# 1 Message Authentication con hash functions

Ho mai detto che una hash function può essere usata per msg auth?
Nuovo problema: come usare hash function per msg auth e ci saranno problemi (perché non è quello il purpose per cui è pensata).
Encrytpion non garantisce integrità a meno che si usi un authenticated enrcyption ⇒ AEAD algorithms, Authenticated Encryption Associeted data ovvero un algortimo che fa sia encryption e authentication.
In TLS 1.3 (la nuova) hanno proibito di usare algoritmi che non abbiamo authentication, quindi sono AEAD.
AES-128 o AES-256 è enrcyption only, AES-GSM o AES-CCM sono auth enrcyption.

## 1.1 Message authentication with simmectric key

Prendi msg m, computa con una chiave k nota ad entrambi gli end(nota ⇒che è pre-shared) C'è una funzione che è usata per generare il tag: riceve una size arbitraria e produce un tag di size fissa e possibilmente piccola (non troppo, per birthday paradox). Trasmetto msg + il tag, message authentication code aggiunge bytes al msg, c'è dell'overhead in quanto non deve aver infromazione(il msg in se è al massimo livello di entropia). Recevier verifica il tag usando la stessa funzione, nota e condivisa, generando il nuovo tag dal msg + chiave k e lo confronta con quello ricevuto.

## 1.2 Definizione di sicurezza per Message Authentication Code

IND-CPA model definiva la confidenzialità, posso trovarne uno analogo per msg auth?
Sicurezza in integrity vuol dire che l'atk non può essere in grado di creare un nuovo msg o poter modificarne uno; anche se il msg modificato perde di senso è considerata una violazione.
Formalmente, faccio un "gioco" contro l'attaccante:

- attacker model può essere Known Message Attack o Chosen Message Attack, ovvero attacker può chiedere qualsiasi coppia (msg,tag) precedente
- può essere adattivo ovvero che il msg è scelto dopo una analisi della situazione.

Ora, se l'attacker scegli uno nuovo messaggio m, diverso da quelli del passato per cui ha i tag, non deve poter forgiare il nuovo tag per il msg m.
Formalmente, la probabilità di forgiare una coppia valida deve essere un $\epsilon$(prob dell'ordine di $2^{-100}$):

- Devo escludere che il tag sia di 1 byte

- Non può tirare a caso il tag del msg con scelta puramente random. Se fosse di 1 byte ⇒ avrei $\frac{1}{256}$, che non va bene, il minimo è almeno 96 bit di tag (meglio 256).

Differenza cruciale nella scurezza: IND-CPA l'attacker poteva scegliere se il msg era A o B e aveva esattamente $\frac{1}{2}$ possibilità.
Message integrity protegge dal man in the middle? Sì, genero il msg m, produco il tag=F(K,m). L'atk intercetta il mesaggio e vuole cambiarlo: se F è cryptographically strong:

- K non può essere computata dal msg e dal tag.
- Non posso cambiare il tag in un nuovo tag senza conoscere k, non posso computare tag*=G(K,m*)
- Non posso cambiare m in m' così che F(K,m)=G(K,m')⇒ anticollision property.

Se lo schema è sicuro, allora potrò sempre intercettare un mitm atk.Mitm ha due aspetti:

- networking class: triviale farlo,ARP poisoning, DNS spoofing.
- Il msg è efficace se posso modificare il msg, non solo cambiare il flow dei msg.

Un buon algoritmo deve anche proteggere dalla creazione di un nuovo msg: message spoofing⇒ creo un nuovo messaggio in cui metto un ip fake facendoti credere che è quello con cui vuoi comunicare.
Posso risolverlo con un auth mechanism:
Se ogni msg è autenticato: DNS è autenticato in plaintext, come faccio a sapere che è proprio, es. Google.com?
Devo aggiungere qualcosa che mi garantisce che sia Google ad esempio con un tag.(la versione di DNSSec dovrebbe proteggere da questo, ma questo aggiunge complessità alla rete quindi si continua ad usare DNS.)Posso spoofare msg, ma devo conoscere il tag⇒se algoritmo è buono probabilità è un $\epsilon$.
Questo schema NON protegge da un reply attack:
voglio mandare due messaggi, es due transazioni. Produco due msg identici, ma la F si applica alla chiave⇒la F deve essere deterministica (va ricomputata all'altro end)e quindi i tag saranno gli stessi,MAC non è abbastanza.Ma se i messaggi hanno un contenuto diverso: timestamp, num. seq etc.. potrei dire che non ci sono problemi.Ma non è così: l'applicazione dovrebbe essere disegnata senza avere in mente problemi di sicurezza.Il protocollo deve essere sicuro, non mi deve importare dell'applicazione.
Prevengo reply atk: uso le nonces, devo garantire a livello di protocollo che tutti i messaggi siano diversi, aggiungo una nonce ai msg. Computo il tag sul msg+nonce, posso mandare la nonce in chiaro.

- Se uso seq num: come gestisco i reboot?Devo prestare attenzione. Parto da 0 e vado avanti, però perdo alcuni n° sequenza, come faccio a dire che i pacchetti ricevuti con alcuni buchi in mezzo sono ok? Devo tenere in mente l'utlimo correttamente ricevuto per discriminare reply atk.

- Random number, se truly random sono meglio.Non ho probelma del reboot, ma come controllo se pkt è nuovo? Ho un certo n° bit,quindi dovrei tenere tutt la lista de msg precedentemente ricevuti, costo di memoria e di computazione per il controllo.

- Timestamp migliore possibile, ma il tempo deve essere garantito o ho problemi.

Settare una nonce sembra facile ma non lo è, la maggior parte dei probelmi implementativi è qui.
1° ingrediente:
Hash functions: molto veloci, sono anticollsione se cryptographic. Buoni prodotti:

- SHA-2 family(SHA256, SHA224,SHA384⇒SHA512 troncato, SHA512). Nel passsato SHA1 e MD5, MD5 la più comune e famosa funzione hash,oggi tutte e due rotte.

- Next: SHA-3 family, sempre gli stessi digest ma con approcci differenti.

es: in TLS e Ipsec, SSH non troppo serio si usa SHA256, sha256sum fa hashing di file su Linux.
2° ingrediente:
Includo il segreto nell'hashing del messaggio messaggio. Facile? Ma dove metto l'auth key nel msg?Lo metto dopo il messaggio: H(M||K), o faccio il contrario? O in altri modi? Ad esempio metterlo sia all'inizio che alla fine etc..Perché me ne preoccupo?
Una funzione hash teorica è una black box, c'è anche definizione per la perfect hash function:
Random Oracle: black box, che preso input X, H(X) = valore truly random, ma che si ripete se X è lo stesso. Ma le due cose non possono coesistere, H(X) deve essere computabile. Nella teoria questo è il modello ideale (come per one time pad) che vorrei avere, ma non posso implementarlo.
Devo vedere nel box:tutte le hash functions (tranne SHA-3, oggi non usate) sono costruite con la costruzione iterative Merkle-Damgard: è difficile trovare f tale che: f(any size)$\longmapsto$ fixed size. Ma è possibile trovare f t.c:
f(fixed size)$\longmapsto$ samller fixed size otuput. Compression function, che possono essere molto buone.
es: sha256
prendo msg di k bit, paddo il messaggio in modo che il risultato(compreso i 64bit di lunghezza del messaggio) sia multiplo di 512 bit: se ad esempio la size del mio file è 1025 bit, metto un bit ad 1 seguito da vari zeri, alla fine del msg mette la size del msg come lunghezza modulo $2^{64}$, sono 64 bit (faccio il modulo nel caso in cui lunghezza sia maggiore di $2^{64}$, così che sia di size fissa). Ora

taglio il msg in chunks di 512 bit: parto con un initialization vector(non crypto) che è noto e fisso, deve poter essere ripetuto. SHA256 prende IV 256bit,diviso in 8 gruppi da 32, è una costante.L'IV fa sì che la funzione di compressione prenda 512(il chunk) + 256(l'IV) = 768 bit di input.Questo perché SHA256 usa aritmetica mod 32 o 64 a seconda dell'architettura.La compression function comprime i 768 bit in 256 bit che è l'hash summary del chunk 1.
Ma ora, se uso questi 256 bit come input per un secondo blocco di compressione, che comprime il chunks 2? SHA256 reitera la stessa funzione di compressione.
La F è il cuore dell'hash function, theorem di Merkle-Damgard dimostra che se F è resistente, ovvero soddisfa le 3 proprietà di una funzione hash⇒l'intera costruzione è sicura.(la F non deve essere lineare)
La chiave è trovare una buona compression function, questa prende un input fisso e ridà un output fisso, a questo punto posso usarla iterativamente; l'ultima iterazione mi darà i 256 bit finali.
In che posizione metto il segreto nell'argomento dell'hash fucntion? Prima del messaggio, o dopo, o in altri modi? La posizione del segreto conta ed è importantissima:
es: msg di 1GB, segreto 128bit, poi ho pad+length.Messaggio è noto, vedo il tag = hash(msg,k), vado da 0 a $2^{128}$ e faccio H(msg,$k_x$)=?tag. Brute force atk devo computare fino al massimo $2^{128}$ hash functions.
SHA256 è white box, so che è costruita iterativamente, il msg è sempre lo stesso: computo i primi blocchi che contengono il messaggio e prenderò l'output precomputato (i 256 bit risultanti), ed ora dovrò computare solo l'ultimo pezzo a partire dal precomputato.Non quindi computare N$\cdot$ $2^{128}$ blocchi, bensì $2^{128}$ ⇒ riduco la complessità d un fattore N.
Se metto il secret all'inizio, posso rompere la forgiability?Posso forgaire un tag valido per un m' scelto da me, partendo da M,tag=H(s,m). Sì è possibile:
triviale forgiare un messaggio autenticato valido m' != m. Estendo msg, che può anche essere insensato, con una parte di plaintext.
Non posso modificare il msg originale ma non è un problema, inoltre lo faccio senza conoscere il segreto: es. aggiungo una transazione alla fine del messaggio.Aggiungo extra chunks, partendo dal MAC code di prima e genero un MAC extended valido.
Questo è un problema ⇒ ho una funzione forte, ma la costruzione rompe tutto (errore tipico della crypto), quindi non si usa mai una funzione non pensata per quel purpose, anche se i purpose sono simili.
La posizione del segreto CONTA TANTISSIMO.
Come fixare il problema:
Hash Based Message Authentication Code (HMAC), che è stata dimostrata essere sicura come l'hash sottostante.
Ho imparato che una secure hash non basta, quindi HMAC aggiunge un modo sicuro di aggiungere segreto nell'hash, non patcho l' hash in se quindi non dipende da come è fatta l'hash.
1996, paper di Bellare, Canetti e Krawczyk con due versioni: crypto e IETR RFC 2104.
Pluggable hash e usando l'HMAC non aumenti il costo computazionale di molto:

$HMAC_k(M)$=H($K^+ \oplus$opad —H($K^+ \oplus$ipad—M)
Il primo pezzo contiene la chiave, i secondo il messaggio. Sembra che sto facendo come prima, ma in realtà sto usando hash del messaggio tra message e secret alla fine. Quindi faccio hash della chiave seguita da hash di message e chiave,come fare 2 volte hash del msg.
Se il segreto K è < della lunghezza di un blocco fai si che sia di pari lunghezza, paddo con zeri, ottengo così $K^+$.Questo è il primo chunk di SHA256.
Per la sicurezza della costruzione, i due segreti che uso negli hash devono essere diversi: miglior costruzione è la nested MAC construction : H($secret_1$ !! H($secret_2$ !! msg)). Ma chiedere di usare due segreti sarebbe stato un disatro, quindi per praticità non era conveniente lasciare all'implementatore la scelta dei due segreti.
Soluzione è che produco due segreti dviersi a partire dallo stesso: in entrambi i due risultati flippo bit diversi riseptto all'originale, sembrano quindi due segreti indipendenti(ma non lo sono) ed hanno una distanza larga in termini di bit.
es : k = 01010101, inner: 01010101$\oplus$01011100=$K_o$, outer: 01010101$\oplus$00110110=$K_i$ (entrambe le sequenze ripetute come serve).
Parto dal msg, aggiungo all'inizio (prefix) $K_i$, runno hash function: parto da IV e lo unisco a $K_i$ ed ottengo un secret IV.Hash sugli altri chunks, ed ottengo l'inner hash: ho un classico MAC secret prefix, devo mettergli una pezza: prendo l'oueter key $K_o$ e faccio hash del singolo blocco (inner hash + pad).
Ottengo quindi HMAC, che è dimostrato essere una costruzione sicura.
storiella: 2005 md5 broken, tutti gli HMAC tags usavano md5. Thm ti dice che la costruzione è sicura quanto l'hash sottostante: se l'hash è unsecure $\Rightarrow$dovrebbe rompersi anche il meccanismo di HMAC. 2006: non era ancora stato trovato un atck pratico ad HMAC di md5.
Assunzioni: modello math dell'hash function:

- pseudorandom output
- anticollision property.

Entrando nei dettagli, Bellare si rende conto che non usa mai la proprietà 2 e capisce che HMAC è più sicuro dell'hash fuction, finché la properietà 1 non è violata.
Paper del 2006 su collision resistance NON necessaria.
Messaggi importanti:

- Confidentiality != integrity
- Message authentication with symmetric key
- Reply atck: MAC non è abbastanza, servono nonces e vanno gestite bene.
- Crypto hash functions
- Come includere key nell'hash fuction? Non è triviale, usa HMAC.

# 2 Gestione dell'accesso remoto: RADIUS

Tool usato nel backend, Remote Authentication Dial In User Service, obsoleto: oggi migliori protocolli(DIAMETER) ma ci sono un sacco di problemi quindi è utile studiarlo.
Posso accedere alla rete usando diverse tecnologie, tutte eterogenee fra loro e largamente distribuite. Gestire la rete con tutte queste tecnologie ed access point: uso server centralizzato, RADIUS server che è incaricato di gestire username e password dell'utente, così da evitare la distribuzione all'interno della rete.
Anche una questione di sicurezza:(di solito in AP: Linux machine con db interni), non lascio username e psw distribuite in giro per la rete.
Devo cambiare l'authentication model: faccio auth con local technology, RADIUS client che counica con l'utene e col server contatta quest'ultimo ed il server ,manda RADIUS response con un si o no a seconda se l'utente può accdere o meno⇒parte più importante.Il client dice quindi all'utente se può entrare o no(l'utente non sa che sta usando RADIUS).

## 2.1 RADIUS: AAA protocol

3 servizi di solito eseguiti insieme:

- Authentication
- Authorization: da non confondere con Authentication, qui voglio sapere che l'utente ha il permesso di usare il servizio (perché ha pagato o per altri motivi). Posso avere
  - authentication senza authorization
  - authorization without authentication ed avrei un servizio privacy preserving.
    Letteratura scientifica è ricca di tecniche per farlo, ma nel mondo pratico non molto usato.
  - l'intersezione fra i due.

  Serve per management
- Accounting: transmitted bytes (quanti GB sto consumando), billing, minuti di telefonate spese etc..
  Segno cosa stai facendo in termini di una risorsa che stai usando.

## 2.2 RADISU è client-server protocol

Richiesta parte dal client, non confondere il RADIUS client con l'end user, ovvero il NAS: ho end user - NAS- RADIUS client- Server.
Basato su UDP/IP porta 1812, client port è ephemeral. Sistema centralizzato,

logicamente centralizzato: in teoria ho un singolo server ma in pratica è ridondato (sennò è single point of failure)
In RADIUS si può usare roaming: se cambio città rispetto a dove sta il server, es. della mia università, dovrei cambiare account, ma quello che accade è che la mia richiesta viene presa dal RADIUS server della città e la inoltra al RADIUS server della mia univeristà, agendo da proxy server.
Architettura complessa, diversi blocchi:

- Server application
- User db: per ogni username ho almeno authentication info, authentication method e authorization attributes
- Client db: clients che possono comunicare col server.
- Accounting db: se RADIUS usato per accounting, deve essere aggiornato in real time, per questo separato dal db utente.Non necessario se si fa solo authentication.

## 2.3 RADIUS security features

Due feature, 1° è Per packet authenticated reply: NAS non ha le mie credenziali, le manda al server, atck interecetta il messaggio e risponde con un "sì", il NAS ora vede che l'utente è autenticato.Non devo pter spoofare il msg⇒deve essere autenticato, ed è quello che è stato fatto: si usa shared secret, approccio CHAP-like,ma:

- solo la reply è autenticata
- l'autenticazione è hash based e non HMAC-based
- funzione hash specifica MD5, quando uso una hash function deve essere future-proof, se metto uno specifico crypto algo in un protocollo è male: qualcuno prima o poi lo romperà. Non è semplice andare poi a modificarlo
- Secret non truly random, ma low-entropy shared key

2° servizio: user password enrcyptata: se uso PAP, ho la psw in chiaro. Standardizzazione di un meccanismo. Problemi:

- Custom mechanism
- Shared secret key usata anche per l'authentication ⇒NUN SE FA, anche se non è exploitabile è errato, perché se rompi la chiave rompi più di un servizio.