i IC_i = -\sum_i p_i log_2(p_i)$$

➔**Bit flip: X_k = {0,1} with probability ½**
➔Entropy = 1 bit

➔**Entropy of sequence of independent r.v.: X_1 X_2 X_3**
➔Entropy = 3 bits

➔**What if they are dependent?**
⇨X_1 is bit flip, but X_2 and X_3 have same value as X_1
» Hence they are still {0,1} with probability ½
but depend on the previous value
➔Entropy = ???
⇨**Answer: 1 bit!**
⇨Why? Two trivial explanations
➔Information is only carried by one bit (bit X_1)
➔r.v. $Y = X_1\ X_2\ X_3$ has only two outcomes; 000 and 111, with p=½

4. **Predictability and Dictionary Attacks**: la scelta delle password è sbagliata e in genere si usano password intuibili dall'umano. I **dictionary attacks** sono una serie di euristiche di **attacco brute force** in cui si utilizzano parole comuni, database pubblici di password (ottenuti da altri attacchi andati a buon termine)o dizionari specifici per l'utente che attacchiamo per indovinare la password di un utente. In particolare, per ogni utente si provano tutte le stringe ottenute dai dizionari. Questo è un tipo di attacco che può essere svolto online (facilmente difendibile imponendo un massimo numero di tentativi) o offline (assenza di difesa a patto che si scelga una password forte).

4.2 Password-Based Authentication vs Challenge-Handshake Authentication - PAP,CHAP

4.2.1 PAP-Passowrd Authentication Protocol

Il **PAP** è il più semplice approccio di autenticazione possibile. Per dimostrare di conoscere il segreto (passworrd) lo si trasmette in chiaro. In particolare, si invia ID e PASSWORD all'authenticator che possiede un database con scritte tutte le associationi utente/password (ovviamente sono pre-condivise); quest'ultimo controlla la corrispondenza di quesi due valori e in caso affermativo garantisce l'accesso.
Tuttaia questi protocolli hanno delle limitazioni:

1. Dato che la password e l'id sono trasmessi in chiaro, se qualcuno ascolta mette a repentaglio la sicurezza;
2. Non vi è protezione da Replay Attack dato che ad un attacker basta ascoltare il messaggio che viene inviato dall'utente all'authenticator per acquisire credenziali valide per entrare nel sistema;
3. Non vi è protezione intrinseca sul numero dei tentativi per entrare e quindi si forniscono più tentativi ad un ipotetico attacker per tentare l'accesso.

4.2.2 CHAP-Challeng-Handshake Authentication Protocol

Nella autenticazione devo dimostrare di conoscere la password, ma non è necessario che debba dirla in maniera esplicita. Infatti nel **CHAP** non viene rilevata la password(P), ma la sua funzione di computazione (H(P)), ovvero un prodotto di byte ottenuto a partire dalla password. Assumiamo che:

1. Questa funzione non sia invertibile: se così non fosse un attacker potrebbe intercettare il messaggio ed eseguire la funzione inversa;
2. H non deve essere una funzione della solo P, ma deve contenere delle *freshness* (**nonce,challenge**);
3. H(P) non deve essere ripetibile.

Il CHAP funzione nella seguente maniera:

1. L'authenticator manda una **challenge** che **NON** deve mai ripetersi.
2. L'utente risponde con l'User ID (**UID**) e la risposta definita come $H(Challenge, Password, Dati\ aggiuntivi)$;
3. L'authenticator prende l'User ID (**UID**) e vede nel databse quale password è associata a quell'utente. Inoltre, conoscendo anche la challenge che aveva mandato, può riapplicare la funzione H da solo per ottenere la rispsota se la risposta dell'utente e quella appena calcolata corrispondono, allora l'utente viene autorizzato ad entrare.

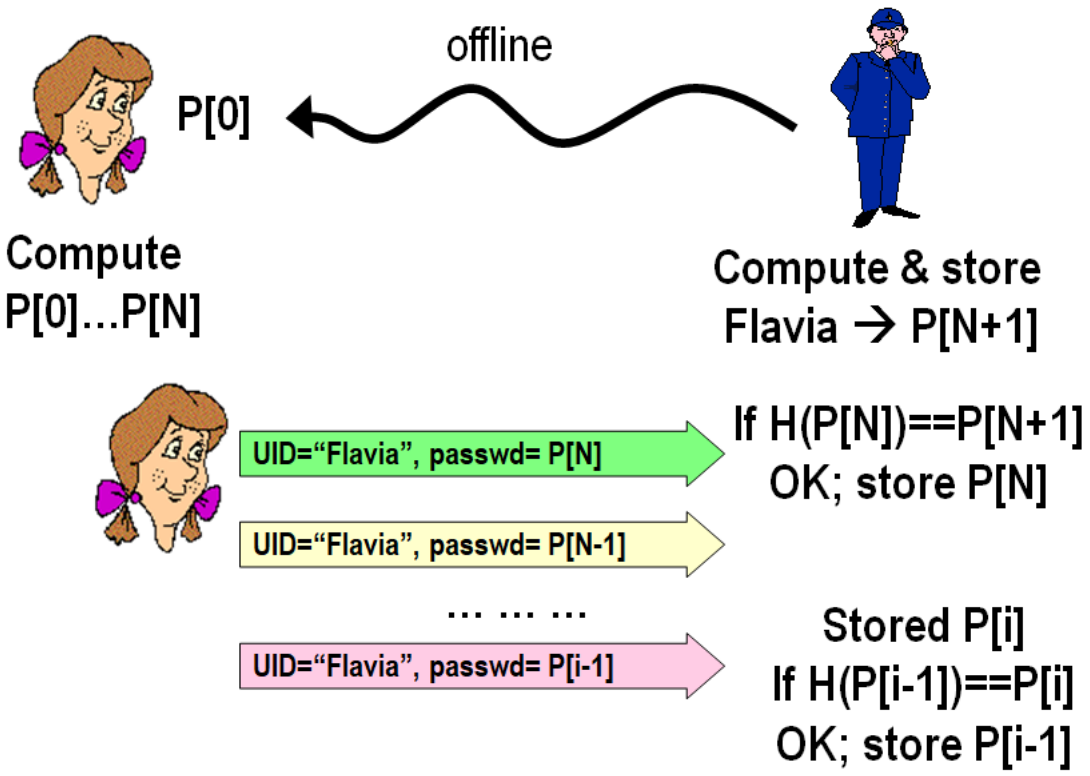
Remark. Il vantaggio di CHAP è che non deve mai trasmettere la password in chiaro. La funzione non invertibile che viene usata è una **hash function**.

Remark. Ricapitolando:

- **PRO**:
  - CHAP protegge dai *replay attacks*, ma bisogna assicurarsi che la challenge non si ripeta mai.
  - Viene avviato dall'authenticator e quindi un attacker può trovare difficolta a convincerlo a mandare challenge diverse più volte;
  - **Challenges Ripetute**: l'autenticazione può essere ripetura durante il tempo di connessione a differenza del PAP che avviene solo una volta all'avvio. Così da verificare più volta l'effettiva identità dell'utente e quindi limita il tempo di esposizione ad ogni singolo attacco.
- **CONTRO**:
  - Il segreto deve essere disponibile in formato *plaintext* e quindi non è possibile utilizzare un database di password crittografate in modo irreversibile.

4.2.3 One-Time Password

Il one-time password (OTP) è una tecnica che differisce da one time pad. Si bada sull'idea di migliorare il PAP per avere avere una password diversa ad ogni successfull attempt; questo garantirebbe al sistema una resistanza ai replay attacks. L'idea è quella di memorizzare più password associate ad uno stesso utente, il che però appesantirebbe tropo il DB. Per ovviare al probelma dellappesantimento del DB si utilizzano le **hash chain**: catene di shash applicate ad una password di partenza. Questa catena garantisce quindi che la password cambi ogni volta; ogni volta verrà utilizzata una password diversa per entrare generata a partire da quella precedente. Il probme a di questo metodo è che un attacker che ascolta una password poi è in grado di calcolarsi le successive. Una soluzione è usare un hash chain inversa, dove si parte dall'n-esima password generata e si torna indietro. Questa soluzione sembra funzionare, ma estendendola su scala mondiale, ogni singolo utente richiederebbe il calcolo di n hash ne momento di inserimento della password.
Mandare le password al contrare va bene, mabisgno risolvere questo problema computazionale.



Durante la registrazione l'authenticator calcola tutte le password offline fino a $P[n+1]$ e si salva l'ultima. In questo modo quando l'utente vuole accedere fornisce $P[N]$, l'authenticator calcola la successiva applicando la funzione hash, e se corrisponde a quella memorizzata allora l'utente entra nel sistema. L'authenticator sovrascrive a quel punto $P[N+1]$ con $P[N]$ nel suo DB. La prossima v olta l'utente fornirà la password precedente e così via.
Questa soluzione prevede che ogni volta l'utente si calcoli offline sul posto fino al $P[N]$ desiderato. I questo modo effettivamente ad ogni autenticazione verrà eseguita solo una hash da parte del server. **Vantaggi**

- Ad ogni autenticazione viene eseguita una sola hash;
- Il DB memorizza un solo valore per utente;
- Robustezza contro attacchi servr-side poiché è impossibile prevedere le password precedenti;
- La password può essere trasmessa in chiaro;

Svantaggi

- Serve un grand valore di N per evitare di finire le password a disposizione;
- Vulnerabilità lato client che deve salvarsi il seed delle password o l'intero vettore;
- Possibilità di desincronizzazione: se per qualche motivo fallisce una ccesso, si perde la catenta, in quanto l'utente la volta dopo proverà ad entrare con $P[N-1]$ in giù, mentre l'authenticator si aspetta ancora $P[N]$

4.2.4 Autenticazione a due fattori

Questo tipo di autenticazione si ispira a one time password. Si ipotizza che sia client che server siano sicuri. Ci sono divers requisiti in questo sistema, come un on time authorization token, generato su un device differente o ricevuto su un canale differente. Oltre ciò, il tutto deve essere human friendly cioè si utilizza un hash che trancono le key ad un massimo di 6/8 digit per essere memorizzata.
Ci sono due protocolli:HOTP e TOTP.

- **HOTP-HMAC-Base OTP**: Questo protocollo si basa sull'idea di utilizzare un contatore N. Sul server e sul client viene memorizzata il segreto k di un utente. Permettendo di fare:

$$P(N) = H(K, N)$$

Questo fa si che non si tratta più di una hash chain (dato che non si applica la hash ripetutamente sullo stesso digest di una password di partenza), ma ogni volta su un bit stream differente. Se un hacker dovesse conoscere

$P(N) = H(K, N)$ non può calcolare quello successivo. Inoltre, $P(N)$ può essere calcolata asincronamente senza calcoli ogni volta che vi vuole, dato che occorre avere la chiave di partenza.

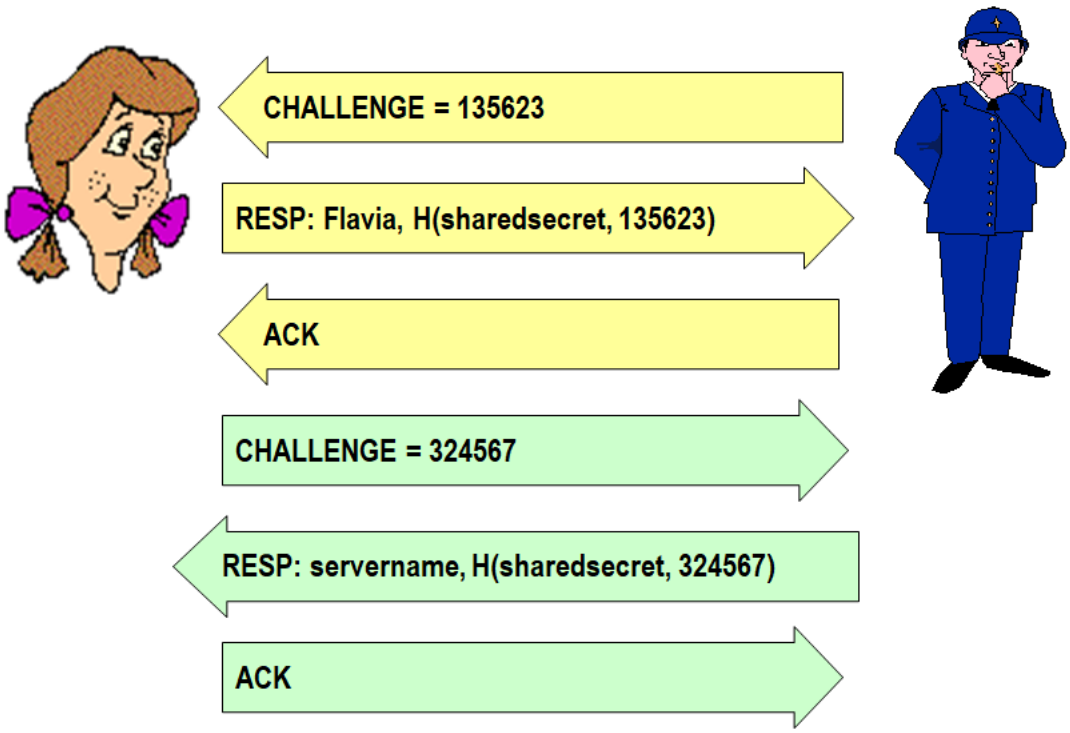
$$HOTP(K, C) = Trunce(HMAC - SHA - 256(K, C))$$

- **TOTP0-Time Based OTP**: ogni periodo di tempo viene cambiato il token N per calcolare la password, e non ad ogni accesso come in HOTP. GLia ttacchi a questo tipo di protocollo si basano sul fatto che l'haver possa manipolare il tempo della macchina, così da decidere lui come farlo scorrere per il cambio di token.

$$TOTP = HOTP(K, T) \quad T = (CurrentUnixTime - T0)/X$$

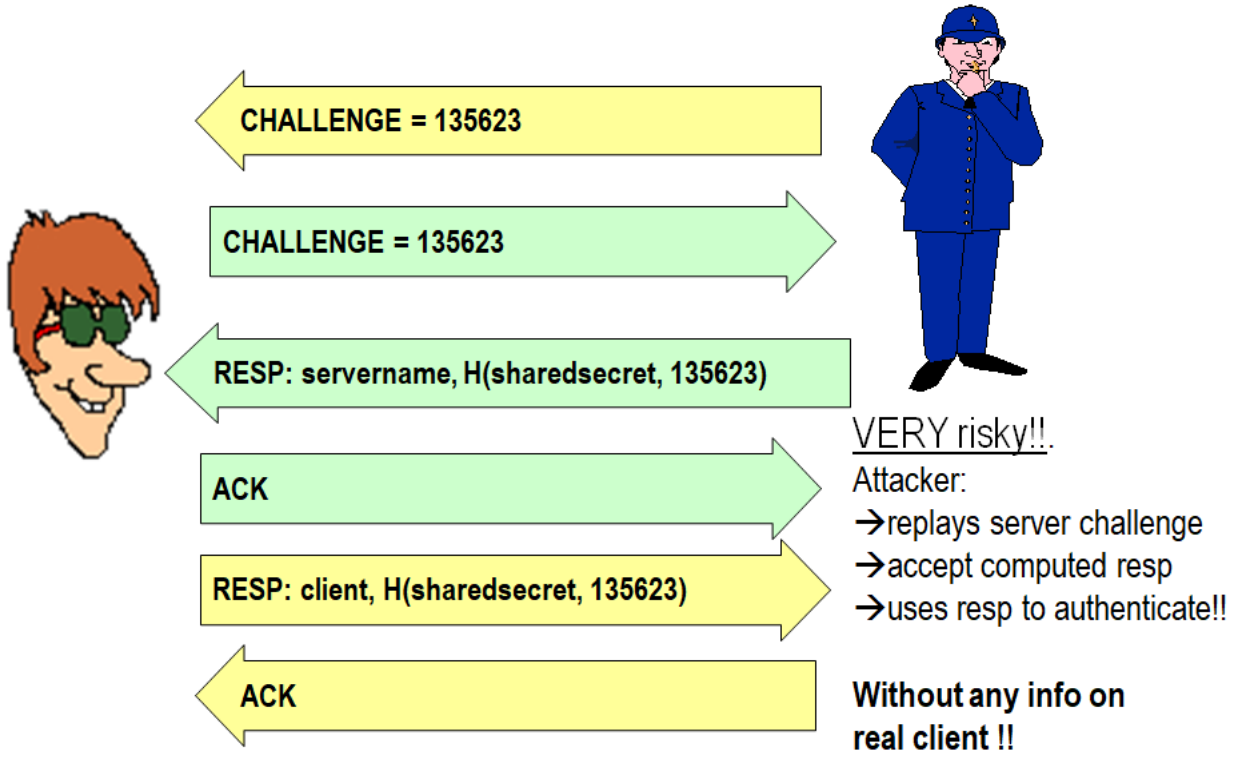
4.2.5 CHAP: Mutual Authentication

L'idea di **CHAP** è quella di provare ad adattare CHAP ad una autenticazione utente server e viceversa. In generale, si ha che:

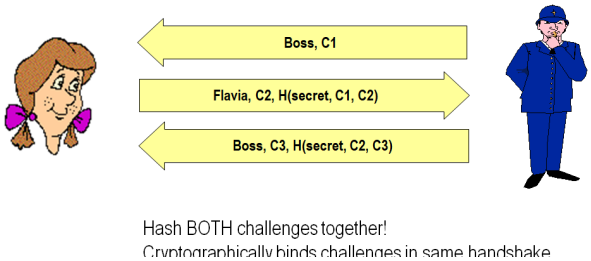


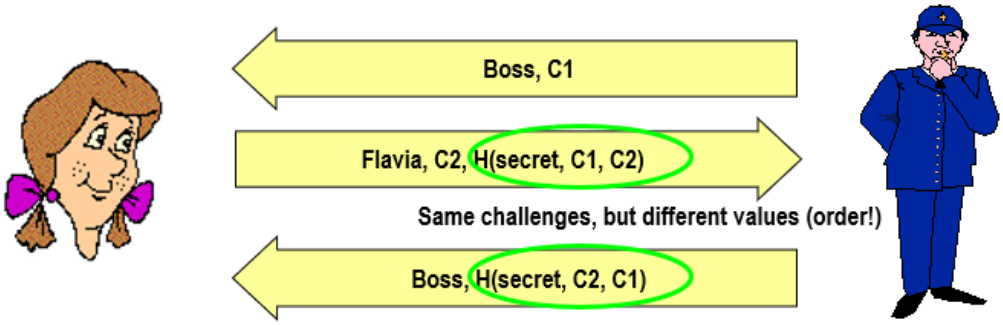
- Arriva la challenge, l'utente risponde con $ID, H(sharedsecrete, challeng)$, l'authenticator manda un ACK;
- A questo punto l'utente rimande una challenge nuova; l'authenticator risponde con $ServerName, H(sharedsecre, challenge$ L'utente manda l'ACK.

Siccome non è specificato l'ordine dei messaggi, un hacker può rispondere alla challenge con la stessa challenge, così da ricevere come risposta dall'access point la stessa che dovrebbe dare lui per entrare.



Questo può accadere perché le due autenticazioni sono unilaterali e non è specificato né in che ordine vanno mandate, né che non possono essere uguali alle challenge. Oltre ciò, il sistema è soggetto al **reflection attack**: l'attacker si finge acce point e si fa inviare da un utente $ID, C2eH(secret, C1)$. L'attacker fa finta di non sentirlo e rimanda la challenge appena inviatagli, aggirando di conseguenza i controlli.





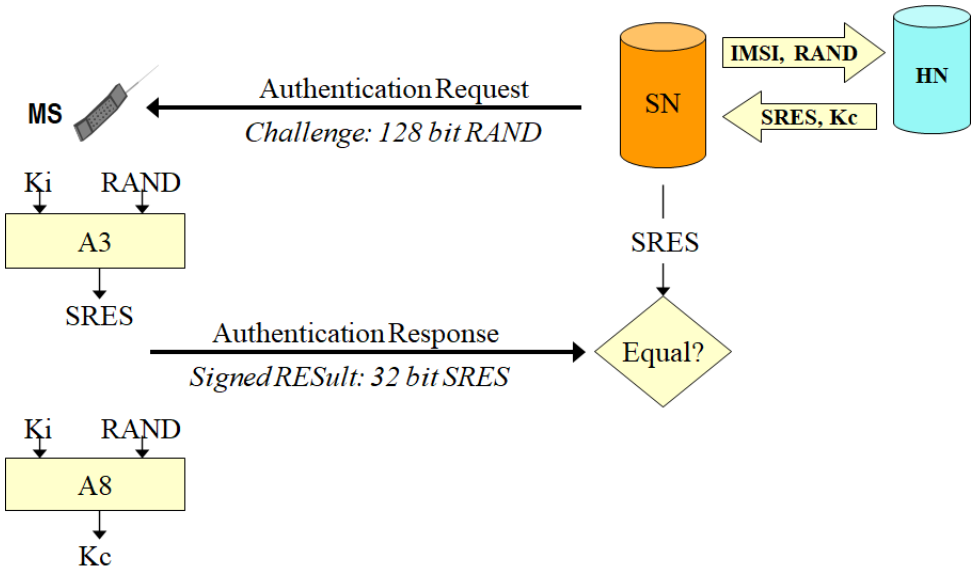
Per **prevenire il reflection attack**, si può procedere nella seguente maniera. Se l'authenticator invia una challenge all'utente, e quest'ultimo propone una challenge diversa che viene inclusa nella funziona hash insieme alla stessa challenge che ha ricevuto. A questo punto l'authenticator rimanda la funzionee hash delle challenge in ordine invero. Quindi, un hacker può sentire le diverse challenge, ma senza aver condiviso precedentemente il segreto e quindi non può calcolare la funzione hash. Il problema di CHAP, che è stato ideato per autenticazione single side, è che se viene usato in due direzioni, la loro indipendenza espone rischi al sistema. Quindi, occorre renderle indipendenti crittografando due challenge.

4.3 Chellenge-Response Authentication in Wireless Cellular Systems

La comunicazione in ambiente wireless di sistemi cellulare si basa sulla comunicazione di tre entità:

- **UE**(User Equipment): cipè il mezzo che permette all'utente di connettersi alla rete. In questo caso è la SIM;
- **Serving Network**(SN): è la rete diversa dal provider della SIM che interpella il provider(Home Network) per effettuare la connessione;
- **Home Network**(HN): è la rete del gestore di rete alla quale un utente si autentica.

4.3.1 Autenticazione in 2G(GSM)

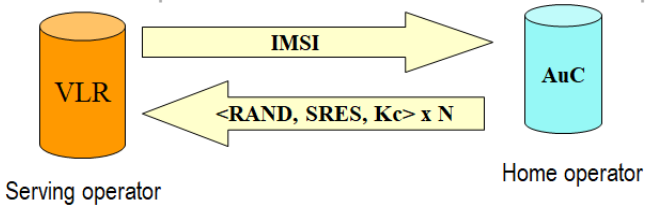


Per instaurare la connessione si seguono i seguenti passi:

1. L'user che si vuole connettere si identifica;
2. Il server network manda l'**IMSI**(subscriber identificator, identificatore dell'utente) e **RAND** challenge all'Home Network;
3. L'Home Network risponde con SRES ($H(RAND, K_i)$ **tramite funzione hash A3**) e K_c(**encryption key**) utilizzata nella sessione e generata dinamicamente ogni volta che si vuole autenticare tramite la funzione hash A(a partire da K_i e $RAND$;
4. Ora che il Service Network è entrato a conoscenza delle informazioni necessarie allo svolgimento della sessione, manda una authentication request alla MS con una challnge(RAND) di 128 bit;
5. A questo punto la Mobile Sim crea 32 bit (SRES) a partire dai 128 di challenge che gli sono arrivati facendo ausiliao della funzione hash A3(conosciuta solo dalla sim e dalla Home Network, fornite dal provider). Questo digest di 32 bit è ottenuto a partire da K_i e $RAND$;
6. Il Service Network conrolla se gli SRES corrispondono e, in caso affermativo, garantisce l'accesso.

Remark. Il punto di forza è che K_i(Identity key- chiave dell'utente) resta nell'Home Network e non esce dal dominio verse il Service Network,grazie alla non invertibilità delle funzioni hash.

4.3.2 TRIPLETS vettori di dati, utili per autenticazioni effettuare molteplici volte



Un concetto importante nei sistemi cellulare è l'utilizzo di vettori di dati. Si prendono due entità in parti differenti del mondo:

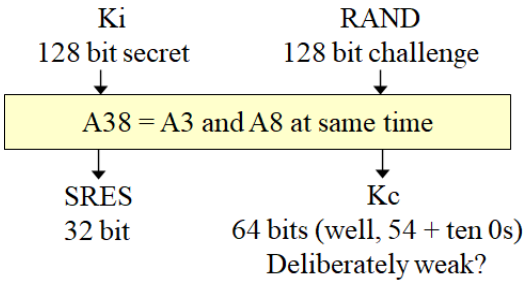
- **VLR**: Serving Operator;

- **AuC**: Home Operator;

Questa volta, VLR invia solo l'IMSI senza la RAND. AuC risponde con un insieme di vettori di tipo $[Challenge, SRES, K_c]$. la risposta è quindi formata da N triplette di questo tipo. Il vantaggio è che le performance aumentano dato che si mantengono delle triplete in più per le prossime connessioni; inoltre, è che questo mette tutta la sicurezza nelle mani dell'home operator poiché è quest'ultima che gestisce le *nonce*. Quindi, si evita che qualunque hacker possa provare ad attaccare le *nonce* cercando di creare o manipolarle.

4.3.3 2G:Problematiche

Dopo l'applicazione di A38(utilizzo A3 e A8 in contemporanea), viene dato K_i e $RAND$ di 128 bit, da cui si ottengono SRES,da 32 bit, e K_c, da 64 bit. Il problema fu che in realtà erano 54 bit poiché alla fine non vi erano sempre 10 zeri. Quindi, si perdevano 2^{10} combinazioni di sicurezza.

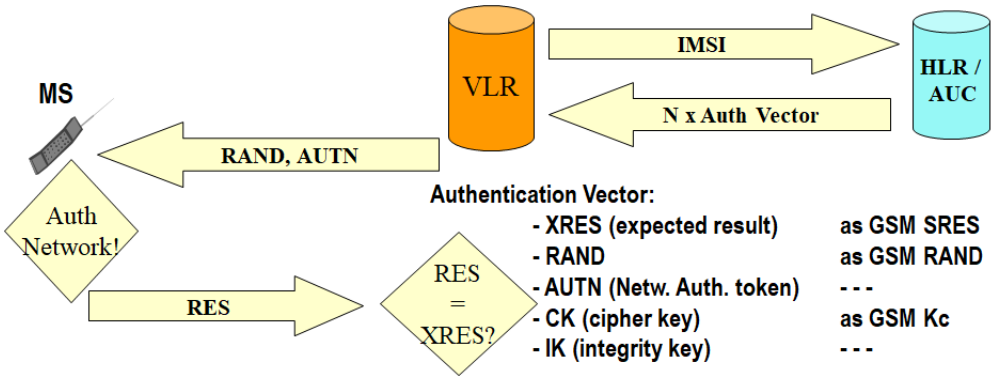


Ogni operatore doveva scegliere come creare la sua funzione di autenticazione: poteva realizzare la sua specifica, oppure utilizzarne una già creata da qualcun altro. Tuttavia, in quel periodo si volle utilizzare COMP128. Questo era il concetto di **security by obscurity**: si credeva *erroneamente* che nascondere il segreto fosse sicuro.

Remark.
- Security by obscurity è fallace. La cosa giusta da fare è lasciar fare algoritmi di sicurezza a chi ne capisce;
- Piuttosto che fare un algoritmo privato e chiuso autonomamente, risulta una scelta migliore renderlo aperto e pubblico così che chi è componente può testarlo evidenziandone difetti e pregi;
- Una ulteriore problematica del 2G è l'assenza di mutual authentication. Quindi era possibile fingersi una BS (Base Station) fittizia.

4.3.4 3G(UMTS),4G(LTE),5G

I protocolli di rete 3G,4G,5G utilizzano il mutual authentication protocol e gli algoritmi sono pubblici (al contrario de 2G).



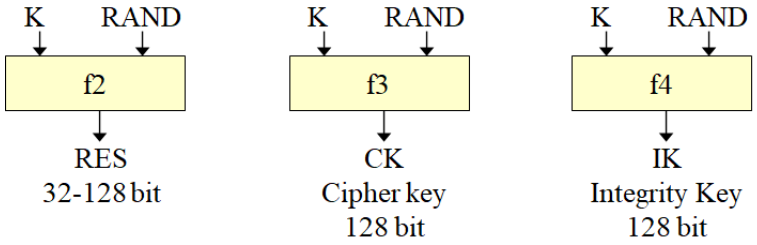
In qesti nuovi sistemi compaiono nuovamente MS, VLR, AUC ed il procedimento è il seguente:

1. Un MS si connette;
2. VLR manda IMSI all'Home Network;
3. HLR manda al VLR delle quintuole che contengono:
   - RAND(Challenge);
   - XRES: il risultato atteso della CHAP type authentication;

- CK: cipher key;
- IK: integrity key;
- AUTN:*network authentication token*, serve a pìrovare che la rete sia autentica per la mutual authentication;

4. VLR rimanda quindi a MS: RAND, AUTN;
5. MS risponde con res per provare che è autentico una volta che riceve un AUTN.

MS Authentication

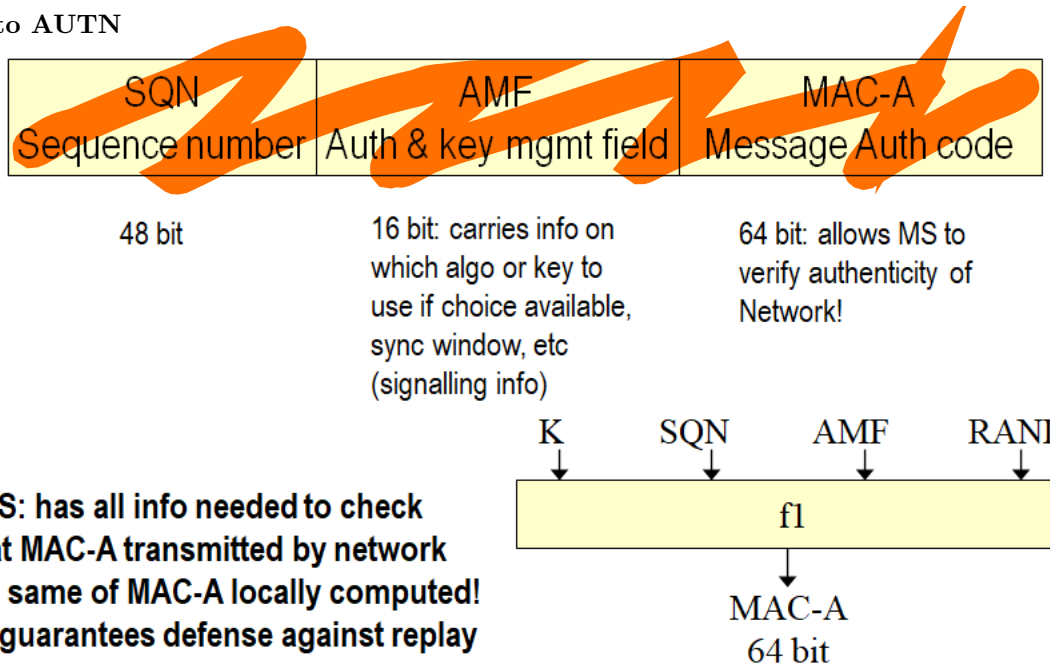


VLR manda il token che assicura l'autenticità della rete sulla base di una challenge che lui stesso ha generato. Secondo le regole visto fino ad ora non si può provare di essere autentici a meno che non sia l'opponente a creare la challenge.

4.3.5 Network Authentication - SEQ as Nonce

La nonce può essere una challenge, una sequenza o un time stamp. Si prende per questo esempio un time stamp ed ipotizziamo che MS e VLR siano in possesso di un GPS e che il tempo non sia attaccabile. Se MS conosce il tempo, in realtà non serve nel time stamp mandare la nonce poiché è già a conoscenza di che ore sono. Considerando che conosce già il riferimento temporare gli basta la risposta di VLR. VLR, quindi, provvede a rispondere diretto con $H(timestamp, secret)$. Usando quindi un secure time reference, è possibile effettuare mutual authentication con un solo messaggio.
Nel caso in cui volessimo fare la stessa cosa con una sequenza di numeri:
MS manda una nonce a VLR ed esso risponde con $AUTH(K, NONCE)$. Vi è un contatore nella sim chiamato SQN-MS che aumenta ad ogni autenticazione; così quando il VLR manda la sua richiesta di identificazione ad MS, deve anche mettere quanto gli risulta che sia il contatore. In questo modo se MS riceve un contatore minore sa che non si deve fidare.SQN-MS è sincronizzato con SQN-HE quindi, un altor contatore nella home network. Alcuni attacchi si basano proprio sulla desincronizzazione di questi due. In particolare: la sim dell'utente conosce SQN MS; quando VLS manda il suo SQN che proviene da HN, l'utente deve controllare che $SQN = SQN - MS + 1$. Se corrisponde, allora può fidarsi della rete.
Anche in questo caso con un solo messaggio ci si può assicure che la rete è sicura.

4.3.6 Formato AUTN



SQN è il sequence number formato da 48 bit. Attaccati ci sono 16 bit di AMF che rappresenzato quale algoritmo è stato utilizzato e quale finestra di sincronizzazione. Infine, ci sono gli utlimi 64 bit di MAC-A che sono costituiti nel seguente modo: si mettono insieme in una funzione di criptaggio f1 SQN,RAND,K,AMF(informzioni dell'algoritmo). Ed infine, tramite gli utlimi 64 bit, MS può testare l'autenticità della rete.

4.3.7 Sistema Sfruttato 3G e 4G

La prima volta che un utente si connette alla VLS, al MS viene inviato un TMSI(temporary MS identifier) che viene allcoato dinamicamente alla prima connessione. Alla prossima autenticazione non verrà utilizzato l'IMSI, ma quel TMSI. Esso è importante perché cambia ogni volta e quindi non è tracciabile. Siccome, vedendo la SQN, un eavesdropper potrebbe discriminare e rintraccia l'utente si masschera con una chiave anonima, si esegue SQN xor AK. L'AK di 48bit viene generato tramite l'applicazione di una funzione f5 insieme ad un valore randomico ed una volta inviato il pacchetto, per ritrovare SQN viene rieseguito lo xor con AK. Altro passaggio importante è l'esecuzione della funzione f4(SQN,K,RAND,AMF)=MAC-A; verrà controllato con MAC-A inviato precedentemente.

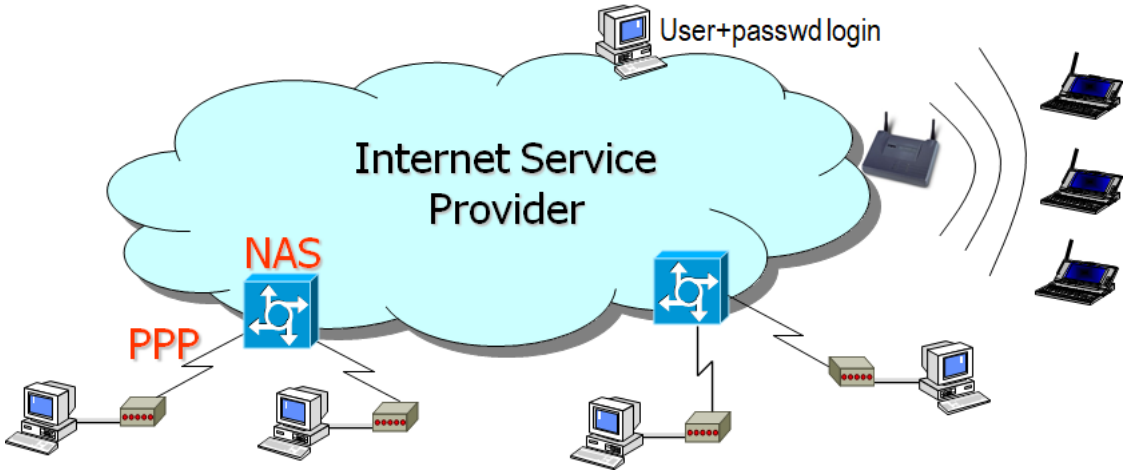
4.3.8 Generazione chiavi da un segreto

Per generare infinite chiavi da un solo segreto, si crea un global secret chiamato S, unico. Ogni volta che arriva un nuovo user crea H. In questo modo si ha la sicurezza che nessun utente nel sistema abbia una stessa password, e analogamente quando ne arriva uno nuovo gli viene affidata la sua K_i senza eseguire contorlli su tutte quelle già fornite.

5 RADIUS

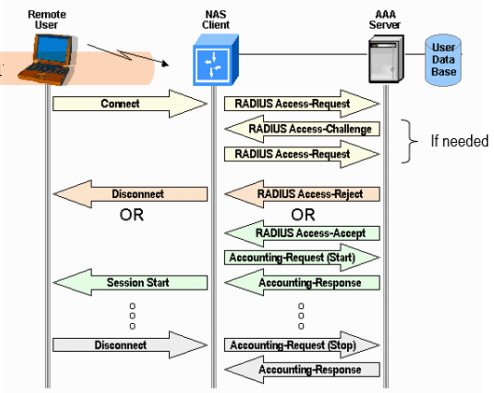
RADIUS è un protocollo client-server di livello applicativo che utilizza UDP per gestire su larga scala di Access Point eterogenee. In particolare, deve gestire:

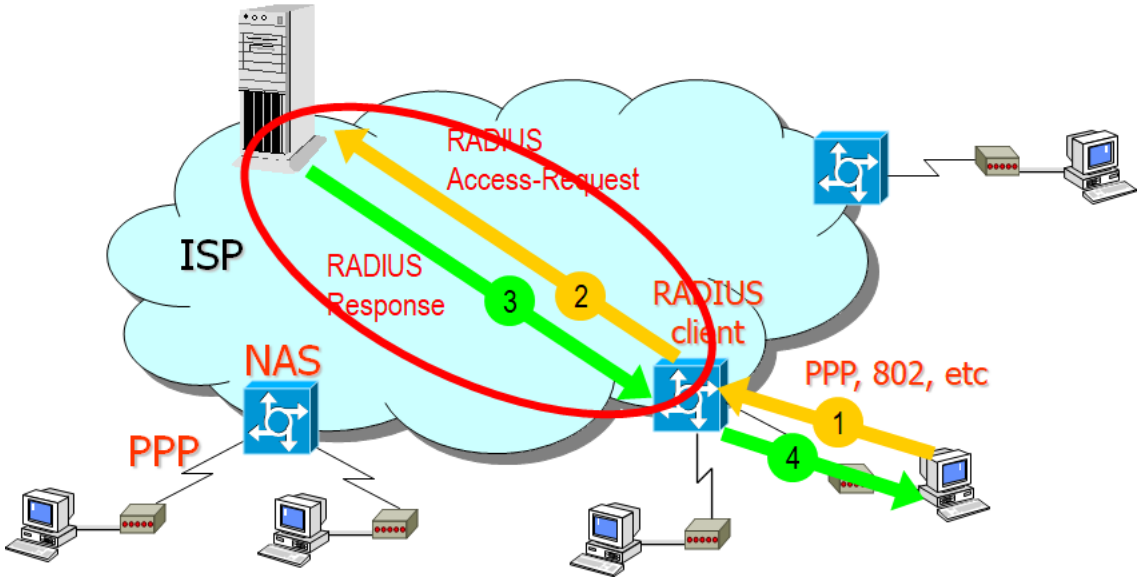
- molteplici NAS (*client*);
- molteplici servizi di accesso;
- Autenticazione e assegnazione della configurazione specifica del servizio.



La solutione ottima è ottenuta gestendo un DB per ogni singolo utente. In particolare:

1. L'utente invia gli attributi dell'autenticazione al NAS. **Assenza di RADIUS**;
2. Il NAS inoltra la richiesta di accesso al server RADIUS;
3. Il server RADIUS risponde OK,NO, Challenge peer alcuni tipi di au
4. NAS notifica l'utente se può entrare o meno. **assenza di radius**;





Remark.
- **PPP:**=point to point protocol;
- RADIUS server (AuC) e client (**NON** RADIUS Client)conosce il vero segreto;

5.1 RADIUS: AAA Protocol

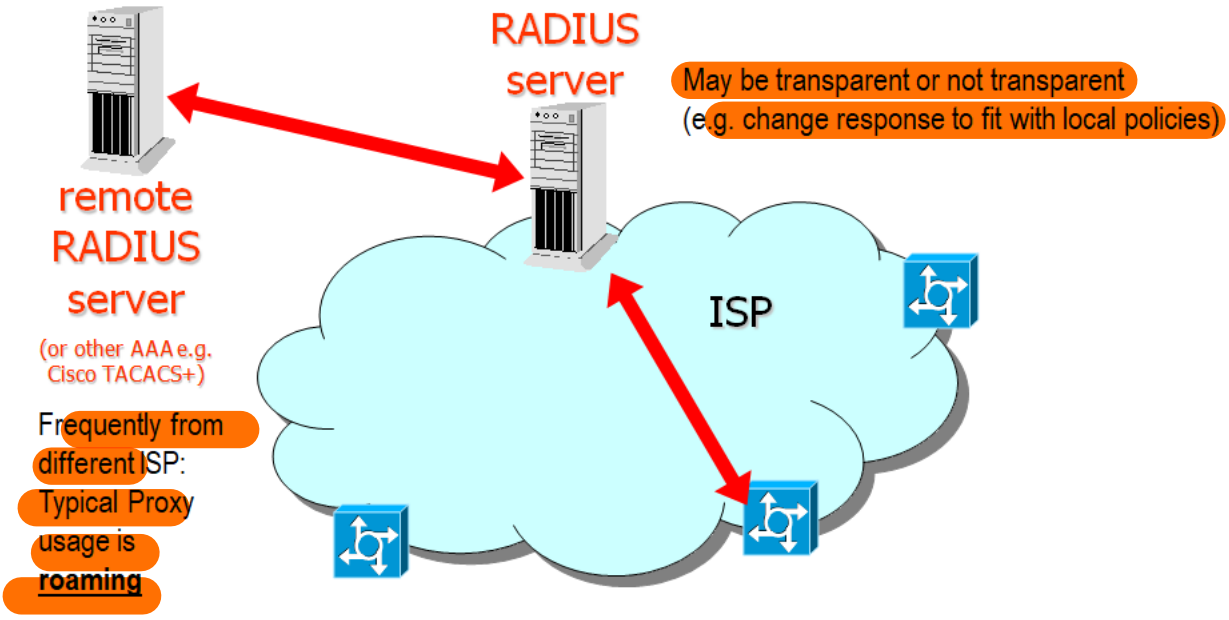
Il protocollo Radius è detto protocollo AAA poiché permette di centralizzare:

- **Autenticazione**: verifica di chi dico di essere;
- **Autorizzazione**: verifica se posseggo i privilegi per accedere ad un servizio anche se sono autenticato. Quindi è una proprietà totalmente scorrelata rispetto all'autenticazione:RADIUS fornisce entrambe le funzionalità;
- **Accounting**: permette di monitorare le attività dell'utente in quel preciso momento. Per esempio il numero di Byte trasmessi o la fatturazione.

5.2 RADIUS-Protocollo Client-Server

Il protocollo RADIUS è un protocollo client-server RADIUS: la parte client viene rappresentata dal NAS che inizia il servizio e a cui l'utente finale (dispositivo) chiede l'autenticazione/autorizzazione/accounting. Inoltre, è un protocollo che si basa su UDP/IP sulla porta server 1812 ed eventualmente il server può agire come proxy.

5.3 Proxy Operation



5.4 Architettura RADIUS

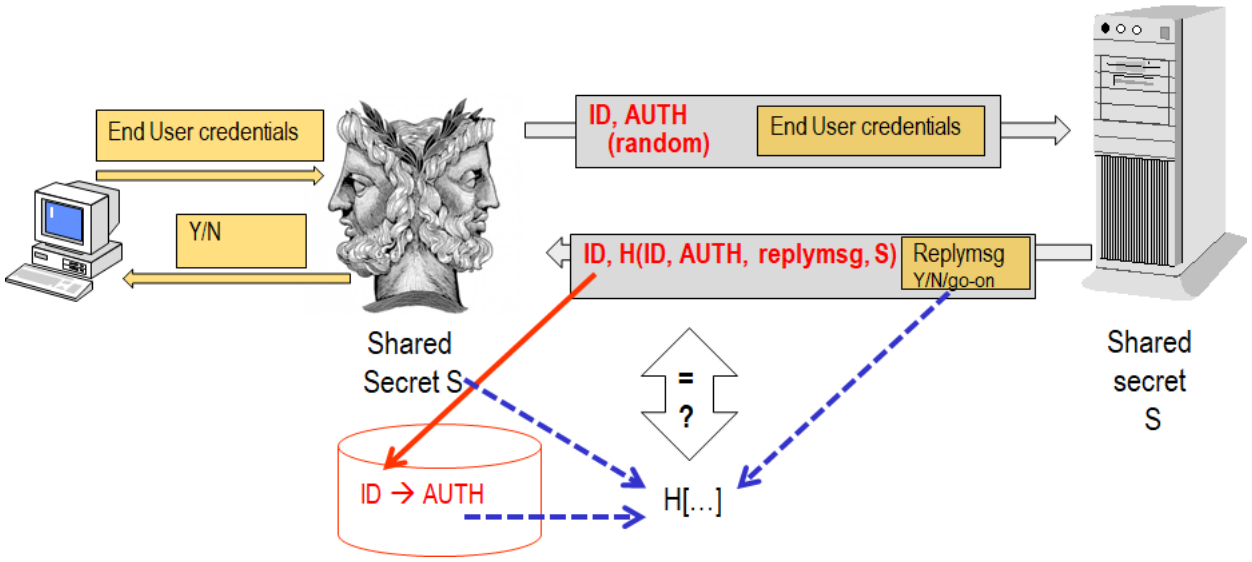
- Applicazione server RADIUS;
- Database utente registrato: ~~yametekudasai~~ per ogni voce nomeutente contiene almento:
  - Informazioni di autenticazione (segreti);
  - Metodi di autenticazione:
  - Attributi di autorizzazione;
  - ~~Database LolloDB della sneffa di Tor Vergata;~~
- Database dei client:
  - Client autorizzati a comunicare con il server;
  - **Client Radius≠END-USER**
- Database accounting:
  - Utilizzato frequentemente per autenticazione;

5.5 Caratteristiche di sicurezza RADIUS

- **Risposta** autenticata per pacchetto:
  - si utilizza un segreto condiviso;
  - Assenza di trasmissione del segreto, come nel CHAP;
  - La risposta viene autenticata;
  - è hash-based e non HMAC-based;
  - si utilizza come funzione hash MD5;
  - Chiave condivisa a bassa entorpia;
- Trasmissione crittografata della password utente. La chiave viene utilizzata per l'autenticazione.

5.6 RADIUS authenticad reply

Occorre definire un modello di attacco che verrà riportato di seguito. L'obiettivo è quello di usufruire di un servizio anche se non se ne ha la facoltà. Inanzitutto sappiamo che il vero segreto lo conoscono l'end-user e il server RADIUS; mentre il client NAS e il server RADIUS condividono lo stesso segreto S.



- Il **NAS** parla con:
  - End-USer: invia le sue credenziali;
  - Server RADIUS.

I passaggi sono i seguenti:

1. L'user invia le credenziali al NAS;
2. crea un *RADIUS alfy packet* contenente le credenziali dell'utente contenente ID e AUTH(Challenge randomica);
3. il client NAS manda il paccheto al server che a sua volta risponde con $ID, H(ID, AUTH, replymsg, S)$ ed il messaggio di risposta che può essere $Y/N/go\text{-}on$.

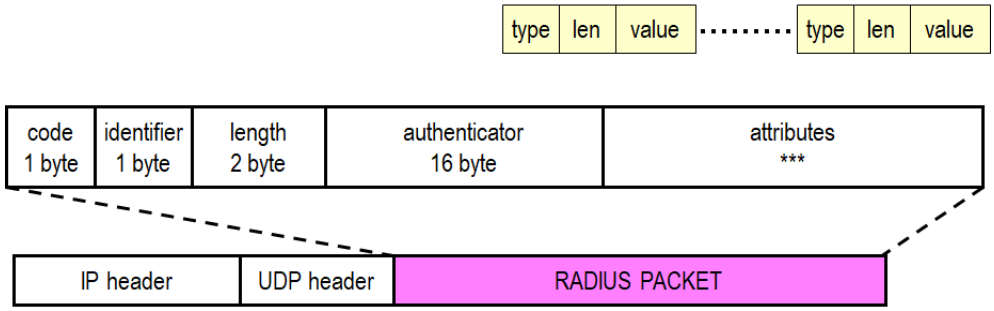
Il campo ID serve a riconosce tramite un numero univoco il pacchetto in questione.

Remark. Questo schema può essere considerato come un challenge-response. In cui:

- Challenge è l'autenticatore della richiesta;
- La risposta include (e quindi convalida) il messaggio di risposta.

5.7 Formato del pacchetto RADIUS

Il pacchetto RADIUS è formato da:



- **codice**: rappresenta il tipo di pacchetto RADIUS.

Code (dec)	Packet
1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge

- **Identificatore**: permettere di trovare una corrispondenza tra le richieste e le risposte;
- **Lunghezza**: lunghezza del pacchetto. Minimo 20 massimo 4096;
- **Autenticatore**: utilizzato per autenticare le risposte dal server e usati anche come nonce per la cifratura della password;
- **attributi**: campo informativo estensibile di circa 256 tipi. Il loro ordine non conta e si possono ripetere.

5.8 Authentication Field

- **Access-Request(C→ S)**: 16 byte generati casualmente. Quindi, imprevedibile e unico per evitare il replay attack;
- **Pacchetto di risposta(S→ C Accept/Reject/Challenge)**:
  - ID della richiesta;
  - Autenticatore della richiesta;
  - Segreto condiviso;
  - Informazioni sulla rispsota del pacchetto: response packet è firmato. In caso contrario la risposta del server potrebbe essere falsificata.
- $MD5(CODE|ID|Length|RequestAuth|Attributes|Secret)$

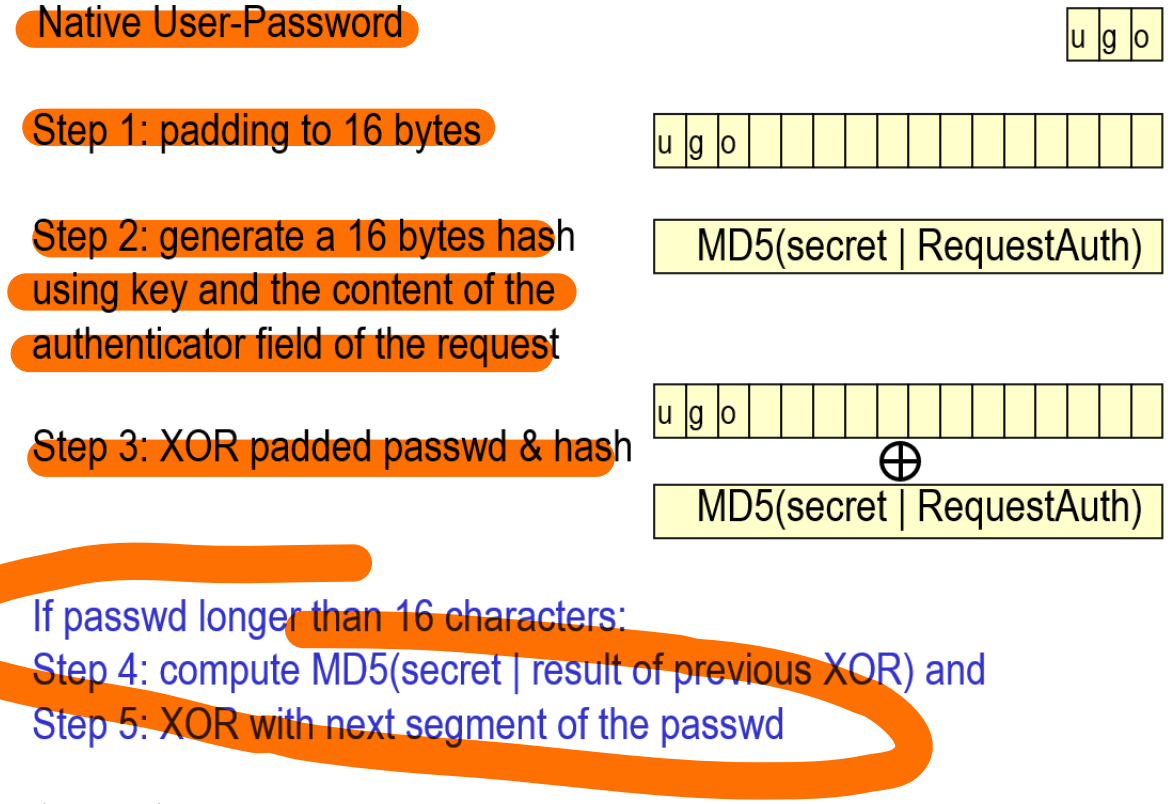
5.9 Access-Request

L'**access-request** tipicamente contiene:

- L'username**obbligatorio**: è la chiave di ricerca per accedere al database dell'utente;
- Password:
  - User Password;
  - CHAP-password (se si utilizza CHAP);
- Identificatore dle client RADIUS:
  - NAS-IP o NAS-Identifier: utile per accedere solo in un sottoinsieme di NAS;
- Identificatore della porta al quale l'utente sta accedendo:
  - NAS-Port se il NAS ha le porte. Pososno essere logiche (es.WiFi) o fisiche (es. Dial Up).

Vi è la possibilità che l'utente non sia abilitato all'accesso su specifiche porte.

5.10 Password Encryption



5.11 Access-Accept

RADIUS come ulteriore tecnica di protezione utilizza il password encryption. In particolare, l'utente invia al NAS username e password e quest'ultimo lo invia al server. Dato che la rete fra il NAS e il server RADIUS non è sicura occorre criptare il messaggio.
A tale scopo, si prende la password e la si padda fino a 16 byte, si prende questo blocco, lo si affianca al NONCE e si applica la funzione di hash MD5. Al risultato di questa operazione si applica l'operazione di XOR con la password. Infin, il risultato dello XOR viene inserito nel campo attributi del pacchetto RADIUS ed inviato.
Nel caso in cui la password dell'utente sia pi+ lunga di 16 bytes, essa viene segmentata in blocchi di 16 bytes ciascuno (l'ultimo di questi blocchi, se minore di 16 byte, viene paddato fino al raggiungimento della dimensione richiesta). I singoli blocchi vengono quindi inseriti nella funzione hash M5D e si attua il procedimento canonico. La concatenzazione di questi risultati vengono inseriti nel campo attributo.

- Risposta positiva del server: allora le credenziali di autenticazione dell'utente sono valide;
- Contiene tutta la configurazione delle specifiche del servizio tra cui un attribut del tipo di servizio e ulteriori parametri.

5.12 Access-Reject

L'accesso può essere rifiutato per due ragioni principali:

- Autenticazione fallita;
- 1 o più attributi nella richiesta di accesso non sono stati considirati accettabili.

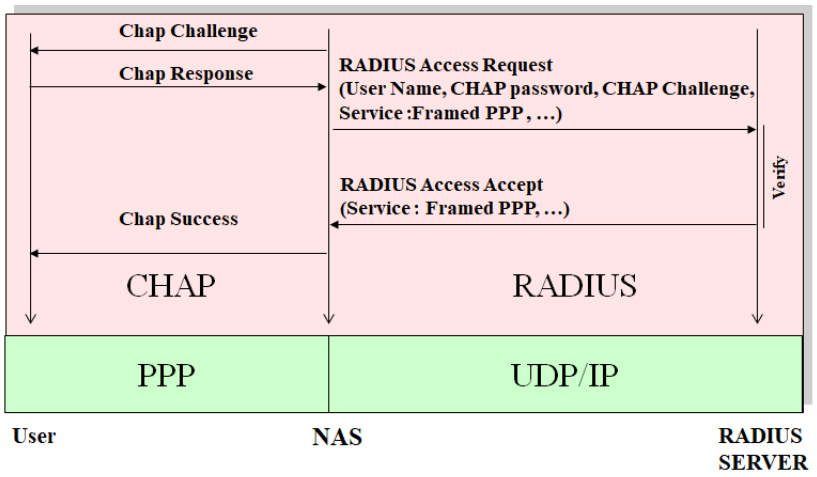
5.13 Access-Challenge

La challenge per l'accesso viene utilizzato ogni qual volta che il server vuole un'ulteriore risposta dell'utente e può contenere:

- 1 o più attributo di reply-message;
- Testo mostrato all'utente;
- Esplicita challenge di autenticazione.

Una Access-Challenge può essere il prompt di richiesta nome utente e password.
Inoltre, il client NAS collezione le risposte dall'utente e le invia in una una richiesta di accesso.

5.14 PPP CHAP supportato con RADIUS



Il collegamento punto punto **PPP** con CHAP può essere supportato con RADIUS. In particolare, la CHAP challenge viene generata localmente nel client NAS in assenza della password dell'utente. Inoltre, si invia la CHAP challenge con la risposta al server RADIUS. Quindi il client NAS, invia una **RADIUS Access-Request** al server RADIUS che dovrà verificare la correttezza delle credenziali fornitegli. Il Server RADIUS risponderà con un **Access-Accept** o **Access-Reject** in base alla validità o meno del pacchetto RADIUS. In caso affermativo, il client NAS notificherà all'utente che il PPP CHAP ha avuto esito positivo.

Remark. Nella comunicazione utente-NAS si utilizza PPP-CHAP.
Nella comunicazione NAS-RADIUS server si utilizza UDP/IP.

5.14.1 Attack on PPP CHAP RADIUS Standard

Lo schema di comunicazione appena proposto è lo standard effettivamente utilizzato nel mondo reale. Tuttavia, può essere attaccato con l'obiettivo di usufruire "a gratis" del servizio anche se non si ha l'autorizzazione nel farlo.
L'attacker si posizione in due punti della cominicazione: User-NAS e NAS-RADIUS Server. Questo permette di modificare le credenziali non valide che andrebbero al RADIUS Server (che porterebbe ad una risposta di tipo Access-Reject) in credenziali.
In particolare, si compone di una prima fase in cui **l'attacker si pone fra User e NAS** si osservano le credenziali utente valide, salvandosi il seguente pacchetto:

Numero pacchetto	AUTH	Request	CHAP reply	CHAP Response

Nella seconda fase, l'attacker si finge User,manda richiesta al NAS per essere autenticato. Il NAS invia la chap Challenge. A questo punto, l'attacker risponderà con credenziali arbitrarie e il NAS, una volta ricevuta la CHAP response, le invia al RADIUS Server. Tuttavia, **l'attacker si pone tra NAS e RADIUS Server**:lascia invariati i

campi che vanno dal numero pacchetto al request (*se vengono modificate, il RADIUS Server si accorgerebbe dell'attacco che sta avvenendo*), ma modifica il pacchetto RADIUS con le credenziali ottenute nella prima fase.Il server, quindi, verifica la validità delle credenziali, invia al NAS un RADIUS Access-Accept. Il NAS, una volta ricevuta risposta dal server notifica l'utente l'avvenuta autenticazione.
L'attacker entra nel servizio "a gratis".

5.14.2 Risuluzione

Un modo per proteggersi da questo attacco è quello di autenticare tutti i campi del pacchetto o legare ogni richiesta di autenticazione alla risposta.

Remark. Come il prof. Bianchi insegna, si potrebbe risolvere utilizzando una copia della risposta nel RADIUS Client (NAS) ed inviare al RADIUS Server l'hash di $H(ID, AUTH, Replymsg, copia, S)$.

Remark. L'ideale è quello applicare una funzione di hash a tutto.

5.15 RADIUS debolezze di sicurezza

RADIUS è sensibile a message sniffing, poiché non vi è confidezialità, a message modification, poiché non c'è autenticazione per le richieste di accesso: ognuno può modificarne e crearne uno nuovo.

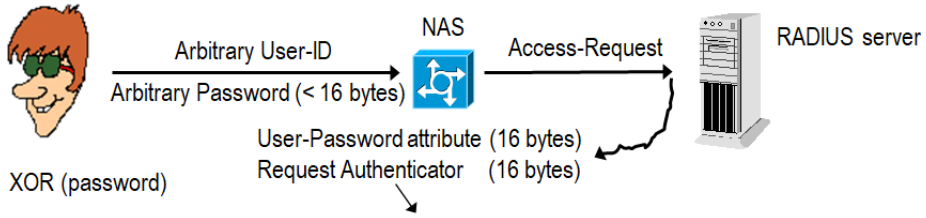
5.15.1 Message-Authenticator

L'autenticazione per le richieste di accesso può essere ottenuta tramite:

- **Message Authenticator**: per fornire la protezione dell'integrità per i messaggi di richiesta di accesso, viene aggiunto un tag a fine del messaggio di autenticazione detto *authenticator* tra gli attributi del request packet. In aggiunta, quando si protegge il messaggio di richiesta di accesso con Message Authenticatori, si calcola un hash dell'intero pacchetto usando il segreto condiviso che condivide con il server RADIUS:$HMAC_MD5(Whole\ Packet) = HMAC - MD5(type|ID|len|RequestAuth|Attributes)$
- **EAP(Extensible Authentication Protocol)** Si tratta di un approccio arbitrario di autenticazione, ovvero di tecniche aggiuntive messe per la sicurezza di sistemi a cura di chi li realizza.

5.15.2 Attack Shared Secret

Le password scelte dagli utenti sono spesso a bassa entropia e spesso se ne utilizza una per tutte le reti (es. Fonera modem basati su RADIUS). Per risolvere questo problema si prende un qualcosa di unico (che può essere un dato fisico di un device) e si fa in modo che il secret sia $HMAC(Secret, Dato\ Unico\ SEGRETO)$. Questa soluzione risulta scalabile e quindi ampiamente utilizzabile per le richieste di autenticazione. Un tipico attacco al secret viene offerto dai cosiddetti **dictionary attack**.



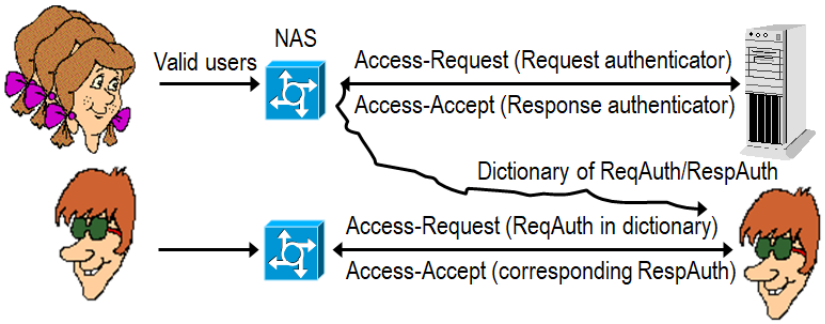
Un attacker prende l'id di un utente, si presenta al NAS con una password a caso e ascolta la richiesta con l'obiettivo di capire quale elemento random è stato generato. Ascoltando la risposta acquisisce l'MD5 associato e, conoscendo la coppia richiesta-risposta, può entrare nel sistema.In particolare la richiesta gli fornisce la variabile causale ReqAUTH e la risposta RespAUTH (MD5). Questo accade perché l'attacker riesegue l'operazione di XOR con la sua password ottenendo l'MD5 della coppia. Quindi, conoscendo la challenge all'interno di MD5, l'unica cosa che gli rimane è il secret che otterrà effettuando un brute force attack. In realtà, non si necessità di una coppia, ma solo di una richiesta valida con la password ell'utente.
Per evitare questo scenario, server un autenticatore di richieste unico e per realizzarlo si necessitano di 16 byte casuali ottenuti in questi modi:

- Attaccare fra loro diverse unità da 4 bytes pseudo-random:
- prendere una unità pseudo rando e paddarla finoa 16 bytes;
- effettuare MD5 dei 4 byte pseudo-random;

Valutando il ciclo di valori pseudo-random prodotti, mi permette di non avere ripetizioni. Tuttavia, il primo metodo è la peggiore soluzionepoiché generare 4 sequenze per volta fa in modo che is ricrea una sequenza già vista quattro volte più velocimente: si diminuisci il ciclo di ripetizione; il secondo e terzo hanno la stessa probabilità di ripetizione poiché si tratta sempre di 4 bit casuali che sono soggetti al paradosso del compleanno, ma comunque sempre migliori del primo.

5.15.3 Replay Attack

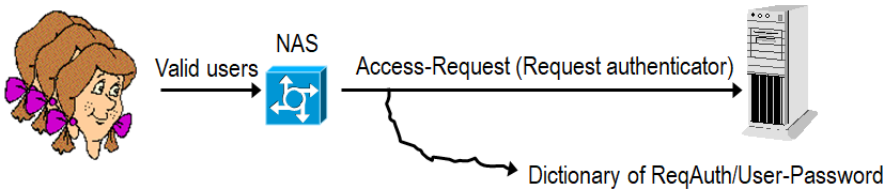
Nei replay attack in RADIUS, l'attacker si pone l'obiettivo di farsi autenticare/autorizzare senza una password valida.



L'attacker può prendere la password di un utente tramite eavesdropping, cioè l'ascolto del traffico tra NAS e RADIUS Server. Così facendo, può costruire un dizionario ed effettuare un *dictionary attack* per "breakkare" il cipher. Una volta conclusa questa fase, l'attacker effettua il replay attack al server con un login valido.

Remark. Il replay attack è un metodo di sfruttare i pacchetti catturati o reinviati all'utente causando un comportamento imprevisto o non voluto dal server. Nel caso in cui il server non riconosce il riuso dei dati e accetta ripetutamente i pacchetti trasmessi, l'attacco ha successo.

5.15.4 Attack User Password

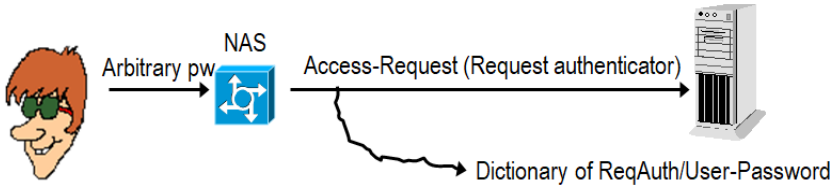


In questo tipo di attacco, l'attacker monitora passivamente il traffico di rete del NAS verso il RADIUS Server per costruire un dizionario compost da **request authenticatorReqAuth** a cui associa gli attributi della password utente. Assumendo che l'attacker può provare ad autenticarsi con una password nota ed infine catturare il pacchetto di Access-Request, si può effettuare l'operazione di XOR della porzione protetta della password utente con la password fornita al client NAS:

$$(USR-PWD1) \bigoplus (USR-PWD2) = (PWD1 \bigoplus MD5(secret, ReqAuth)) \bigoplus (PWD2 \bigoplus MD5(secret, ReqAuth))$$

$$= PWD1 \bigoplus PWD2$$

Quindi, l'attacker può lanciare un brute-force attack offline per scoprire il segreto condiviso S. Un'altra alternativa è la seguente:



In questo scenario, l'attacker invia attivamente delle password arbitario per formare un dizionario di password note e Access-Request. Quindi, se il server non impone un limite alle autenticazioni dell'utente, permette all'attacker di effettuare una ricerca esaustiva online per la password utente.

5.15.5 Conclusioni

Riassumendo, il protocollo RADIUS **non** è sicuro poiché:

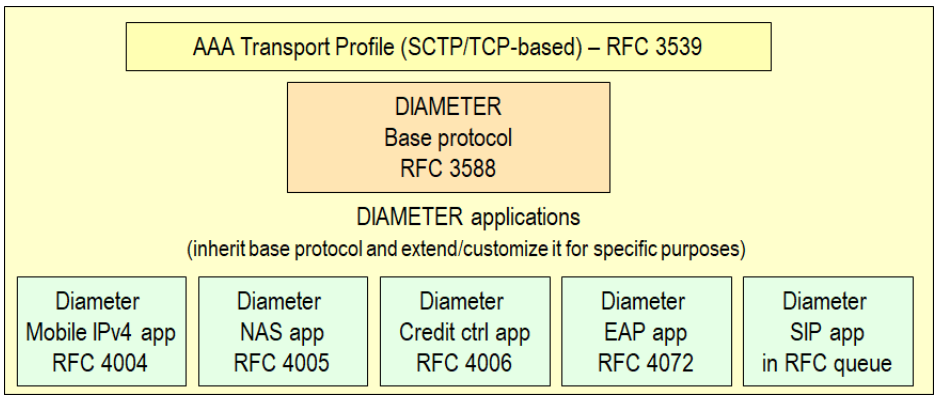
- Se un hacker può monitorare dei tentativi di accesso, può costruire un database di coppie AUTH Request NONCE-Access Accept Packet (e quindi tentare un replay attack). Sarà a quel punto in grado di entrare nella rete quando il NAS genererà un access request nonce già usato. L'attacker potrà, allora, rispondere al NAS fingendosi server usando la risposta che aveva collezionato nel database. L'attacker si autentica da solo in quando conosce già la risposta che deve dare.
- Un altro modo è costruire una tabella dove ad ogni usare id si associano le informazioni password XOR MD5(secret,NONCE); alla ripetizione di una nonce, si effettua uno XOR per ottenere password1 XOR password2. A partire da queste due password legate da XOR è possibile, tramite un'opportuna analisi risalire alla password.

Un protocollo di livello applicatori non deve includere aspetti di sicurezza. Si può provare a sviluppare un proprio sistema di sicurezza, ma non è consigliato. Considerando questi aspetti, RADIUS presenzt limitazioni funzionali dovuti a:

- **Scalabilità**: RADIUS usava UDP, protocollo non affidabile per migliaia di utenti. Pensando in ottica id un sistema esteso, in caso di crash di un server, a causa di UDP ci sarebbe packet loss su larga scala.**AAA server sono affetti da congestione e quindi perdita di pacchetti**. Utilizzare TCP, molto meglio;
- **Extendibility**:(type|len|value) per il tag non erano sufficienti per l'autenticazione. Questa idea adottata da RADIUS di utilizzare i suddetti byte negli attribuiti non fu abbastanz aper affrontare le nuove tecnologie. Questo portò ad una limitazione degli attributi utilizzabili;
- **Interoperability:** L'utilizzo di tanti server che adottano soluzioni differenti,crea consistenti problemi per l'interoperabilità-

6 DIAMETER

Il protocollo **DIAMETER** è un protocollo che nasce come evoluzione del protocollo RADIUS cercando di risolvere le sue problematiche. Questa tecnologia, insieme al RADext (estensione di RADIUS in assenza di sicurezza), è stata sviluppata da IETF (*Internet Engineerign Task Force*). DIAMETER è tecnologicamente più avanzato ed è stato sviluppato in ottica Object Oriented Programming: il protocollo base è un protocollo di scambio di messaggi che poi si specializza nel settore in cui viene implementato.



6.1 Catatteristihe

Le caratteristiche che lo differenziano dal protocollo RADIUS sono le seguenti:

- DIAMETER non è un AAA protocol come il RADIUS, ma è un messaging protocol. In questo senso è detto protocollo Peer to Peer;
- AAA transport profile comunica come deve funzionare il trasposto dei messaggi che che nel caso di DIAMETER, non potendo utilizzare TCP come protocollo di trasposto (dato che garantisce l'arrivo in ordine dei pacchetti), si utilizza SCTP: protocollo TCP in cui si mantiene una connessione persistente tra clien e server e i pacchetti sono in pipeline su una connessione;
- Data la modularità di DIAMETER, dal protocollo base si sviluppano le varie derivazioni in base alla necessità;

- Al livello applicativo si hanno funzionalità di controllo degli errori e di ritrassmisione;
- Funzionalità di rilevamento di duplicati. (*Fondamentale nelle funzioni di Accounting*);
- Presenza dei cosiddetti *watchdog*: pacchetti periodici che consentono di capire quando DIAMETER sta fallendo.

6.2 Problemi TCP

I problemi di TCP in questa specifica applicazione sono che:

- Strict Order Delivery e HOL;
- Mancanza di Multi-Homing;

Quando richiesta, il NAS deve creare una connessione affidabile con l'AAA Server. Per garantirla, basta usare una connessione di tipo TCP. Una volta instaurata la connessione (Handshaking TCP), viene utilizzata per tutti i pacchetti successivi al primo. Ora, l'AAA server può usare più threads per gestire le richieste così che possano essere presi in carico diversi messaggi alla volta. Se un thread si ferma per un problema, TCP diventa svantaggioso in quanto grantisce l'ordine dei pacchetti: fa in modo che non si possa lavorare sui pacchetti successivi se non sono stati serviti i precedenti. Sarà quindi necessaria una nuova ricezione a partire dal messaggio perso.
Questa problematica viene detta **HOL(Head of the line blocking effect)**,
Per quanto riguarda il secondo problema: preso un NAS, se la rete fallisce si può garantire un servizio di continuità grazie a delle connessioni di ricambio ulteriori. Il problema è che l'IP è differenze, e siccome TCP si basa sulle socket che richiedono di pre-specificare l'IP, bisognerebbe creare una nuova socket ogni volta. Quindi, è necessario il **multi-homing** cioè la possibilità di avere più connessioni insieme in modo tale che se una fallisce se ne utilizzi un'altra.

6.3 SCTP Stream Control Transport Protocol

Si tratta di un protocollo che permetterebbe di avere multihominh, ma on viene utilizzato per il cosiddetto *Chicken and egg problem e Internet Ossification.*
Non molto tempo fa si utilizzava memorizzare i dati degli utenti su dispositivi, l'edge Network era quindi povero di dati. L'idea fu quella di portare i dati ai bordi della rete. Si utilizzarono i NAT per risolvere il problema degli indirizzi IP corti, al fine di connettere più device di quanti si poteva. Riguardo al DIAMETER non è possibile fare lo stesso discorso del NAT poiché, per renderlo reale, dovremmo porre due intermediari nella connessione fra due server. Se si utilizzasse un protocollo sconosciuto intermediario, esse bloccherebbero il traffico. Quindi, affinché sia possibile l'utilizzo di questo protocollo serve che i nodi di rete (intermediari fra due server) supportino SCTP,ma questo non accade.

Remark. Nessuno può usare STCP poiché i nodi di rete lo bloccano e per far si che lo supportino si necessita che il protocollo venga utilizzato (*Chicken and egg problem*). Questo problema ha portato all'*internet ossification* ossia il blocco della crescita di Internet.

6.4 Miglioramenti DIAMETER rispetto a RADIUS

- **trasporto affidabile**: affidabilità essenziali per questioni monetari;
- **Standardized error and fail-over control**: al livello applicativo c'è il controllo degli errori e una serie di funzionalità di ritrasmissione gestita tramite l'invio di pacchetti che controllano se un device ha fallitO;
- **Peer Discovery, configuration,capability detection**;
- **Supporto della comunicazione server client**;
- **Gestione delle entità intermedie**;
- **Estensione di limiti funzionali di RADIUS**:
  - (Code|ID|Len|ReqAuth|Attr.) era la struttura del pacchetto di RADIUS. In DIAMETER venne adottato l'utilizzo di un header in quando non era chiaro l'ID assegnato ai pacchetti per via dei limiti funzionali legati alla tecnologia che era cambiata: adesso è possibile comunicare fra più server con i dovuti accorgimenti. DIAMETER modificò il tag di un header seguito da uno più *AVPs*(attribute value pair).
    - **Header(R|P|E|T)**;
      - R: specifica se si tratta di request o answer;
      - P: indica se è proxabile;

- E:indica la presenza di errore;
- T: potentially retransmitting message, fa capire se il paccheto è uno che sto ritrasmettendo e non è nuovo.

* **AVPs(V|M|P)**;
  - V: indica la presenza o meno dei dati vendor specific;
  - M: Mandatory bit che serve in caso di diverse versioni fra NAS e Server che garantisce che un pacchetto non venga perso. Il server può richiedere di ritrasmettere perché non ha capito, risolvendo **l'interoperabilità**. Se M è impostato ad 1, allora il pacchetto deve essere rifiutato.
  - P: dice se un messaggio è criptato o meno.

Diameter header

Version: 0x01	Message length (3B)
R P E T res-flags	Command-Code (3B)
Application-ID (4B)	
Hop-By-Hop Identifier (4B)	
End-To-End Identifier (4B)	

Flags:
R: 1=request, 0=answer
P: proxiable
E: this is an error message
T: potentially retransmitted message

AVPs

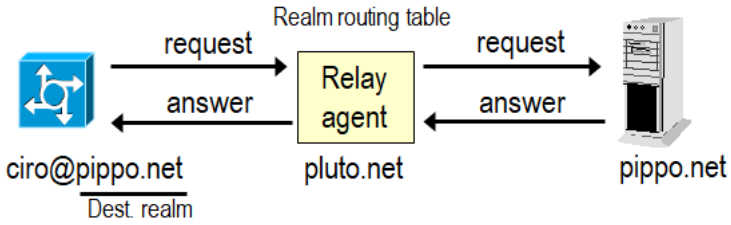
AVP code (4B)	
V M P res-flags	AVP length (3B)
Vendor-ID (optional, 4B)	
..... DATA	

Flags:
V: vendor-specific bit: vendor ID follows, code is from vendor
M: mandatory bit: reject if this AVP unsupported
P: privacy bit: need for e2e encryprion

6.5 Standardizzazioni DIAMETER

- Rely Agent;
- Proxy Agent;
- Redirect Agents;

In DIAMETER sono stati introdotti questi 3 agenti ntermediari che contengono i routing table per connettersi fra diversi server.
Un **relay Agent** accetta le richieste e le indirizza al server corretto basandosi sulle informazioni contenuti nel messaggio; un **proxy agent** si comporta nella stessa maniera, ma può modificare anche i messaggi.



Preso un sistema di server, se uno crolla o cambia qualcosa, comunica il cambiamento a tutti gli altri. Anche quando qualcuno si unisce comunica con tutti. Questo discorso costituisce un incubo per la gestione poiché, se ci sono migliaia di server, è impossibile gestirli senza i *relay e proxy agents*. Se si lasciano a loro le routing table se li gestiscono loro.
Quindi,l'dea è quella di creare un controller centralizzato che non fa niente a parte mantenere le routing table. Ogni volta il serve, che ha il pacchetto da inoltrare, chiede al controller dove si trova il server a cui mandarlo. Così tutti il sistema risulta meno complicato in quanto le routing table sono centralizzate sul controller e tutti i server interrogano quello.

Infine il **redirect agent** non gestisce i dati, ma il controlla. Provvede alla routing decision sulle richieste in arrivo, ma non manda effettivamente la richiesta, ma solo il redirect. Ciò è utile quando le routing decision sono centralizzate. Il relay agent parla con il redirect agent.