1 Overview about TLS and IPsec

TLS ha come progenitore il protocollo SSL da un certo momento in poi cambiò nome a seguito di importanti aggiornamenti al protocollo. Nelle varie versioni di TLS hanno aggiunto notevoli funzionalità aggiuntive tra cui il supporto al protocollo di trasporto dati UDP, l'utilizzo (per poi esser abbandonati in TLV v.1.3) di algoritmi di encrypt come MD5 e SHA-1 ed infine, in TLS v.1.3 (la versione ad oggi in uso) di utilizzare i protcolli AEAD:protocolli che garantiscono confidenzialità e integrità dei dati. Dato che TLS utilizza TCP (o nella versione DTLS UDP), si assegna ad ogni processo una socket per rendere sicuri i dati di livello applicativo è considerato un protocollo che opera tra il livello di trasporto e il livello applicativo. Il suo nome, quindi, è fuorviante *Transport Layer Security*. Come vedremo in seguito questa possibilità di aprire una socket per ogni applicazione risulta problematica. Infine, TLS protegge solamente il payload di TCP e non l'intero pacchetto poiché per come è stato progettato, se così non fosse, non si saprebbe dove e a chi inviare i pacchetti. Da questo aspetto se ne conclude che l'header del pacchetto TCP può essere modificato e quindi si è soggetti ad attacchi di tipo TCP spoofing, MITM, CCA etc...
Un altro protocollo, sviluppato di pari passi a TLS, è IPsec. Esso viene considerato una versione molto più crittograficamente sicura e, data la problematica di TLS nell'utilizzare socket diverse per le varie applicazione, si pone tra il livello di trasporto e di rete. Infatti, "monta"sopra IP e, grazie a questa proprietà, è possibile rendere crittograficamente sicuro l'intero pacchetto TCP/UDP/Altro incapsulando il pacchetto crittografato IP in un altro paccheto così da nascondere l'intero contenuto di quello che si voleva mandare.
Da qui il concetto di **traffic flow confidentiality** che rappresenta un nuovo requisito di sicurezza:la straficazione del pacchetto deve aggiungere sicurezza al protocollo.
Una differenza tra TLS e IPsec è che nella loro progettazione hanno sviluppato il set-up dell'ambiente e il modo in cui si trasferiscono i dati in maniera differente. Questi due aspetti, infatti, rappresenzanto due concetti cardini nello sviluppo di un protocollo di sicurezza.
Quindi, TLS decise di unire il set-up dell'ambiente e il trasferimento di dati delegano il primo aspetto venne realizzato tramite una fase di handshake in cui si negoziano gli algoritmi e, tramite crittografia asimmetrica, si comunicano tali chiavi, necessarie a garantire integrità e confidenzialità, la seconda fase, invece viene detta record phase;mentre in IPsec, si decise di disaccoppiare questi due aspetti: la prima fase di set-up è delegata ad un protocollo automatico detto IKE (Internet Key Exchange);la seconda fase, di trasferimento dati delegata al protocollo utilizzato a supporto di IPsec.

2 TLS Protocol Stack

TLS, nella record phase, utilizza un protocollo detto TLS Protocol Stack che è composto da diverse componenti dette:TLS Record Protocol responsabile del trasferimento dei dati, Handshake Protocol responsabile dello scambio delle informazioni necessarie alla sicurezza della comunicazione, Alert Protocol responsabile della definizione dei messaggi di warning/alert, il Change Cipher Suite responsabile dell'inizializzazione del cipher. Il Protocol Stack si interpone tra HTTP e TCP.

3 TLS Record Protocol

Il TLS Record Protocol è responsabile del trasferimento dei dati dell'utente e, in base agli obiettivi proposti da TLS, a questi dati bisogna aggiungere integrità e confidenzialità. In aggiunte, per questioni tecniche, si decise di aggiungere anche una funzione di compressione dei dati. In totale, quindi vi sono queste funzionalità: compressione, HMAC (per l'integrità) e Encryption. Quello che si potrebbe pensare è che non vi è alcuna differenza tra l'ordine in cui vengono effettuate queste operazioni, tuttavia non è così:ne è un esempio l'attacco CRIME in cui si frutta la compressione e poi l'encryption per decifrare l'intero dato che abbiamo ricevuto. Inoltre, un'altra operazione totalmente scorretta è quella di effettuare prima l'integrità con HMAC e poi l'encryption.Alla fine delle tre fasi (compression,integrity,encryption **IN QUESTO ORDINE**) si aggiunge un record header composto da Content Type,Major Version, Minor Version,Compressed Lenght.Questo header è formato da 5 bytes.

4 Message Authentication Code

Il Message Authentication Code rappresenta la parte del pacchetto TLS che serve a fornire integrità.TLS scelse di adottare HMAC per le sue proprietà di difesa contro attacchi di tipo Message Spoofing e MITM. Tuttavia, non si è protetti contro attacchi di replay attack, cioè attacchi in cui si reinvia un messaggio o una porzione di esso per indurre comportamenti non desiderati. Per proteggersi da questo tipo di attacco bisogna aggiungere al protocollo una informazione che aggiunge freshness alla comunicazione. TLS decise di adottare il NONCE. Il NONCE è una serie di bit casuali che cambiano in ogni comunicazione e può essere inviato in chiaro poiché non costituisce un requisito di sicurezza. In TLS per ottenere un NONCE che cambia in ogni comunicazione, scelsero di utilizzare un numero di sequenza scambiato tra Client e Server. Inoltre, per motivi di ottimizzazione e ricordando che il protocollo TCP utilizza già dei propri numeri di sequenza per fornire un servizio affidabile di trasporto dati, decisero di legare TLS e TCP utilizzando gli stessi numeri di sequenza. Da qui nasce l'esigenza di sviluppare DTLS (Datagram Transport Layer Security) per supportare UDP e la sua natura di trasporto non affidabile di datagrammi.

In conclusione, quindi le operazioni che vengono effettuate per aggiungere integrità al pacchetto è quella di prendere i dati, affiancare un TLS Header e prima di esso il sequence number che, visto che viene trasmesso direttamente in TCP, non viene trasmesso.

5 Encyption and Authentication

L'utilizzo dell'encryption permette di assicurare la sicurezza semantica, mentre l'authentication permette di avere integrità ed,in particolare, si ottiene la cosiddetta proprietà di unforgeability che consente di rendere inforgiabili i messaggi (protezione da message tampering e message spoofing) se non si possiede la chiave. L'unica encryption che consente di ottenere sia sicurezza semantica che integrità sono gli algoritmi AEAD (i.e.Ephimeral Diffie-Hellman) . Per garantire questi aspetti in un protocollo di sicurezza bisogna scegliere fa cifrare e poi autenticare o viceversa. In particolare, si ha che in SSL (la versione peggiore) si è scelto di cifrare il dato orginario e poi effettuare l'autenticazione proprio sul dato originario. La soluzione che garantisce un livello di sicurezza maggiore, ma non comunque corretta, è quella proposta in TLS in cui si effettua l'integrità sul dato cifrato. Infine, la soluzione che fino ad ora è risultata la migliore è quella proposta da IPsec in cui prima si cifra il dato da inviare per poi aggiungere l'integrità sul ciphertext.

6 Block Ciphers:Introduzione

I Block cipher rappresentano un'alternativa ai stream cipher. In particolare, come dice il nome, lavorano in blocchi: preso un plaintext lo si divide in blocchi a cui verrà applicata una certa funzione PRF (Pseudo Random Function) per ottenere un ciphertext; in seguito, per decifrare il ciphertext, si effettua l'operazione inversa applicando PRF^{-1} per ottenere tutti i blocchi di plaintext che, ricongiunti, formeranno il messaggio inizialmente inviato. Un PRF è una pseudo random function che consiste nel considerare tutte le permutazioni di un insieme e si sceglie una permutazione di essa in maniera uniforme. Il vincolo di queste PRF è che la funzione deve essere biettiva e quindi due elementi del dominio devono avere due elementi diversi nel codominio. Nel mondo reale, questa probabilità deve essere negligible. Inoltre, nel caso in cui il messaggio è superiore di un blocco, si divide il messaggio e si cifrano indipendentemente i singoli blocchi. Questo tipo di algoritmo viene detto ECB (Electronic Code Book) e non deve mai essere utilizzato per le seguenti ragioni:preso un plaintext, porzioni di messaggio, sono codificate nello stesso ciphertext; preso un plaintext, se esso viene cifrato due volte si produrrà lo stesso plaintext. Per risolvere queste problematiche si deve quindi introdurre una freshness data da IV (bit casuali lunghi quanto il blocco) che, nel momento dell'encryption viene XORato con il plaintext, e poi introdotto all'interno del PRF;nel momento della decyption, il ciphertext viene inserito all'interno del PRF^{-1} per poi essere XORato con l'IV.

Condizione necessaria affinché è possibile decriptare è che si invii l'IV insieme al ciphertext.
La maniera più corretta di usare un block cipher è quella di prendere il messaggio, dividerlo in blocchi e a ciascun blocco inserire un IV che viene XORato con il plaintext, e poi introdotto nel PRF per generare il ciphertext. Ad ogni blocco del ciphertext si affianca il relativo IV, viene compattato ed inviato a destinazione. Nella decryption si effettua il procedimento inverso.
In conclusione, se dobbiamo necessariamente utilizzare ECB dobbiamo usarlo per messaggi molto piccoli e che non si ripetano. In tutti gli altri casi, si utilizzano i modes of operation: sono costruzioni particolari necessarie per utilizzare i block cipher che trasformano un block cipher in uno stream cipher.

7 Modes Of Operation

I modes of operation sono tecniche utilizzate per combinare i block cipher ai messaggi con dimensione maggiori di un blocco.
I più utilizzati sono CBC (Counter Block Chaining), CTR (Counter Mode), CGB (Cipher Feedback Mode), OFB (Output Feedback Mode). I più avanzati sono GCM (Galois Counter Mode), OCB (Offset Codebook Mode).

8 CBC- Cipher Block Chaining

Il CBC (Cipher Block Cipher) è una tecnica di ENC e DEC per plaintext di dimensione maggiore di quella di un blocco. In particolare, vengono risulte le principali problmeatiche di ECB.Infatti, nella fase di encryption si prende un IV generato da un TRNG viene XORato con il plaintext, si applica il PRF e il suo output sarà il ciphertext che fungerà da IV per il blocco successivo. Si invia come ciphertext l'IV affiancato da tutti i blocchi cifrati. Nella fase di decryption si prende l'intero ciphertext, lo si suddivide in blocchi. Ora, partendo dal primo blocco si applica la funzione inversa di PRF che verrà XORato con l'IV per produrre il plaintext;il cipher del blocco precedente verrà utizzato come IV del blocco successivo per produrre il plaintext. Le problematiche di questo protocollo è che necessita di due circuiti per essere attuato (uno per ENC e uno per DEC), non è totalmente paralizzabile ma lo è solamente la fase di DEC. Quindi, la decript è molto veloce (a discapito dell'ENC).
Questa tecnica viene detta chaining poiché si lega il risultato precedente a quello successivo.
Questa caratteristica, tuttavia, presenta delle problematiche dato che il PRF soffre del cosidetto problema degli shor cycle:presa una permutazione dei blocchi, in base al punto di partenza con cui si applica la catena di PRF, può presentare dei cicli che rappresentano un problema di sicurezza dato che rivelano delle informazioni sul messaggio cifrato.
Di questo problema ne soffrono anche delle versioni parallelizzate e/o simili a CBC dette OFB (Output Feedback Mode) e CFB (Cipher Feedback Mode).

In CFB, la fase di encryption è così composta:si prende l'IV, si applica la funzione PRF e a questo punto il suo risultato viene XORato con il plaintext, il risultato dello XOR rappresenta l'IV per il blocco successivo e così via;per la fase di decryption, si prende l'IV, si applica la funzione PRF e se ne effettua lo XOR con il primo blocco del ciphertext, che in seguito rappresenterà l'IV per il blocco successivo, per ottenere il plaintext. Questa costruzione, come quella di OFB, ha la caratteristica di usare la funzione PRF sempre in avanti sia in ENC che DEC riducendo così i circuiti che vengono utilizzati per applicare questo algoritmo. Inoltre, questa costruzione non risulta parallelizzabile, ma ha una fase di ENC lenta mentre la DEC veloce.
La seconda alternativa è quella parallelizzabile sia in ENC che DEC detta OFB (Output Feedback Mode). Nell'encryption si prende l'IV si applica PRF, che fungerà da IV per il blocco successivo, si applica lo XOR con il plaintext per ottenere il ciphertext. Nella fase di decryption si prende l'IV si applica PRF. Il suo risultato sarà l'IV del blocco successivo, ma prima verrà XORato con il ciphertxt per produrre il plaintext.

9 Counter Mode-CTR

Il CTR è un modes of operation che prevede l'utilizzo di un contatore incrementale che viene utilizzato come IV. In particolare si iniziliazza il contatore, entra nel blocco PRP e il suo output viene XORato con il plaintext. A differenza da CBC e derivati, qui ogni blocco viene cifrato e decifrato in maniera singola rendendo più efficiente il sistema dato che non si necessita più di decifrare l'intero pacchetto per verificare l'autenticità del pacchetto in mio possesso.
In un'applicazione reale, si aggiunge sempre una parte di randomicità affiancando a questo contatore 96 su 128 bit una sequanza di bit casuale. Inoltre, questo procedimento equivale all'effettiva trasformazione dei block cipher in stream cipher.

10 BEAST Attack e Chosen Bondary Attack

Il BEAST attack è un attacco ai block cipher ed in particolare al CBC (Cipher Block Chaining) che sfrutta la problematica di short cycle e quindi della predicibilità dell'IV. In particolare, supponiamo che l'attacker possa predirre l'IV del blocco successivo, che abbiamo visto il ciphertext del blocco precedente e che possa far cifrare un messaggio a sua scelta.
Se CBC fosse CPA, allora la predicibilità dell'IV non dovrebbe nuocere alla sicurezza e soprattutto condizione necessaria e sufficiente per essere CPA è che l'IV non si ripeta mai e che non deve essere predicibile.
L'attacco procede come segue, l'attacker conosce il ciphertext del blocco precedente e ipotizza che nel campo password del ciphertext vi sia una password fra due scelte. A questo punto, l'attacker prende il ciphertext del blocco precedente e lo mette in XOR con l'IV e con la password che ipotizza esserci e ci applica

PRP. Questa azione, nel momento dell'encrypt fa cancellare l'IV predetto e se nel ciphertext finale vi è il ciphertext del blocco che voglio indovinare allora conosco la password del plaintext.
Nonostante questo attacco sia potente poiché mi permette di conoscere l'intero blocco che vogliamo attaccare non è molto pratico dato che deve essere flessibile alle tecnologie utilizzate per confincere l'implementazione a cifrare un messaggio a nostra scelta, bisogna avere un agente in esecuzione nel browser e si necessita la possibilità di effettuare injection di plaintext all'interno del messaggio. Anche se un attack riuscisse ad avere tutte questi poteri dovrebbe essere in grado di effettuare un brute force attack per conoscere il blocco da decifrare.
Una sua altertaniva, prende il nome di chosen boundary attack. Questi tipi di attacchi sono lineari nella dimensione del messaggio e permettono di scoprire un byte/bit alla volta del plaintex. Infatit, si fa corrispondere l'ultimo bit che voglio decifrare con la fine del blocco e facendo cifrare un messaggio a mia scelta si effettua un brute force attack per capire un byte alla volta partendo dall'ultimo blocco.

11 MAC-Then-Encrypt, Padding Oracle Attack

I Padding Oracle Attack sono un gruppo di euristiche di attacchi che si possono effettuare ogni qual volta che si utilizza il padding in un protocollo di sicurezza e si basano sulla possibilità di indovinare la lunghezza del padding.
In TLS, il padding del messaggio viene effettuato tramite la specifica PKCS 7 che consiste di inserire alla fine dei dati da inviare un byte contenente la lunghezza del padding del messaggio e i byte precedenti, fino al raggiungimento della lunghezza di padding, vengo riempiti con il lavore della lunghezza di pagging. Inoltre, vi è la possibilità di aggiungere blocchi interi di padding.
La scelta di utilizzare questo tipo di padding può rivelarsi utile dato che tramite esso si possono rilevare e scartare pacchetti malformati sia causati da errori di connessione e sia causati da attacchi alla comunicazione.
In TLS, si scelse di utilizzare CBC (Cipher Block Chaining) come algoritmo di cifratura e le operazioni che vengono effettuate sono MAC-then-Encrypt nel seguente ordine:prima si inizia a decifrare il messaggio se la lunghezza del messaggio non è un multiplo della dimansione del blocco o se non ha un padding corretto, allora si invia un alert decryption failed;inoltre, se queste due fasi sono andate a buon fine si fa il check di integrità sul messaggio decifrato e si ritorna un messaggio BAD_MAC,
Questo sistema di notifica potrebbe sembrare utile, ma a livello crittografico no poiché si danno informazioni all'attaccante e bisogna stare anche attenti all'implementazione di questi algoritmi poiché vi si potrebbe lasciare dei side channel aperti che consentirebbero di rompere le patch a implementazioni non corrette. Il Padding Oracle Attack si basa su questo.
Ipotizziamo che un attaccante vuole conoscere l'ultimo byte del messaggio e la sua abilità è quella di poter modificare il ciphertext e di sottoporre il nuovo ciphertext ad un oracolo che mi dice se il padding è ben formato oppure se ho

un BAD_MAC. Nel caso in cui si utilizzi CBC, devo modificare l'ultimo byte del blocco del ciphertext precedente a quello che voglio indovinare, scartando tutti blocchi successivi a quello che voglio indovinare, e per fare ciò devo modificarlo effettuando lo XOR tra il ciphertext che conosco, il byte che penso sia il pt e un padding valido secondo lo standard di TLS. A questo punto, invio ad un oracolo il mio nuovo ciphertext modificato:se mi risponde con un alert decryption error allora il padding non è stato ben formato e non ho invodinato il byte;altrimenti riceverò un BAD_MAC alert che mi indica che il padding è valido (sono riuscito a indovinare l'ultimo byte e il ciphertext presenta il padding da me voluto). Iterando questo procedimento con i restanti byte e mantenendo le modifiche dei guess precedenti, posso risalire a tutto il blocco.
Vi è anche la possibilità di indovinare un intero blocco in una passata prestando attenzione a effettuare lo XOR dell'intero blocco con un padding valido e un guess valido.
Questa tipologia di attacco rientra nelle cosiddette chosen ciphertext attack CCA:schemi di attacco in cui all'attacker viene data la facoltà di scegliere e cifrare un ciphertext di sua scelta.

12 CRIME Attack

L'attacco CRIME è un attacco utile a dimostrare come la compressione seguita dall'encryption è sbagliata. Così facendo, si va a modificare la natura del ciphertext portanto a rivelare il plaintext del messaggio.
Questo attacco rientra nelle euristiche dei cosiddetti Plaintext Injection Attack ed è stato svolto in TLS. In particolare, TLS utilizza come metodo di compressione il metodo DEFLATE che comprende due algoritmi Lempel Coding Z77 e Huffman Coding:il primo serve a comprimere le sequenza ripetute del messaggio mentre il secondo serve a comprimere i simboli ripetuti. Come risultato si ottiene che, per ogni sequenza di simboli/sequenze ripetute, si assegna un offeset e lunghezza alla prima occorrenza di quella determinata sequenza/simbolo. Vi è la possibilità di impostare una lunghezza massima di offset per ogni ripetizione. Ritornando all'attacco CRIME, un attacker deve avere la facoltà di inettare del plaintext all'interno del messaggio e supponiamo che sia noto l'header e non noto il contenuto del messaggio. Per capire la prima lettera del messaggio, si inietta del plaintext e si fa comprimere il nuovo messaggio. Se la dimensione del messaggio diminuisce abbiamo indovinato il primo carattere del messaggio, altrimenti occorre ciclare i caratteri del messaggio.

13 Authentication Encryption

L'Authentication Encryption nasce dall'eseigneza di combinare confidenialità e integrità dato che l'encryption da sola non permette di ottenere questo risultat. Per dimostrare che la costruzione di sola confidenzialità non ci assicure l'integrità e confidenzialità possiamo supporre questi due schemi di possibili attacchi. Per

quanto riguarda il primo attacco, supponiamo che un utente stia comunicando con un server su una specifica porta a che quest'utlitmo prenda il pacchetto TCP in arrivo, lo decifri e lo invii alla socket corrispondente all'applicazione corrispondente al destinatario del pacchetto. All'attacker diamo la facoltà di poter modificare il pacchetto, ma non il ciphertext. Quindi, schematicamente, l'attacker si pone dalla parte del server su una porta e quinid una socket differente;quello che può fare ora è modificare il campo socket del pacchetto TCP per farsi reindirizzare tutto il traffico in entrata in quella determinata socket. Non è detto riesca a decifrare i messaggi in arrivo, ma nel frattempo ha disabilitato la connessione.
Un secondo tipo di attacco viene schematizzato in questa maniera: supponiamo che un utente stia inviando pacchetti ad un server in AES-CRT in un pacchetto TPC, ma il payload del pacchetto contiene un singolo keystroke. Adesso, supponiamo che un attacker possa effettuare injection nel ciphertext e che possa modificare il campo chksum di TCP. Questo campo effettua un controllo di integrità sul messaggio in arrivo verificando che il numero di 1 all'interno del messaggio: se questo numero è corretto, allora il pacchetto è stato trasmesso correttamente;altrminenti si ritorna un REJECT stando ad indicare problemi sul pacchetto. Date queste considerazioni, possiamo effettuare un attacci basato su oracolo (l'oracolo è rappresentato dal protocollo TCP che mi dice se il pacchetto è ben formato o meno).
Quindi, l'attacker può inviare al server un nuovo pacchetto in cui si cicla il campo chksum e il singolo keystroke. A questo punto si attende la risposta del server per verificare se ho trovato o meno il keystroke inviato verificando sia il campo chksum e il ciphertext.
Dati questi due tipi di attacchi si può concludere che se in un protocollo di sicurezza se vogliamo ottenere solamente integrità, allora dobbiamo utilizzare HMAC se invece vogliamo includere sia integrità che confidenzialità dobbiamo utilizzare i protocolli AEAD (Authenticated Encryption With Associated Data): i pacchetti così formati saranno cifrati in un ciphertext da un cipher (dipendente dalla chiave k) e all'interno di esso vi sarà un tag permette di avere un controllo di integrità. L'applicazione di questi protocolli può ritornare un output REJECTED nel caso in cui il tag non è valido. Infine, AEAD è sicuro se è semanticamente sicuro sotto un CPA e garantisce integrità.

14 Ciphertext Integrity

Per Ciphertext integrity intendiamo il seguente schema: supponiamo che un challenger possegga una chiave segreta e un avversario. L'avversa può scegliere un messaggio a sua scelta, cifrarlo e mandarlo al challenger n volte. Quest'ultimo lo cifra n voler tramite la chiave, produce il CT ed lo invia all'avversario. Ora l'avversario invia un nuovo ciphertext al challenger (diverso da tutti quelli precedentemente inviati) e se non riceve il messaggio REJECT, allora il ciphertext non è integro e il gioco è perso;altrimenti il gioco è vinto.
In altre parole, la probabilità di non ricevere REJECT è trascurabile.

15 AES-GCM

AES-GCM è un algoritmo di cifratura che appartiene alla famiglia delle AEAD (Authenticated Encryption With Associated Data) e si compone di due fasi (a differenza di OCB): la prima fase è quella di encryption in cui si utilizza AES-CTR e la seconda è quella di integrità in cui si utilizza una funzione di hashin non crittografica (GHASH).
In particolare, si prende un IV affiancato ad un contatore, si applica AES e il suo prodotto viene XORato con il messaggio così da produrre il ciphertext. A questo punto, si deve aggiungere la parte di integrità tramite la funzione di hash non crittografica che opera nel campo dei prodotti tra polinomi di Galois. Questa costruzione segue quella di Wegman-Carter. Il CT appena prodotto viene inserito in un blocco GM che dipende da una diversa chiave rispetto all'encryption ottenuta tramite l'applicazione di AES con input una stringa di bit e la chiave di partenze (questo procedimento viene detto Key Derivation). Quindi, con questa nuova chiave e il CT (che vengono insertiti nel blocco GM), il suo prodotto viene XORato con il ciphertext successivo e così via.
Arrivati a questo punto, siamo ancora soggetti a expansion attack e per risolvere questo problema autentichiamo anche la lunghezza del messaggio (e quindi entrerà nel blocco GM con chiave ottenuta dall'IV e contatore zero). Così facendo abbiamo prodotto un authentication tag,
Come già detto, la procedura di autenticazione segue la costruzione di Wegman-Carter che specifica la procedura da utilizzare per aggiungere la parte di autenticazione da una funzione hash non crittografica e una funzione PRF (Pseudo Random Function). Tuttavia, questa funzione di hash non crittografica deve appartere alla famiglia delle Universal Hash Function che hanno le seguenti proprietà: la probabilità che un attacker, cambiando chiave, trovi una collisione è trascurabile e che si utilizzi un PRF. L'unione di una UHK (Universal Hash Function) e un PRF (Pseudo Random Function) fornire un Authentication tag: UHK prende in ingresso una chiave K1 e il messaggio mentre il PRF prende in ingresso la chiave K2 e l'IV.
Infatti, in AES-GMC si effettua lo XOR tra GM e AES per produrre il TAG.
Fino a questo punto abbiamo ottenuto la parte di authenticated encrypted.
Quello che ci manca ora è aggiungere la parte di Associated Data ottenuta antemponendo al primo ciphertext la parte di dati associati per cui il dato precedente entra nel blocco GM che viene XORato con il blocco dati successivi per poi entrare in XOR con il primo ciphertext e così via.
I difetti di questo algoritmo crittografico è che la funzione GHASH è linerare rispetto all'operazione di XOR e quindi un attacker potrebbe risalire allora XOR fra i GHASH e forgiare messaggi anche se K rimane ancora segreta.

16 Crittografia Asimmetrica

La crittogrrafia asimmetrica è una forma di crittografia per garantire confidenzialità che si basa, a differenza dalla crittografica simmetrica, su due chiavi differenti per la cifratura e la decifratura. In particolare, si ha che nel momento della cifratura si utilizza la chiave pubblica (nota a tutti) e per decifrare si utilizza la chiave segreta.
Tutti possono cifrare, ma solo una entità può decifrare. Ipotizziamo che un client deve inviare dei dati ad un server in maniera sicura ed utilizzando la crittografia asimmetrica. Il client comunica di volte inviare dei dati e il server risponde con la sua chiave pubblica, a questo punto il client prende la chiave, cifra i dati e li spedisce al destinatario. Un attacker anche se prendesse i dati non li potrebbe cifrare e quindi risalire al plaintext.
Nella realtà si utilizza la cifrattografia asimmetrica solo per scambiare il segreto condiviso per poi utilizzarlo per effettuare il trasferimento tramite chiave simmetrica. Un approccio che realizza questo schema è detto Hybrid Encryption e si basa su due fasi:il trasporto chiave e l'accordo della chiave. Nella prima fase, il server manda al client la sua chiave pubblica, il client genera un suo random secret e lo rispedisce al server cifrandolo con la chiave pubblica del server. Ora, una volta trasportata la chiave, vi è la fase di accordo sulla chiave in cui il client manda il suo valore pubblico che, una volta ricevuto dal server, effettua una determinata operazione con il suo proprio valore privato e viceversa così da aver trasmesso e condiviso la stessa chiave sia dalla parte del server e del client. Più nel dettaglio abbiamo che il dato viene cifrato con al chiave simmetrica, e si invia al server il dato cifrato più la chiave simmetrica che è stata cifrata secondo la chiave accordata dalla crittografia asimmetrica.

17 Pubkey Cryptografy

La cifratura a chiave pubblica può essere vista in due modi ma concettualmente si applica sempre DEC (ENC (M)). La cifratura a chiave pubblica si dirama in due:cifratura a chiave pubblica e firma digitale.
Nella firma digitale solamente l'utente possiede la chiave segreta per decifrare il messaggio e tutti gli altri possono cifrare;nella cifratura a chiave pubblica solo l'utente ha la chiave per decifrare mentre tutti gli altri possono cifrate, attraverso la chiave pubblica. In particolare per la firma digitale, supponiamo che un utente voglia inviare una firma e usare la crittografia asimmetrica. Risulta lampante che il messaggio debba essere ridotto e quindi si utilizza una funzione di hash con la chiave segreta, nota solo al proprietario della firma, per formare un digest di piccole dimensioni. Ora un qualsiasi utente che voglia verificare la firma deve applicare la chiave pubblice dell'utente al messaggio e verificare che il digest del messaggio sia uguale a quello ricevuto dall'utente. Se questo processo va a buon fine allora la firma sul documento è originale.
Inoltre, questo procedimento è differente dal MAC e dall'autenticazione dato che nei secondi si utilizza la crittografia a chiave simmetrica (quindi una unica chi-

ave) mentre nei primi si utilizzano due chiavi (una segreta per generare la firma e una pubblica che le altre entità devono applicare per verificare il messaggio).

18 Algoritmi Asymmetric Criptography

L'algoritmi di crittografia asimmetrica si basano sul render difficile una operazione all'indietro, ma facile un'operazione in aventi. Ne è un esempio il protocollo di Diffie-Hellman che sfrutta la problematica del discrete logarith problem: Dato un numero primo x è facile calcolare $y = g^x \mod p$, ma è difficile calcolare, dato $y = g^x \mod p$, x numero primo. Il primo aspetto è facile poiché il logaritmo ha un andamento monotono, mentre il secondo no;quindi si risolve utilizzando il metodo DLOG che rappresenta un metodo ottimizzato per calcolare x. (Algoritmo Square Multiply)
Un altro esempio, viene dato da RSA che basa la sua costruzione sulla difficoltà di trovare due grandi numeri primi in grado di fattorizzare il numero n: dato che non vi è una regola precisa di fattorizzazione di numeri molto grandi a meno che non si proceda per tentativi.
Da Diffie-Hellman viene il cosiddetto accordo delle chiavi di Diffie-Hellman e funziona come segue: l'utente e il client generando un valore pubblico che rispettivamente valgono $g^x \mod p$ e $g^y \mod p$ che vengono scambiati tra di loro. A questo punto, si calcolano con i loro valori segreti x e y $g^{xy} \mod p$. Così facendo, sono riusciti ad avere la stessa chiave scambiandosi un valore pubblico. Questa soluzione, tuttavia, soffre di MITM poiché un attacker si potrebbe porre tra l'utente e il server, generare un nuovo valore z con il suo corrispondente valore in modulo così da gestire la connessione tra le due entità e possedere entrambe le chiavi e quindi rompere il sistema. Una versione migliorata è rappresentata dal cosiddetto Fixed-Diffie-Hellman che consiste nell'aggiungere all'accordo i certificiati che permettono di firmare i valori pubblici delle entità e quindi evitare un MITM, tuttavia siamo soggetti a brute force attack poiché le chiavi, ora, sono statiche dato che per cambiare sessione dovremmo sempre interpellare le CA.
Infine, per ammettere anche la possibilità di avere chiavi diverse in ogni sessione senza interpellare i CA occorre procedere nella sequente maniera detta Ephemeral Diffie-Hellman: il client genera il suo valore pubblico e lo firma usando la sua chiave privata con il suo nome. A questo punto, invia al server sia la firma sia un certificato contentente la sua chiave pubblica per decifrare. Così facendo, il server può verificare l'autenticità del client tramite il certificato ed ottenere il valore pubblico.
Abbiamo evitato sia MITM che garantito l'identità degli end-points e possiamo cambiare in ogni sessione la chiave senza interpellare il CA dato che ci firmiramo da soli e la certification authority garantisce che la mia firma sia valida. Anche per questo motivo viene detto Ephimeral.

19 RSA:Key Transport

Sappiamo che il protocollo TLS utilizza RSA per garantire integrità ai messaggi. Inoltre, nella fase di handshake di questo protocollo si ha una fase di scambio chiavi e RSA key Transport è una alternativa a Diffie-Hellman.
Supponiamo che due entità devono comunicare. Il client richiede al server la chiave pubblica per cifrare e, una volta ricevuta, calcola la chiave, la cifra tramite la chiave pubblica del server e la invia. Questo protocollo potrebbe risultare a priva vista corretto, tuttavia, come in Diffie-Hellman sia soggetti ad attacchi di tipo MITM:un attacker si può interporre tra client e server e intercettare la chiave pubblica del server, modificarla e gestire così la comunicazione tra server e client eseguendo una funzione simile al proxy. L'attacker diventa un intermmediario nella comunicazione in grado di conoscere tutti i contentui dei messaggi.
Cosa si può fare per risolvere questa problematica?Occorre legare la chiave pubblica ad una identità tramite i CA:certificate authority.
Più nel dettaglio, RSA non solo è soggetto a MITM, ma anche ad attacchi di tipo CCA (Chosen Ciphertext Attack):supponiamo che un attacker sia in grado di vedere il cipher text di RSA e che possa fare decifrare un ciphertext a sua scelta. L'attacker, cifrando un plaintext a sua scelta con la chiave pubblica del server così da formare un nuovo messaggio C', lo facciamo decifrare e moltiplichiamo per l'inversa del plaintext che abbiamo inserito nel ciphertext così da ritrovare il messaggio in P. In RSA questa problematica viene detta MALLEABLE:noto un ciphertext non dovremmo risalire ad m tramite un messaggio C' che è in qualche modo collegato a m.
Per risolvere queste problematiche si introduce il padding standardizzato PKCS. Questo standard risolvere le problematiche di MITM, CCA e criptazione di messaggi di tipo 00 e 01 etc... introducendo un preambolo di 2 bytes composti da 00—02, 8 byte di casuali che non devono essere uguali a 00, il resto sono dati.
Il nuovo schema di RSA Key Padding quindi comporta che il server invia la sua chiave pubblica al client che manderà la sua chiave criptata (Client Key Exchange Message) con RSA e paddata opportunatamente secondo lo standard PKCS generando così la pre-master secret. A questo punto, quando il server riceverà questo client key exchange message, lo decifra e in caso di padding non corretto secondo la specifica ritorna un abort message al client;in caso contrario si continua con il setup della sessione.
Il ritornare questo tipo di informazione, rilascia un bit informativo che permette di avviare un attacco di tipo oracolo detto bleichenbacher's Oracle Attack.

20 Bleichenbacher's Oracle Attack

Il Bleichenbacher's Oracle Attack è un attacco che sfrutta i messaggi di abort di RSA nel momento della decripatazione. In particolare dato un CP, si effettua un CCA modificando il CT dell'utente con un valore r cifrato e quindi moltiplciato con il CT tramite la chiave pubblica. A questo punto, si invia questo messaggio

all'oracolo che mi può rispondere con abort message o meno. Nel caso in cui si riceve un messaggio di abort, allora in base alla r inviziale cambio il suo valore ascoltando conseguenza la risposta in base al preambolo del pagdding.
Con un numero sufficiente di tentativi si può risalire al plaintext.
Due attacchi pratici che fanno leva su questo oracolo son il DROWN Attack e il ROBOT Attack. Nel primo attacco si effettua un downgrade ai server di SSL per rompere il protocollo nel server più debole per aggirare ed atrtaccare il server principale. Questo tipo di attacco è un attacco immideat dato che una volta effettuato non ci si può difendere ed è stato così pericoloso poiché i due terzi del web nel fare l'upgrade del protocollo utilizzavano la stessa chiave sia per il nuovo e vecchio server.
Per quanto riguarda il secondo attacco, si effettua il cosiddetto Fuzzing Protocol:si inviano volontariamente messaggi mal formato per innescare delle eccezioni nell'implementazionie del server per esporre le vulnerabilità dei sistemi.
Per risolvere queste problematiche occorre o eliinare il padding, eliminare le notifiche di decifrazione sbagliata oppure sbarazzarsi completamente di RSAKey Transport come è stato fatto in TLS v.1.3.

21 Certificate Authority

Come abbianmo detto in RSA dobbiamo trovare un modo per legare una entità digitale ad una chiave pubblica e per fare questo si utilizzano i certificate authority CA. Questi sono degli organi che rilasciano certificati e che attestano che, collegata a quella chiave vi è chi si dice di essere.
Infatti per richiedere un certificato l'ente deve recarti in una CA, presentarsi ad una CA con la sua chiave pubblica e farsi rilasciare il certificato:non è necessario presentarsi alla CA, ma basta anche utilizzare un canale differente da quello utilizzato per comunicare. Gli unici che non devono percorrere questo procedimento sono le CA che dovrebbe essere dei trust anchor cioè ci si fida che l'identità della CA sia valida.
Supponiamo che un utente voglia comunicare con una banca, quindi gli manda la richiesta e risponde con un messaggio del tipo "Ciao sono la banca e questo è il mio nome e questa è la mia chiave pubblica". La banca invia questo messaggio fimato dalla CA ed ora quello che deve fare l'utente è controllare localmente la lista di tutti i certificati preinstallati nella macchina e verificare che la chiave sia valida.
Se ci fermassimo a questo punto abbiamo solamente verificato che la chiave è valida e non appartiene ad un attacker, cosa possiamo fare per accertarci che il nostro interlocutore possegga oltre alla chiave pubblica anche la chiave privata? Per fare ciò, occorre far decifrare o cifrare un qualcosa di nuovo e sempre diverso al server. Entrambe le soluzioni sono valide, ma la differenza è che al momento di ricevere la chiavem (nel primo caso), viene utilizzare per cifrare un NONCE, viene poi inviato al server e se quest'ultimo mi risponde con il NONCE allora OK;altrimenti occorre inviare il NONCE che verrà firmato dalla banca e, per controllare, bisogna applicare la firma del nonce con quella effettuata da noi con

la chiave pubblica. Se questo confronto va a buon fine allora siamo sicuri che stiamo effettivamente parlando con la banca.
Tuttavia, non sempre tutte le chiavi vengono fornite dal root authority, ma talvolta si utilizzano i cosiddetti certificate chain (il cui procedimento è molto simile alle block chain dei Bitcoin a patto di sostituire le transazioni con i certificati). In particolare, si riesce a legare un certificato all'altro firmando il certificato con il prossimo in fila e solamente l'ultimo è detto trust anchor poiché non viene firmato da nessuno.

22 PKI and Merkle Trees

Il PKI è l'acronimo di Publci Key Infrastructure e rappresenta il framework utilizzato per realizzare i certificati. Comprende:il numero della versione, il nome utente e della CA, la chiave pubblica dell'utente e la firma digitate della CA. Il problema di PKI è che talvolta non sono trust neanche i CA e possono rilasciare dei certificati falsi.
Per ovviare a questa problematica si utilizza un grande DB scalabile, unico e pubblico a cui possonoa accedere sia i server che i clinet. Così otteniamo il certificate transparency. Quando un utente comunica con un server e ne riceve il certificato può verificare all'interno di questo database (controllando anche nella lista dei certificati revocati sia per scadenza del certificato sia per certificati falsi) e risalire alla validità del certificato in ogetto. In aggiunta anche le entità all'interno del DB possonmo verificare la correttezza dei suoi certificati e nel caso in cui ci fossero anomalie possono procedere alla revoca.
Al livello implementativo, per gestire questo immenso database si utilizzano i cosiddetti Merkle Trees. Come idea di base seguono la procedura per il salvataggio di un file di grande dimensione: prima si effettua il fingerprint del file applicando un funzione di hash che mi consente quindi di comprimere il file;poi si applica la firma a digest appena prodotto.
I Merkle Trees funzionano proprio in questo modo: le foglio di questo albero rappresentano i certificati;mentre i nodi interni rappresentano l'applicaizone della funzione di hashing fino ad arrivare alla fine. Inoltre, al momento di verificare la validità di un certificato basta fornire i pezzi di albero corrispondenti alla funzione di hash che porta fino al rootCA e così facendo siamo riusciti a mantenere il DB di piccole dimensioni con la firma data dal rootCA. I Merkle Trees sono unfeseable cioè la probavilità di torvare due hash uguali è trascurabile.
Tuttavia, il difetto di questa costruzione è che è molto statica e quindi occorre renderla dinamica nel seguente modo: ogni giorno si raccolgono un tot.di certificati che vengono hashati e poi aggiunti al Merkle Trees. Bisogna inoltre ricordare che quando si aggiunge un certificato al merkle trees si rilascia un sct che rappresenta il time stamp di valdità del certificato che verrà poi controllato dall'end-user quando necessita di verificare il certificato inviato dal server. Durante la TLS hanshake, oltre a verificare la validità del certificato, si verifica anche la presenza nella lista pubblica.