1. Segreti
    - I segreti possono essere
        - intercettati
        - dedotti (dal ciphertext)
        - **indovinati**
    - Le contromisure sono rispettivamente:
        - mascherare i segreti cifrandoli
        - rendere i segreti indeducibili usando degli algoritmi crittografici robusti
        - usare dei segreti casuali
    - Per quando riguarda l'indovinabilità, la sicurezza dipende da come è stato scelto il segreto (chiave). Se l'utente, per potersela facilmente ricordare a memoria, ha usato un'informazione che l'intruso può prevedere (ad esempio il suo nome, o il suo anno di nascita, o i dati di un suo familiare, ecc.), il tirare ad indovinare ha una notevole probabilità di successo.
    - La prima difesa preventiva è quindi quella di usare dati casuali.
        - R12: "I bit della stringa che rappresenta un dato segreto devono essere equiprobabili ed indipendenti "
        - Ciò richiede la disponibilità di un nuovo meccanismo per la sicurezza, il generatore di numeri casuali o RNG (Random Number Generator)
    - la complessità computazione non mette in conto il caso più favorevole per l'intruso; utile è dunque la adozione di un secondo provvedimento.
        - R13: "un dato segreto deve essere frequentemente modificato".
            - In questa maniera non concediamo all'intruso il tempo di svolgere molte prove.
            - per lo stesso motivo, usare tante volte lo stesso segreto per molti messaggi non è raccomandabile

2. Funzioni di hash crittograficamente sicure
    - proprietà
        - facile da calcolare
        - resistenza debole alle collisioni
            - dato un x e il suo hash H(x) è infattibile trovare un altro y con lo stesso hash
            - altrimenti diventa difficile garantire integrità ed autenticazione
                - si riesce a trovare un m* : H(m*) = H(m)
                - quindi è sufficiente *inviare il messaggio m||H(m||s)**
                    - con HMAC (se il segreto è allineato con la dimensione del blocco)
            - vale anche se si utilizza la firma come autenticatore S(H(m))
            - l'attaccante **non ha neanche bisogno di conoscere il segreto**
        - resistenza forte alle collisioni
            - infattibile trovare due stringhe qualunque che hanno lo stesso hash
            - altrimenti uno potrebbe presentare un documento firmato e in un secondo momento dire che il documento era un altro con lo stesso hash

        - infattibili da invertire
            - altrimenti potrei invertire un HMAC e risalire al segreto -> autenticità rotta
    - numero di bit di un hash
        - bisogna garantire sia resistenza forte che debole
        - è più difficile garantire resistenza forte (paradosso del compleanno mi fa scalare i tentativi con ^(n/2))
        - ho bisogno quindi di 256 bit di hash
    - implementazione con schema a stadi detto compressione iterata per comprimere stringhe arbitrariamente lunghe
        - all'interno di ogni stadi vi è una mini-funzione di hash *f* che calcola il suo hash prendendo gli *r* bit del suo blocco dati e gli __*n* bit dell'hash dello stadio precedente__
        - il primo blocco usa un IV come 'hash dello stadio precedente'
        - **NB**: Se l'ultimo blocco del messaggio m è più piccolo di r bit lo si completa con un **padding**.
            - **Il padding è fondamentale per evitare attacchi con estensione del messaggio**.
            - per questo motivo, **viene introdotto in ogni caso**, anche se il messaggio è già multiplo dei blocchi.
    - attacco con estensione del messaggio
        - Consideriamo una sorgente che, dopo aver concordato un segreto s con la destinazione, le invia un messaggio m ed il suo MAC H(s||m), **calcolato però senza aver aggiunto alla fine di m l'indicazione di quanto è lungo.**
        - L'intruso può in questo caso condurre un attacco di length extension:
            - individua un'estensione m' di m di suo interesse (forma cioè un messaggio m||m') e calcola H(m'),
                - **ponendo attenzione a fornire all'algoritmo come stato iniziale non la costante h0, ma il dato H(s||m) che ha intercettato**.
            - L'impronta H(m') così ottenuta è **per costruzione uguale a quella di H(s||(m||m'))** e quindi **il messaggio forgiato dall'intruso sarà giudicato dalla destinazione come generato dalla sorgente (autentico e integro)**.
            - la destinazione è convinta che il mittente abbia mandato m||m' e non m
        - con il padding attenuo il problema
            - l'attaccante può produrre un messaggio con due padding che più difficilmente ha senso ma potrebbe comunque averlo
            - HMAC / usare H(m||S) risolve il problema

3. Generazione di numeri casuali

    - servono per generare
        - chiavi segrete
        - IV/seed
        - nonce
    - è importante che i bit prodotti siano casuali
        - altrimenti un attaccante puà indovinare
    - Per ottenere verà casualità si usano i TRNG
        - frequenza di generazione bassa
        - riproducibilità impossibile

        - in cifrari a flusso, sorgente e destinazione devono produrre lo stesso flusso di chiave
    - per risolvere i problemi sopra si utilizzano i PRNG, algoritmi deterministici che seguono il modello di ASF
    - PRNG garantiscono casualità, riproducibilità (se uso lo stesso seme produco la stessa sequenze) e alta frequenza di generazione, ma **non imprevedibilità**.
    - la sola casualità non è sufficiente. Occorre, anche l'imprevedibilità: `un intruso che è riuscito ad intercettare l'uscita o ad individuare, in tutto o in parte, lo stato del generatore non deve poter dedurre da quale seme sono partiti i calcoli e/o quali saranno i prossimi valori generati.`
    - Un generatore pseudocasuale che ha anche la proprietà di imprevedibilità è detto **PRNG crittograficamente sicuro**
    - Per conseguire imprevedibilità occorre che:
        - il periodo sia grandissimo (10^50, 10^100), per poterlo suddividere con il seed in moltissime sottosequenze;
        - **il seme sia imprevedibile e tenuto segreto**
            - tipicamente il seme viene generato da un TRNG
        - sia unidirezionale o la funzione di stato futuro, o la funzione d'uscita
            - per rendere impossibile
                - ad un avversario che ha individuato un uscita il risalire allo stato che l'ha generata (e quindi poi risalire al seme)
                - ad un avversario che ha individuato uno stato il risalire agli stati precedenti ed al seme;
    - Riassumendo, abbiamo che PRNG crittograficamente sicuro è caratterizzato dalla produzione di una sequenza di bit
        - casuali
        - imprevedibili in quanto scelti da un seme che seleziona una sottosequenza tra le molte presenti nel periodo grandissimo del PRNG
        - seme deve essere tenuto segreto ed è anch'esso imprevedibile (i punti 2 e 3 sopra servono a questo) altrimenti la sequenza scelta viene svelata
            - per questo si impiega TRNG
    - **NB**: spesso, oltre a casualità ed imprevedibilità, è importante che IV/Seed siano **usati una e una sola volta**, altrimenti
        - two-time key vulnerability in cifrari a flusso
        - messaggi uguali vengono cifrati nello stesso modo in CBC il che abilità chosen plaintext attack (confronto se i cifrati sono uguali)

4. Cifrari a flusso

    - sono in pratica uno xor con un flusso di chiave pseudocasuale
    - il flusso di chiave prodotto si basa su un seed.
        - Questo deve essere scambiato in segreto da sorgente e destinazione
        - e deve essere imprevedibile
        - altrimenti un attaccante può riprodurre lo stesso flusso di chiave
    - possono essere
        - a flusso sincrono -> flusso di chiave generato con un PRNG (funzioni di F e G)
            - meglio per canali rumorosi ma affidabili

        - a flusso autosincronizzante -> flusso di chiave generato con un registro a scorrimento in cui ci finisce in retroazione l'uscita prodotta, e una funzione F
            - meglio per canali non rumorosi ma poco affidabili
    - a causa dello xor si ha
        - two-time-key vulnerability -> c1 ^ c2 = m1^k ^ m2^k = m1 ^ m2
            - non bisogna usare lo stesso flusso di chiave più volte, ovvero, bisogna sempre cambiare il seme (vedi WEP)
        - malleabilità: modifiche al testo cifrato hanno un effetto prevedibile sul testo in chiaro
            - c ^ m' -- viene decifrato come --> m* = m^k^m' ^k = m ^ m'
            - Se l'attaccante conoscesse il messaggio (o se lo immagina, perché m, magari, è fortemente strutturato) potrebbe cambiare parte del testo a suo piacimento.
            - Se l'intrusore pensa che i primi bit siano corrispondenti a "from Bob", e vuole cambiare il mittente, deve trovare quel p che in xor con "from Bob ..." produce "from Eve ..." (facile).
    - se si utilizza un cifrario a flusso sappiamo che è malleabile e quindi è fondamentale implementare meccanismi che garantiscano integrità

5. Cifrari a blocco

    - conclusioni ECB

        - Pro:
            - Alto parallelismo (efficente): i blocchi sono tra loro indipendenti e quindi possono venire cifrati in parallelo
                - ottimo per cifrare roba che sta in un blocco (chiavi) o dati totalmente non strutturati (per cui non si riesce a sostituire blocchi)
            - Non propagazione degli errori.
        - Contro:
            - **Deterministico**, a blocchi identici di testo in chiaro corrispondono blocchi identici di testo cifrato.
            - suscettibile ad attacchi di malleabilità (sostituzione di blocchi cifrati con la stessa chiave intercettati) che di chosen plaintext (controllo due cifrati e vedo se combaciano)
                - funzionano se la chiave è la stessa (nel secondo caso magari qualcuno cifra per me (voto elettronico) o magari posso ingannarlo a farlo)
                - motivo in più per cambiarla spesso
            - ci vuole il padding per l'ultimo blocco

    - Un buon cifrario simmetrico dovrebbe:

        - produrre ciphertext diversi anche partendo da plaintext identici
            - altrimenti posso vedere se i cifrati combaciano (chosen plaintext)
        - mascherare le regolarità del plaintext facendo **dipendere ogni blocco del ciphertext da qualcos'altro oltre corrispondente blocco del plaintext (e la chiave)**
            - altrimenti malleabilità attraverso sostituzione di blocchi
        - questo è quello che fanno le prossime modalità di cifratura

    - CBC

- ogni blocco di cifrato si ottiene cifrando il relativo blocco di plaintext in xor con il cifrato precedente
    - Sorgente (cifratura): c0 = IV, ci = Ek(ci-1 ⊕ mi)
    - Destinazione (decifratura): IV = c0, mi = Dk(ci) ⊕ ci-1 = ci-1 ⊕ mi ⊕ ci-1
- per il primo blocco si utilizza un IV **che cambia sempre**
    - in questa maniera messaggi uguali o che iniziano in maniera identica vengono cifrati in maniera diversa e non si è suscettibili al chosen plaintext attack citato sopra
- In CBC **l'attacco di malleabilità è infattibile**
    - se si prova a sostituire un blocco cifrato tutti i successivi vengono decifrati male
    - in ogni caso è impossibile trovare un blocco cifrato nella stessa maniera dato che l'IV cambia sempre
- modalità di cifratura abbastanza lenta in quanto non parallelizzabile
    - in decifratura è parallelizzabile
- ci vuole padding per l'ultimo blocco

◦ OFB e CFB

- sono implementazioni di cifrari a flusso basati su cifrari a blocchi:
    - OFB realizza un Cifrario a flusso sincrono
    - CFB un Cifrario a flusso con auto-sincronizzazione
- **impiegano la sola trasformazione E per generare un flusso di chiave**
    - identico lato sorgente e lato destionazione
    - notare in queste implementazini, un attaccante che conosce il seed non può comunque riprodurre lo stesso flusso di chiave dato che non ha la chiave k
        - il seed può
- il flusso di chiave è sempre diverso grazie ad un IV che fa da seed
    - 2-time key attack prevenuta
    - anche malleabilità (intesa come sostituzione di pezzi di cifrato) prevenuta dato che non si riescono a trovare due messaggi cifrati con lo stesso flusso di chiave
        - (vale comunque la malleabilità dei cifrari a flusso)
- si può parallelizzare cifrando/decifrando molteplici blocchi alla volta
- non parallelizzabili in quanto ci sono delle retroazioni (flusso di chiave / cifrato precedenti)

◦ Counter

- simile ad OFB però opera su blocchi e al posto di usare una funzione di stato futuro, si incrementa un contatore
    - il comportamento è quindi quello di un cifrario a flusso sincrono (con le relative considerazioni sulla sincronizzazine)
    - i blocchi sono indipendenti e quindi si può parallelizzare
- impiegano la sola trasformazione E per generare un flusso di chiave
    - identico lato sorgente e lato destionazione
- di nuovo non malleabile perchè il seed (IV) non si ripete
    - (di nuovo vale la malleabilità dei cifrari a flusso)
- parallelizzabile operando come con ECB aggiustando i vari counter

- **Beast attack (block injection attack)**
    - se conosco il CBC residue che verrà utilizzato (IV prevedibile) posso costruire un messaggio che annulla roba (xor) e viene cifrato come un messaggio precedente su cui sto facendo ipotesi
        - m1' = la mia ipotesi sul contenuto di un messaggio = Kimberly
        - K = CBC residuo al blocco 1 = cifratura del blocco 1
        - K1 = CBC residuo al blocco 2 = cifratura del blocco 2
        - do come input al terzo blocco m1'^K^K1 ottenendo una cifratura analoga a quella del blocco 1
            - m1^K = m1'^K1^K^K1
    - rompo il non determinismo di CBC e faccio un attacco simile a quello che facevo con ECB
- paradosso del compleanno per cifrari a blocchi
    - La probabilità che un attaccante trovi due blocchi di testo cifrato uguali scala con 2^(n/2) e non con 2^n (paradosso del compleanno)
        - con n dimensione del blocco
    - dopo 2^(n/2) cifratura continuare a cifrare con la stessa chiave non è più consigliato perchè diventa probabile produrre cifrature uguali
    - se produce cifrature uguali posso sfruttare le proprietà dello xor (varia in base alla modalità di cifratura in questione) per attuare un **two time key attack**
    - un cifrario a blocchi è sicuro se e solo se i blocchi hanno dimensione >= 128 bit

6. Scambi di chiave con Master Key e KDC
    - Master key
        - due corrispondenti precondividono una master key con cui cifrano chiavi di sessione ogni volta che comunicano
            - la master key viene condivisa con un costoso incontro diretto
        - vi è il pericolo che cifrando troppi messaggi con la stessa master key si incorra in un attacco di crittanalisi
        - per questo motivo la master key ha una vita limitata, seppur lunga
        - questo schema è costoso in qunto richiede un canale out-of-band per lo scambio della master key è non scala (O(N^2))
    - KDC
        - in questo schema si sfrutta un'autorità FIDATA **con cui tutti gli utenti della comunità concordano individualmente una chiave segreta**.
            - permette di avere comunità in cui **il numero di chiavi è pari al numero di utenti**
        - A tale Autorità compete:
            - la **generazione di una chiave di sessione per ogni coppia di utenti che intendono comunicare**
            - il **suo invio in modo riservato agli interessati**
                - KDC cifra la chiave di sessione con il segreto preconcordato dell'interessato
        - in questo schema è importante evitare gli attacchi con replica per avere la corretta identificazione della parti
            - interessante l'utilizzo della cifratura con un segreto come strumento di identificazione

                - solo se sei chi dici di essere saprai decifrare questo messaggio contenente la chiave di sessione
                - usato anche in kerberos
            - inoltre una volta condivise la chiavi di sessione, è importante che i due interlocutori si identifichino a vicenda, sempre con sfide e risposte fatte a colpi di nonce
                - in particolare, si può verificare se la mia controparte è in grado di usare la chiave di sessione decisa dal KDC. Questo identifica in quanto solamente chi possiede il segreto preconcordato con il KDC è in grado di farlo.
    - Infine, ho problemi se implemento le cifrature in modalità ECB?
        - per cifrare chiavi di sessione la modalità ECB va bene dato che stanno in un blocco, ma ...
        - determinismo
            - se al passo 5 non si modifica di molto R_B l'attaccante può provare a forgiare una risposta nel tentativo di impersonare qualcun'altro
        - malleabilità
            - **con ECB e_k(A||B) diventa e_k(A)||e_k(B) se i blocchi sono allineati**
            - Un A malevolo potrebbe spacciarsi per chi vuole sostituendo il blocco relativo all'identità nel passo 3!
    - Questo modello scala linearmente in una comunità di utenti, ma se si vuole avere una rete mondiale di KDC per fare comunicare tutti con tutti, si ritorna a punto e a capo con un numero di chiavi precondivise tra i KDC che scala con O(N^2)

7. Scambi DH

    - DH anonimo
        - L'obiettivo dello scambio DH è far si che due utenti qualsiasi A e B, **senza aver preso alcun precedente accordo segreto** (no master key sia essa in KDC o meno), riescano a **condividere un dato segreto K (chiave di sessione)** dopo aver calcolato ed essersi **scambiati senza alcuna segretezza due dati YA e YB (pubblici)**.
            - Non abbiamo più una terza parte fidata (KDC) e nemmeno una Master Key pre-concordata.
        - A tal fine, è necessario che ciascuno dei due utenti **scelga a caso un numero X (che terrà segreto)** ed usi poi una **funzione unidirezionale F** per calcolare il numero **Y = F(X) da inviare al corrispondente**
            - in questo modo, l'intruso che riesce ad intercettare Y non dispone di algoritmi efficienti per calcolare il segreto X = F^-1(Y).
        - Una volta avvenuto lo scambio (A manda YA e B YB), il metodo prevede che A e B dispongano di una **particolare funzione G** che garantisca ad entrambi di **ottenere lo stesso risultato K (chiave di sessione)** a partire dai dati in loro possesso:
            - **G(XA,YB) = G(XB, YA) = K**
            - **NB**: ho comunicato solo roba pubblica YA, YB, ma **HO CONCORDATO UN SEGRETO CONDIVISO**
        - Il protocollo, nella sua versione più semplice detta DH anonimo, prevede quindi solo due passi:
            1. **Generazione delle chiavi segrete XA, XB e pubbliche YA, YB**
                - A e B, dopo aver generato a caso rispettivamente i numeri
                    - 1 < XA < p-1

            - 1 < XB < p-1
        - (che tengono segreti), calcolano
            - YA = g^XA mod p
            - YB = g^XB mod p
        - e si scambiano i risultati in modo non riservato.
    2. **Calcolo del segreto K**
        - A e B calcolano rispettivamente:
            - KA = YB^XA mod p = (g^XB mod p)^XA mod p = (g^XB)^XA mod p
            - KB = YA^XB mod p = (g^XA mod p)^XB mod p = (g^XA)^XB mod p
        - **KA = KB = K**
- I dati **p e g, NON sono segreti**, devono essere **noti ed uguali per entrambi** e quindi vengono comunicati in chiaro.
    - Ad esempio chi inizia il protocollo può deciderli e comunicarli al corrispondente insieme al suo dato Y
    - in fixed DH sono presenti nel certificato
- **NB**: Se si vuole generare ogni volta una chiave di sessione diversa, bisogna rigenerare ad ogni sessione i dati X e Y
    - in alternativa, è possibile generare solo una volta i dati X e Y concordando così un pre-master-secret sempre uguale
    - sarà sufficente che i due interlocutori si scambino ognuno un proprio nonce.
    - Questi due nonce verranno concatenati al pre-master-secret e infine verrà fatto l'hash di tutto per generare il master-secret che cambia sempre
        - master_secret = H(pre_master_secret || RC || RS)

◦ DH non anonimi
- ci si avvale dei certificati per rendere lo scambio non anonimo (vogliamo sapere chi è che ci sta mandando il dato Y)
- **NB**: TLS sfrutta questi scambi per autenticare gli interlocutori
- **Fixed DH**
    - il dato pubblico <p, g, Y> di ciascun utente è comunicato all'altro tramite un certificato X.509v3.
    - il resto dello scambio è analogo alla modalità anonima
    - notare che dato che i dati pubblici sono in un certificato, essi non cambiano mai, per non concordare ogni volta la stessa chiave di sessione si usa il trucco coi nonce per generare un master-secret a partire dal pre-master secret
    - **NB**: la CA autentica la tripla, ma nello scambio non c'è una PoP che mi identifica il mittente; è dunque possible che un attaccante presenti un certificato che non è il suo
        - poco male, non avendo il dato segreto X non può concordare la stessa chiave di sessione
        - non è comunque il massimo dato che (se non si controlla di avere concordato la stessa chiave come in TLS) si potrebbe mandare del testo cifrato ad un attaccante
- **Ephemeral DH**
    - analogo a scambio anonimo, solamente che gli interlocutori inviano il loro dato pubblico Y firmato, il ricevente può autenticare verificando la firma

                - come sempre, bisogna che gli intelocutori si scambino i loro certificati in maniera da fornire all'altro la possibilità di verificare la firma
            - qua X e Y cambiano sempre e quindi la chiave di sessione concordata è sempre diversa (per compatibilità si usano comunque i nonce)
            - anche qua non c'è identificazione, dato che un attaccante potrebbe replicare un Y firmato intercettato in comunicazioni precedenti
                - di nuovo poco male dato che non avendo X non riesce a concordare la stessa chiave
            - **NB**: è importante distinguere il requisito di autenticazione da quelle di identificazione
                - il primo richiede che i messaggi che arrivano siano effettivamente appartenenti a Bob
                    - associazione paternità del dato
                - il secondo mi richiede che i messaggi che arrivano siano stati mandati proprio da Bob
                    - le repliche sono per definizione autentiche, ma non vanno bene dato che chi me le manda non è chi ha creato il messaggio

8. Certificati

    - i certificati vengono firmati da una terza parte fidata, questo garantisce l'autenticità tra l'associazione di identità e chiave pubblica presente nel certificato
    - per identificare correttamente un interlocutore che non si conosce a priori, è necessario:
        - sia il certificato -> che associa una identità ad una chiave pubblica
        - che la POP -> che dimostra che che il mittente è il proprietario della chiave pubblica presentata (e per transitività, identifica il mittente con l'identità presente nel certificato)

9. CRL e OCSP

    - CRL = listone di tutti i certificati revocati
        - viene emessa ed aggiornata periodicamente
        - struttura simile ad un certificato nel senso che è firmata da una CA ed ha un periodo di validità (fino alla prossima emissione)
        - Può diventare molto grande è quindi è importante adottare delle tecniche che consentono di ridurre la dimensione/la quantità di dati da trasferire all'utente
            - elimino le entry relative a certificati scaduti nell CRL successive alla loro data di scadenza
                - sono scadute non c'è bisogno di manterle
                    - se il certificato era stato revocato elimino la entry
                    - se non era stato revocato non la aggiungo neanche
            - pubblico i diff -> aiuta solo la bandwidth
            - sotto-liste -> si grazie
        - problema della freschezza, tra un aggiornamento e l'altro potrebbero essere stati revocati dei certificati -> c'è anche OCSP
    - OCSP
        - l'utente può avere **informazioni in tempo reale circa lo stato di revoca di un certificato grazie ad un server sempre online**.

        - OCSP È un protocollo “client-server” in modalità “pull” che **funziona solo online** (al contrario delle CRL che una volta scaricate possono essere consultate anche offline).
        - data una richiesta mi risponde con lo stato di revoca di un certificato
        - OCSP attinge informazioni dalle CRL ma non solo!
            - magari chiede direttamente anche ad una CA (che riceve le richieste di revoca)
        - **Le risposte sono firmate dal server, non dalla CA!**
            - Il server OCSP avrà una coppia di chiavi certificata da una CA.
        - OCSP viene quindi usato principalmente per due motivi:
            - Freschezza delle informazioni;
            - Un utente non si deve scaricare l'intera CRL ma solo l'informazione sullo stato di revoca del certificato richiesto.

10. Gerarchia delle CA

    - non tutti gli utenti vengono certificati dalla stessa CA
    - le CA sono in relazione gerarchica, questo permette di trasferire la fiducia
        - se io mi fido della mia CA e la mia CA si fida di CA_B, allora anche io mi fido di CA_B
    - bisogna trovare le catene di fiducia (catene di certificati) fino ad una root-CA e verificarle
    - cross-certificates sono certificati che certificano la chiave pubblica di un altra CA
        - certificano i nodi intermedi del percorso di fiducia

11. Identificazione passiva

    - ripeto sempre lo stesso dato come prova di identità
    - sempre suscettibile ad attacco di intercettazione e replica -> assolutamente inadatto a sistemi in rete
    - per lo stesso motivo è anche impossibile fare identificazione reciproca
        - un MIM può aspettare che io gli invii le mie credenziali, non rispondermi, replicare le mie credenziali verso il destinatario legittimo spacciandosi per me

12. Identificazione attiva

    - l'identificando manda una prova di identità sempre diversa
    - abbiamo visto 3 varianti:
        - one-time pwd
        - sfida/risposta
        - zero-knowledge proofs

13. Identificazione attiva con sfida/risposta

    - esistono 3 varianti
        - hash con segreto preconcordato (pwd modem)
        - firma asimmetrica
        - cifratura asimmetrica
    - tutte e 3 le varianti usano come sfida un nonce in modo da evitare gli attacchi con replica
        - **NB**: il nonce è generato da un PRNG e come al solito occorre che sia **imprevedibile**. Se il nonce fosse prevedibile l'attaccante potrebbe spacciarsi per l'identificatore B e farsi mandare da A *H(S || RB')*; con RB' = nonce previsto dall'attaccante che B vero produrrebbe nella prossima sfida. A questo punto l'intrusore può spacciarsi per A con B

nella prossima sessione dato che conosce l'hash che verrebbe usato da A alla sfida con RB'.

14. Identificazione mutua con sfida e risposta

- se si usa la variante con hash e segreto precondiviso non si può identificare prima uno e poi l'altro come ci si aspetta
- ho il problema che si possono tenere aperte due sessioni di identificazione
  - **attacco di interleaving (gran maestro scacchista)**: apro due sessioni contemporanee con due entità diverse
  - **Attacco di reflection**: apro due sessioni contemporanee con la stessa entità
  - in entrambi gli attacchi l'attaccante può rigirare messaggi ottenuti da sessioni diverse in cui svolge sia il ruolo di identificatore che di identificando
  - con un protocollo ingenguo, si lascia il tempo all'attaccante di svolgere più sessioni (in particolare, può aspettare che uno tenti di identificarsi presso di lui)
    - primo accorgimento utile è quindi limitare il periodo di validità delle sfide
  - un ulteriore accorgimento è rompere la simmetria dei messaggi nel protocollo e linkarli aggiungendo dentro all'hash nelle risposte
    - entrambi i nonce della sessione
    - e anche il destinatario della risposta
  - un ultimo accorgimento è quello di numerare le identificazioni (seq number) tra i due interlocutori (sia per inteleaving che per reflection).
    - in questa maniera se un attaccante ripropone una sfida/risposta il seq number non combacia

15. Cifrari asimmetrici

- è importante che il numero corrispondente al blocco binario, sia minore del monulo n. Questo perchè ad un testo in chiaro deve corrispondere uno e un solo testo cifrato; se se si eccede il modulo due blocchi di plaintext possono venire cifrato nello stesso modo dato che il modulo fa il giro
- **NB**: un Cifrario asimmetrico è almeno **mille volte più lento** di un Cifrario simmetrico ed è quindi fortemente auspicabile impiegarlo con messaggi “corti”.
  - perchè?
    - dobbiamo fare delle esponenziazioni modulari e generare numeri primi grandi!
    - nel caso simmetrico devo fare solo trasposizioni e sostituzioni (con xor)
  - Ciò non ne limita l’utilità, dato che il suo uso tipico è la comunicazione della chiave di un Cifrario simmetrico (vedi cifrario ibrido) oppure la firma di messaggio per appendice.
    - il plaintext sta dentro ad un blocco
- **NB**: è ancora più importante che qui i cifrari siano probabilistici dato che non c'è neanche bisogno di fare intercettazione per attuare attacchi di chosen-plaintext / malleabilità con sostituzione di blocchi (ricorda ECB)
  - la chiave pubblica è a disposizione di tutti e quindi tutti possono cifrare
  - ricordiamo che:
    - un cifrario è deterministico se plaintext identici producono ciphertext identici
    - è probabilistico se quanto detto sopra non è vero dato che viene impiegato un numero casuale per ottenere cifrati distinti
- Per eliminare questi punti deboli, il Cifrario deterministico deve essere “randomizzato”.

        - Se il **messaggio è più lungo del modulo**, si impiega tipicamente la **modalità CBC ed un IV casuale**.
        - Se il **messaggio è più corto del modulo**, ed è questo il caso di maggiore interesse (chiavi), il mittente, seguendo uno standard ben preciso indicatogli dal destinatario, **gli aggiunge in testa un padding contenente un numero a caso e poi cifra il tutto**.
    - la dimensione del messaggio influisce anche in come bisogna impiegare il cifrario
        - se m > n
            - m deve essere frammentato in blocchi inferiori a 1024 bit, altrimenti a più testi in chiaro potrebbe corrispondere lo stesso cifrato
        - se m < n
            - È assolutamente necessaria l'aggiunta di un padding (tipicamente OAEP) per estendere il numero di bit iniziale fino alla lunghezza del modulo.
            - Se non usassimo il padding, il messaggio essendo più corto di 128 bit sarebbe vulnerabile ad un attacco con forza bruta

16. RSA

    - cifratura -> c = m^e mod n
        - *e* ed *n* formano la chiave pubblica
    - decifratura -> m = c^d mod n = m^(e*d) mod n = m^1 mod n
        - *d* ed *n* forma la chiave privata
        - d inverso moltiplicativo di e -> e*d = 1
    - **è importante nascondere anche p e q oltre a d!**
        - altrimenti si riesce a calcolare prima *e* e poi *d*
        - Non a caso dopo la generazione della coppia di chiavi solitamente vengono distrutti
    - RSA è deterministico, per renderlo probabilistico si aggiunge del padding con lo standard OAEP
    - il tempo di esponenziazione modulare scala con il cubo della dimensione in bit della chiave (1024 -> 2048 -> 8 volte il tempo)
        - è desiderabile usare l'algoritmo più efficente possibile per questo calcolo
            - Repeated square and multiply per cifratura
            - per la decifratura, anche Algoritmo di Garner
        - per l'inviolabilità del Cifrario asimmetrico le chiavi devono essere molto lunghe, ma questo rende poi molto onerosi i calcoli di cifratura e di decifrazione; **bisogna quindi non esagerare mai nel dimensionamento delle chiavi**.
    - RSA possiede la proprietà moltiplicativa
        - Consideriamo un messaggio m = m1 × m2.
        - La sua cifratura è: c = m^e mod n = ((m1^e mod n) × (m2^e mod n)) mod n
        - La cifratura del prodotto di due messaggi è uguale al prodotto dei due testi cifrati.
        - Analogamente, si ha che la decifrazione di un testo cifrato ottenuto moltiplicando due testi cifrati è uguale al prodotto dei due corrispondenti testi in chiaro
    - RSA è reversibile, posso scambiare il ruolo delle chiavi per effettuare delle firme
        - Nota: come per la chiave di sessione in un cifrario ibrido, l'hash ha un numero di bit molto minore del modulo n ed è quindi ritenuto insicuro trasformarla direttamente. Di nuovo si usa del paddding.

17. Attacco con proprietà moltiplicativa

- si suppone che l'intruso possa richiedere al proprietario della chiave privata la decifrazione di qualsiasi messaggio di sua scelta, ad esclusione del messaggio c di suo interesse.
- Un intruso che ha intercettato un cifrato c di cui vuole scoprire il plaintext m, può
  - costruire un cifrato c' ottenuto come prodotto di c e qualcos'altro che gli pare: c' = c * r^e
  - far decifrare questo c'
  - annullare il termine in eccesso ed ottenere la decifratura che gli interessa
- Contromisure:
  - difficile però che l'intrusore riesca a convincere il proprietario a decifrare qualsiasi messaggio di sua scelta sopratutto se si ha identificazione
    - **NB**: se però una stessa coppia di chiave viene usata per cifrare e firmare uno potrebbe presentare il cifrato come messaggio da firmare, e la firma lo decifrerebbe a causa della reversibilità di RSA
  - se si usa padding (OAEP) la proprietà moltiplicativa non vale più

18. Cifrario ibrido

- posso decidere una chiave di sessione e comunicarla al mio interlocutore cifrando la chiave di sessione con la sua chiave pubblica
- chiaramente la sua chiave pubblica deve essere autentica e quindi devo avere il relativo certificato
- cifrare messaggi molto più piccoli della dimensione delle chiavi in RSA non è sicuro, per questo motivo, quando si cifrano chiavi simmetrica in un cifrario ibrido è importante adottare il padding (OAEP)

19. Firma digitale

- Autentica un dato
  - ovvero, crea dunque un attendibile e verificabile **collegamento tra il dato e la persona/identità che l'ha creato**.
- ha validità legale (quella con appendice)
- due schemi:
  - con recupero -> cifro con la chiave privata, suddivedendo in blocchi se necessario
  - con appendice -> hash del documento e poi firmo solo l'hash
- molto più efficiente lo schema con appendice dato che devo firmare (cifrare con RSA) un solo blocco
- la firma con recupero poi non ha validità legale dato che posso intercettare firme di vari documenti e relativi documenti e cominciare a fare sostituzione di blocchi mantenendo la validità della firma -> posso presentare come autentico un documento che ho modificato
  - la firma con recupero va anche bene se firmo un solo blocco -> questo è il caso di RSA

20. **Firma cieca**

- possible grazie alla proprietà moltiplicativa di RSA
- Siccome la firma è una cifratura, vale come prima che **la firma del prodotto di due messaggi è uguale al prodotto delle due firme**
- Tale tecnica risulta particolarmente utile nella situazione in cui un utente X può **richiedere ad un'Autorità T di autenticargli un messaggio m, di cui, però, non vuole rivelare il contenuto**.

        - vedi voto elettronico in cui una terza parte mi firma il mio voto senza che sappia quale sia
    - un utente può quindi
        - oscurare un messaggio moltiplicandolo con qulcosa che sa 'r' lui cifrato con la chiave pubblica della terza parte
        - presentare il prodotto a T per farselo firmare
            - la firma decifrerà r
        - eliminare r ed ottenere il messaggio firmato senza che la terza parte abbia mai visto il contenuto
    - **NB**: siccome la terza parte firma, e quindi si assume una responsabilità, solitamente la terza parte impiega anche un protocollo di identificazione per capire chi è che mi sta chiedendo di firmare della roba.
        - Questo elimina l'attacco con proprietà moltiplicativa
    - **A che cosa serve la firma cieca?**
        - in sostanza, ad autenticare i dati mantenendo però l'anonimato
        - La firma dell'autorità dice:
            - Questo messaggio è valido
            - Proviene da una persona autorizzata (perchè T ha identificato)
            - Non è stato modificato
        - Perché serve oscurare il messaggio nella fase di firma?
            - Se l'autorità vedesse il contenuto del voto quando lo firmi, saprebbe per chi voti
                - violazione della segretezza del voto.
            - Con la firma cieca l'autorità non sa cosa ha firmato
            - Tu rimuovi l'offuscamento e ottieni il voto firmato, pronto da inserire nell'urna
            - Chiunque controlla la firma sa che è un voto valido, **ma non sa chi è il proprietario del voto**
        - Conclusione: se firmassi io, tutti saprebbero per chi ho votato. La firma cieca mi permette di mantenere il mio anonimato

21. **Attacchi alla firma con RSA**

    - **attacco alla firma basato su proprietà moltiplicativa**

        - Supponiamo che l'intruso **possa richiedere al firmatario X di firmare qualsiasi messaggio di sua scelta, ad esclusione del messaggio m di suo interesse** (questo scenario è detto chosen message attack).
        - Per ottenere la firma di m è sufficiente che
            - l'intruso generi un numero a caso r
            - costruisca m1 = m × r
            - calcoli m2 = r^-1
            - chieda ad X di firmare entrambi i messaggi
            - e moltiplichi infine le firme di m1 e di m2 che gli vengono restituite.
                - Ricordiamo: **La firma del prodotto di due messaggi è uguale al prodotto delle due firme**
                - Abbiamo quindi che S(m1) * S(m2) = S(m1 * m2) = S(m * r * r^-1) = **S(m)**
                - **X ha firmato inconsapevolmente un messaggio arbitrario dell'attacante**

    - **attacco alla riservatezza facendo firmare un cifrato**

- L’intruso sfrutta la proprietà moltiplicativa
  - oscura c in modo che il firmatore non si accorga che firmando decifra qualcosa,
  - se lo fa firmare da X (il che corrisponde alla decifrazione con la chiave privata),
  - elimina il numero a caso ed ottiene il testo in chiaro che non doveva poter conoscere.
- Vale dunque la seguente regola: `chi impiega RSA per decifrare e per firmare deve utilizzare due differenti coppie di chiavi`
  - in questa maniera, firmare un cifrato non lo decifra dato che le coppie di chiavi sono diverse

22. PGP

- interessante come servizio in cui il modello di fiducia (autenticità della chiave pubblica) è decentralizzato
- permette a interlocutori che non si conoscono di scambiarsi messaggi cifrati e autentici
  - per autenticità si fa una firma con appendice
  - per riservatezza si fa una cifratura simmetrica e il mittente scambia la chiave di sessione mediante cifrario ibrido
- Il PGP si prende totalmente in carico la gestione delle chiavi dell’utente e dei suoi corrispondenti
  - Nel **portachiavi pubblico** sono alloggiate le sue **chiavi pubbliche** e quelle **dei suoi corrispondenti**.
  - Nel **portachiavi privato** sono alloggiate, **cifrate**(con passphrase) , le sue **chiavi private**
- I messaggi che si scambiano i due mittenti PGP contengono quindi degli id di coppia di chiave
  - quando un mittente cifra la chiave di sessione, cifra con la chiave pubblica del destinatario, specificando l'id della coppia di chiave, il destinatario saprà quale chiave privata nel suo portachiavi dovrà usare per decifrare
  - quando un mittente firma, firma con la sua chiave privata, specificando l'id della coppia di chiave, il destinatario saprò quale chiave pubblica nel suo portachiavi dovrà usare per verificare la firma
- PGP usa modello di fiducia decentralizzato, senza certificati, alternativo all’ Autorità di certificazione.
  - non c'è nessun certificatore
  - Ma allora, chi mi da la garanzia che le chiavi pubbliche siano effettivamente del destinatario dichiarato?
  - C'è una componente soggettiva (quanto mi fido io di una determinata chiave pubblica) ed una componente esterna data dalle firme di altri utenti
    - se io mi fido della chiave di A, e A si fida della chiave di B (e A mi firma la chiave di B come dimostrazione della sua fiducia), allora io mi fido anche della chiave di B.
    - la fiducia in questo sistema è sempre data da una componente soggettiva. Per questo motivo non ha alcuna validità legale (nessuno si assume responsabilità)
    - Il livello di fiducia attribuito per una determinata chiave dipende dalla fiducia nei confronti dei firmatari
  - A tal proposito una chiave pubblica può essere ricevuta:
    - Personalmente/direttamente -> massima fiducia
    - Tramite intermediario/indirettamente (via mail, pubblicate su un sito, pubblicate su un db, ...) -> meno affidabile (magari MIM)

        - Le strutture dati salvate all'interno del portachiavi pubblico sono quindi simili a certificati, non chiavi pubbliche crude.
            - abbiamo una identità
            - associata ad una chiave pubblica
            - seguita da, non una singola firma di una CA fidata, ma da molteplici firme di utenti di cui mi posso fidare più o meno (magari anche firma del proprietario, autocertificazione)
        - PGP calcola automaticamente sulla base di Owner Trust e le varie Signature Trust il livello di fiducia della chiave (fidata, non fidata, incerta, ...).
            - A intervalli regolari questo campo viene automaticamente ricalcolato da PGP: infatti, un peer potrebbe passare da fidato a non fidato (perché magari è stata compromessa la sua chiave privata).
        - L'utente può, se vuole, firmare la chiave ricevuta ed inviare poi la sua certificazione al proprietario della chiave e/o ad un database di chiavi PGP. Da questa spontanea collaborazione tra gli utenti discende una rete di fiducia (web of trust), che consente di trasferire da una chiave all'altra la confidenza sulla sua autenticità
        - In questo modello si rinuncia ad alcuni vantaggi della CA
            - tempo di revoca molto più lungo
                - l'informazione di revoca si deve propagare nella rete di fiducia
            - tempestività molto bassa

23. TSS

- A noi interessano le marche temporali impiegate insieme alle firme digitali.
- Le firme digitali spesso dipendono da una marca temporale:
    - per validità della firma (scadenza del certificato)
    - data di creazione o quantaltro (pensa a momenti di invio di un messaggio per una graduatoria)
- Quando firmi digitalmente un documento, la firma garantisce due cose:
    - Integrità – il contenuto non è stato modificato dopo la firma.
    - Autenticità – la firma è stata fatta proprio da quel titolare del certificato.
- Ma non garantisce di per sé la data e l'ora reali della firma
    - Infatti, la data scritta dentro il documento o anche quella che il software di firma mostra può essere presa dall'orologio del computer del firmatario
    - ma quell'orologio può essere impostato a piacere, non da alcuna garanzia sull'istante reale di firma
- Se hai solo la firma digitale senza marca temporale sicura, in un contenzioso qualcuno potrebbe dire:
    - "Io l'ho firmato prima che scadesse il mio certificato!", oppure
    - "L'ho firmato quando il contratto era ancora valido!"
    - e sarebbe difficile dimostrare/rifiutare queste affermazioni
- Con la marca temporale sicura, invece, un terzo fidato certifica (con a sua volta una firma):
    - "Questo documento era così alle 15:23 del 12 agosto 2025" e questa data non è falsificabile.
- In pratica, la marca temporale serve per:
    - Dimostrare quando un documento è stato firmato o esisteva in quella forma

        - Rendere valida nel tempo una firma digitale anche dopo la scadenza o revoca del certificato
            - si sa se la firma è stata apposta prima o dopo la scadenza del certificato
        - Evitare dispute su date falsificate
    - chi riceve un dato firmato con una marca temporale sicura è in grado di
        - conoscere con esattezza la data di creazione del dato (di nuovo vedi graduatorie)
        - accertare che a tale data la chiave di firma era valida.
    - La soluzione usuale è prevedere che TSS sia un Ente fidato (ci fidiamo che lui firmi sempre con le misurazioni del tempo prese da TAI/UTC) e che la chiave di verifica della sua firma sia certificata.

24. Kerberos

    - Kerberos è un servizio di Autenticazione per un ambiente client/server

        - Consente quindi a un utente tramite la propria workstation (comunità di utenti tipicamente piccola, ambito di tipo aziendale) di autenticarsi mutuamente su un server (tra tanti disponibili) e accedere al servizio fornito.

    - Il servizio di autenticazione si ispira al modello del KDC

        - impiega solo meccanismi a chiavi simmetriche (cifrari simmetrici)
        - e richiede la presenza in linea di una terza parte fidata (auth server).
        - come il KDC è adatto solo a domini limitati con utenti conosciuti a priori, come in azienda dato che necessita di una serie di precondivisione di segreti

    - ragionamento iniziale | dove posiziono il servizio di autenticazione?

        - sulle workstation?
            - ogni workstation dovrebbe tenere traccia delle prova di identità di tutti gli utenti
            - ci dovrebbe essere una relazione di fiducia tra le workstation e i server
            - scala con n^2
            - e devono essere aggiunte/tolte se vengono aggiunte/tolte delle workstation/server
            - non scalabile
        - sui server
            - uhm, stessi problemi in realtà
        - Idea: non ha senso distribuire su tutte le macchine le funzionalità di autenticare gli utenti, utilizziamo un unico server di autenticazione centralizzato.
            - relazione di fiducia tra i server ed il server di autenticazione (singola).
            - I server accettano tutti i messaggi autenticati dal server di autenticazione
            - permette Single Sign On
                - credenziali singole per accedere a tutti i servizi

    - Il servizio di autenticazione è implementato tramite due server particolari:

        - AS (Authentication Server):
            - memorizza la password (segreti precondivisi) ed i diritti (autorizzazioni ai vari servizi) di tutti gli utenti

        - lancia una sfida d'identificazione a chi inizia una sessione di lavoro presso una qualsiasi delle workstation e gli restituisce un documento cifrato (ticket_tgs) che dovrà presentare a TGS al fine di ottenere specifiche autorizzazioni.
            - la sfida di identificazione contiene il ticket tgs cifrato con la password dell'identificando (identifichiamo con cifrari simmetrici come in KDC)
    - TGS (Ticket-Granting Server):
        - condivide una chiave segreta con AS ed una con ogni altro server V
        - decodifica e verifica il documento di AS che l'utente gli invia
        - esamina la correttezza della risposta alla sfida lanciata da AS e, se tutto è "a posto", restituisce all'utente un diritto d'accesso (ticket) al server V di suo interesse valido per tutta la sessione di lavoro.
            - se l'utente, all'interno di una stessa sessione, vuole accedere più volte allo stesso servizio, non dovrà ripetere l'autenticazione, ma semplicemente continuare ad utilizzare il ticket che gli ha fornito TGS (se non è scaduto) per accedere a V.
            - Analogamente se vorrà accedere a un altro servizio dovrà richiedere un nuovo ticket a TGS, senza interpellare AS (se l'autenticatore fornitogli da AS non è scaduto);
        - **NB**: AS mi identifica sfidandomi a decifrare con la chiave segreta di chi dico di essere il ticket_tgs che mi manda.
        - Se io però, dopo aver decifrato (ed essendomi quindi identificato) mando a tgs il ticket_tgs in chiaro, questo è suscettibile a replica e quindi essermi identificato non è servito a nulla
        - bisogna dimostrare di essere il legittimo proprietario di un ticket eliminando così gli attacchi con replica
            - solo chi si è identificato presso AS superando una sfida è in grado di produrre un autenticatore corretto
        - il Client produce un autenticatore cifrando un messaggio con una chiave di sessione fornitagli da AS quando si è identificato
            - TGS ottiene la stessa chiave di sessione con cui può verificare l'autenticatore siccome è contenuta dentro al ticket
            - l'autenticatore contiene un timestamp con cui si può controllare se è fresh ed evitare quindi la sua replica
            - autenticatoreC serve per far sapere a TGS che **chi ha inviato il ticket è veramente C**
                - **l'autenticatore dimostra che l'utente è riuscito a superare la sfida di AS**
                - è un dato che protrae l'identificazione nel tempo
    - **NB**: autenticatori e timestamp vengono usati anche tra C e V una volta che TGS ha fornito a C ticket_v.
    - il protocollo si completa con l'identificazione di V presso C
        - questa avviene con l'estrazione della chiave di sessione da ticket_V che usa per rispondere ad una sfida di C
        - solo il v vero ha la chiave segreta con cui può fare questa operazione

- **Considerazioni**

- Completato il protocollo il client C inizia ad avvalersi dei servizi del server V.
    - Se successivamente l'utente ha ancora bisogno di V il protocollo ricomincia dal passo 5 (ripresentando ticketV)
        - basta rigenerare un nuovo autenticatore (T5 fresh) e che il ticketV sia ancora valido ([t4, t4+d])
    - Se durante la sessione l'utente ha bisogno di accedere anche ad un altro server, il protocollo deve essere riavviato dal passo 3 (ripresentando ticketTGS)
        - C è già stato identificato da AS
        - bisogna rigenerare un autenticatore (T3 fresh) e chiedere un ticket_v per il nuovo V
- identificazione mutua serve per
    - evitare di scambiare informazioni con un attaccante
    - evitare intercettazione e replica dei ticket
        - un intruso che finge di essere un server V può intercettare il ticket e provare riutilizzarlo
        - nota che mentre ticket_tgs ha un autenticatore, ticket_v non ce l'ha e quindi la replica funzionerebbe
- **per quale motivo ci sono i timestamp**
    - prevengono gli attacchi con replica
    - se un autenticatore non è 'fresh' significa che è stato replicato e quindi non è da considerare valido
    - **NB**: timestamp != da periodi di validità: in kerberos i periodi di validità possono essere anche lunghi perchè non sono quest'ultimi a prevenire gli attacchi con replica come nel protocollo precedente
- **perchè c'è l'autenticatore (utile da generalizzare)?**
    - anche questo previene gli attacchi con replica
    - permette di dimostrare di essere il legittimo proprietario di un ticket
    - solo chi si è identificato superando una sfida è in grado di produrre un autenticatore corretto
- **cos'è un ticket?**
    - i ticket sono **permessi di accesso**
        - al TGS
        - ai vari V
    - permettono ad un utente di **non doversi riautenticare un sacco di volte**
        - basta ripresentare un ticket (se ancora valido) per poter riusufuire dei servizi
- **Cos'è un authentication server?**
    - È un server che verifica l'identità di un utente (lo identifica) o di un servizio (OAuth) quando tenta di accedere a una risorsa come:

                - rete aziendale
                - applicazione web
                - database
                - VPN
                - servizio cloud
            - In pratica, il client dice: “Sono Alice” e il server risponde: “Dimostralo”.
            - fondamentalmente il “buttafuori digitale” di un sistema informatico:
                - controlla chi può entrare e che permessi ha una volta dentro.
            - A cosa serve?
                - **Centralizzazione**
                    - un unico punto per gestire credenziali e permessi.
                    - non devo replicare per ogni servizio in cui mi autentico
                - **Single Sign-On (SSO)**
                    - un login per più servizi, riducendo password multiple e login multipli.

        - L'autenticazione di Kerbers è fortemente centralizzata dato che c'è solo un AS.

            - Questo è contrapposto all'autenticazione federata in cui gli AS (IdP) sono molteplici
            - (questa sembra essere l'unica differenza)

25. TLS/SSL

    - servizio semitrasparente che garantisce alle applicazioni identificazione (anche mutua) degli interlocutori e autenticità, integrità e riservatezza dei dati
        - semitrasparente dato che le applicazioni devono comunque usare socket sicure e non normali, delegano però tutta la gestione della sicurezza al livello TLS
    - Insieme di protocolli, a noi ne interessano due:
        - Handshake
            - si occupa della **negoziazione** dei meccanismi (detti anche parametri) di sicurezza (algoritmi, come scambiare la master key, ...), dell’**identificazione** (anche reciproca) tra client e server e dello **scambio della chiave** simmetrica di sessione
            - la chiave di sessione scambiata è una master key da cui vengono derivate altri 6 segreti
            - Per ogni direzione è, infatti, richiesto:
                - un primo dato segreto per autenticare/verificare l’impronta di ogni blocco di dati scambiato (HMAC),
                - un secondo dato segreto per inizializzare il Cifrario simmetrico che cifra/decifra i blocchi
                    - (un vettore di inizializzazione se si usa la modalità CBC, un seme se si usa OFB o CFB),
                    - notare che deve essere presente da entrambi i lati (se cifro usando un IV/seed, devo decifrare usando lo stesso IV/seed)
                    - sono due perchè ogni lato cifra in maniera diversa
                - un terzo dato segreto per definire la chiave di sessione ks del Cifrario a blocchi.
        - Record
            - Terminato il protocollo Handshake inizia il protocollo di Record.

        - calcola/verifica le etichette di autenticazione (HMAC) e cifra/decifra i dati.
            - la cifratura è simmetrica con chiave di sessione scambiata durante la negoziazione.
            - per autenticazione e integrità si applica HMAC, con l'uso di un segreto condiviso durante la negoziazione.
                - si usa hmac dato che è più efficente rispetto alla firma, quest'ultimo è però ripudiabile
        - NB: TLS prima autentica e poi cifra
            - questo è più oneroso rispetto al contrario (ipsec) dato che prima di poter decidere se scartare o meno un pacchetto devo decifrare
        - Ogni pacchetto, secondo le decisioni prese da Handshake, può essere o non essere autenticato e cifrato.

26. Handshake

    - Possiamo distinguere quattro fasi:

        - Accordamento algoritmi e meccanismi di scambio delle chiavi in base a quelli supportati da entrambi.
        - Identificazione server presso il client (obbligatoria)
            - server fornisce qua il suo certificato al client
            - invia parametri per lo scambio delle chiavi (vedi tripla g, p, Y in DH) se necessario (se negoziato cifrario ibrido con RSA non serve)
            - firma per la POP assente, ma identificazione è implicita in quanto senza chiave privata non si riesce a concordare lo stesso segreto (DH) oppure a decifrare li messaggio con la chiave di sessione dentro
        - Identificazione client presso il server (facoltativa)
            - specchiata rispetto al server
            - alla fine di questa fase il master secret è stato scambiato
        - Controllo scambio master key concluso correttamente.
            - ci si assicura di aver concordato lo stesso segreto (scambiando un hash)
            - se qualcuno ha presentato un certificato che non è il suo, in questa fase ci si accorge del tentativo di impersonificazione dato che la chiave non combacia
            - se la verifica ha successo le chiavi si considerano installate e si procede con lo scambio di pacchetti cifrati ed autenticati (Protocollo Record)

    - **quand'è che il client si accorge che è presente la PoP del certificato del server (il server potrebbe avere mandato il certificato di un altro)?**

        - i casi sono 3.
            - ho negoziato cifrario ibrido
                - me ne accorgo in fase 4, vedendo se il server riesce a decifrare il segreto cifrato con la chiave pubblica presente nel certificato
            - ho negoziato ephemeral DH
                - me ne accorgo in fase 2, quando il server mi manda la tripla pubblica firmata e la verifica della firma non va a buon fine
                - come al solito, server potrebbe aver replicato una tripla firmata che ha intercettato, allora me ne accorgo in fase 4

        - ho negoziato fixed DH
            - qua non c'è una prova di possesso esplicita
            - me ne accorgo in fase 4 quando verifico di aver scambiato lo stesso segreto
            - se il server non è chi dice di essere **la fase 4 non si chiude** (per questo è importante questa conferma finale)

27. Ipsec

- Garantisce sempre confidenzialità e autenticazione alle applicazioni in maniera (stavolta totalmente) trasparente
    - stavolta però siamo a livello IP, autentichiamo e cifriamo la comunicazione non tra un client e un server (processi) ma tra macchine fisiche
- Ipsec è composto da
    - tre protocolli (AH, ESP e IKE) per svolgere altrettanti servizi sicuri
        - IKE non ci interessa (analogo ad handshake -> fa anche identificazione mutua)
    - tre strutture dati (SA, SAD, SPD) per configurarli come desiderato dagli utenti.
    - due modalità di incapsulamento (trasporto e tunnel)
- Con IPsec (grazie alle SA) abbiamo una **granularità maggiore dei servizi di sicurezza che possiamo impiegare** rispetto a SSL
    - possiamo ottenere solo riservatezza (ESP)
    - o solo autenticazione (AH)
    - o entrambi (ESP con autenticazione)
    - SSL mi da sempre tutto

28. Protolli IPSEC

- AH
    - Autentica il payload di un pacchetto e indirizzi ip
    - aggiungendo un header contenente un HMAC
- ESP
    - Cifra il payload di un pacchetto con un cifrario simmetrico e chiave di sessione scambiata con IKE
- ESP con autenticazione
    - cifra e autentica il payload di un pacchetto
    - ipsec prima cifra e poi fa hmac -> più corretto in fase di ricezione

29. Modalità di Incapsulamento

- Durante la definizione della SA è possibile scegliere tra due differenti modalità di incapuslamento dei pacchetti: il transport mode ed il tunnel mode. Queste rappresentano le modalità con cui il pacchetto IP originario viene incapsulato nel pacchetto IPsec che si va a costruire.

    - abbiamo quindi un pacchetto ip interno (con payload e header) ed un pacchetto ipsec esterno il cui payload è il pacchetto ip interno
    - Con entrambe le modalità, il servizio può poi essere o AH o ESP

- **Transport mode**

    - viene protetto (cifrato e/o autenticato) **solo il payload del pacchetto IP interno**

        - AH autentica anche indirizzi ip

    - **Tunnel mode**

        - **l'intero pacchetto IP interno viene cifrato e/o autenticato** e il pacchetto esterno ipsec viene incapsulato in un nuovo pacchetto ip con cui viene fatto il tunnelling (instradamento)
            - protezione totale del pacchetto anche degli header
            - AH adesso autentica indirizzi ip del pacchetto ipsec (non so bene a cosa serva)
        - gli indirizzi del pacchetto ip esterno sono tipicamente quelli di due gateway (ma vale anche la configurazione H2G se si vuole solo mandare roba) peer della comunicazione ipsec che hanno stabilito una SA e quindi sanno verificare l'autenticità e decifrare i pacchetti che arrivano
        - **Con questo metodo è possibile utilizzare Internet come supporto di una rete virtuale privata (VPN)**.
            - se i pacchetti hanno l'intestazione ESP nessuno può vederne indirizzi e contenuto;
            - è comunque in ogni caso impossibile modificarli o falsificarli senza che il ricevente se ne accorga (il tutto è autenticato).

    - **molto importante**: A seconda del protocollo IPSEC, e che modaliltà di incapsulamento uso, ottengo proprietà di sicurezza più o meno estesa

        - autentico/cifro solo payload e non header oppure tutto
        - bisogna scegliere la combinazione adeguata ai proprio scopi

30. Strutture dati ipsec

    - **SPD (Security Policy Database)**

        - entità che esamina tutto il traffico IP, sia in ingresso che in uscita, per decidere quali pacchetti debbano usufruire dei servizi IPSec.
            - una sorta di dispatcher con cui è possibile configurare cosa viene gestito da ipsec (quali porzioni del traffico), e come (politica di sicurezza per un determinato pacchetto)
        - in pratica una mappa con **chiave=selettore del traffico**, e **valore=SA**
        - Esempi:
            - tutto il traffico verso 192.169… deve essere protetto con ESP in modalità trasporto usando DES-CBC;
            - il traffico FTP verso 192… deve essere protetto con ESP in modalità tunnel usando 3DES-CBC;
            - ecc….

    - **SA (Security Association)**

        - accordo stipulato (con IKE) tra sorgente e destinazione che specifica
            - quali protocolli (AH o ESP)
            - quali algoritmi
            - e quali chiavi
            - dovranno utilizzare per proteggere ogni tipo (vedi selettori SPD) di traffico che intendono scambiarsi

        - una SA vale per una sola direzione nel flusso di dati, per proteggere entrambi i sensi bisogna stabilire due SA (una per lato)

    - **SAD (Security Association Database)**

        - Database in cui sorgente e destinazione mantengono tutte le SA che hanno negoziato
        - Tutte le SA attive su una connessione sono contenute nel SAD, in questa maniera non devo rinegoziare chiavi, segreti o quant'altro

    - **Invio e ricezione del traffico in IPSEC**

        - Invio di un pacchetto ip:

            - in base ai selettori in SPD si sceglie la politica di sicurezza da applicare al pacchetto, che specifica per il pacchetto IP
                - che modalità di incapuslamento adottare (trasporto o tunnel)
                - e che protocollo di sicurezza adottare (AH o ESP).
            - In base alla politica scelta, si controlla in SAD se esiste già una SA utilizzata in passato (istanziata con IKE) da poter riutilizzare o se si deve crearla.
            - A questo punto si applicano gli algoritmi di trasformazione per ottenere il pacchetto IPSec, e si invia quest'ultimo

        - Ricezione di un pacchetto ipsec:

            - il pacchetto ip interno è cifrato/autenticato.
            - si va a trovare la SA nel SAD per capire quali sono gli algoritmi da applicare per poter decifrare/ verificare
            - alla fine dell'applicazione degli algoritmi si va a vedere in SPD la politica di validità per capire se quello è un pacchetto valido.
            - Se lo è, si è riottenuto il pacchetto IP originario.

31. Antireplica in ipsec

    - si utilizza un finestra scorrevole di dimensione fissa e numeri di sequenza
        - la sorgente aggiunge un numero di sequenza ai pacchetti che manda per mettere in grado la destinazione di **scartare ogni pacchetto replicato da un intruso**.
        - se arriva qualcosa a sinistra delle finestra si assume che sia qualcosa che sia gia arrivato e quindi si scarta il pacchetto
        - se ricade dentro la finestra si marchia quel numero di sequenza come ricevuto (se il pacchetto è integro ed autentico)
            - successivi arrivi dello stesso seq num vengono scartati
        - se arriva qualcosa a destra della finestra si sposta la finestra e si marchia quel numero di sequenza come ricevuto

32. Autenticazione federata

    - Il numero di servizi è enorme, se l'autenticazione fosse centralizzata (ogni servizio autentica i proprio utenti):
        - numero di identità pari al numero di servizi 😦

        - ogni servizio dovrebbe preoccuparsi di salvaguardare le credenziali (poco casuali) dei propri utenti 😦
    - Per questi motivi si preferisce autenticazione federata: a group of trusted IdP share authentication responsibilities, allowing users to **access multiple services with a single identity**
        - l'applicazione non deve per forza registrare lei gli utenti e gestirne le credenziali in maniera sicura, ci pensa l'IdP
        - Individuals can use the same identity across multiple services
            - Users are not required to create different accounts
            - Users are not required to login multiple times (SSO)
    - drawbacks di autenticazione federata
        - Relies on trusted relationships between IdP and Services (RP) to authenticate and authorize users (vabè)
        - il mio IdP (es. google) vede tutti i servizi a cui accedo privacy lesa
        - Users and organizations may become dependent on a single IdP (e.g., Google, Microsoft) without an easy way to migrate (identity vendor lockin)

33. OAuth

    - A cosa serve?
        - a fornire autorizzazioni a servizi di terze parti per accedere (leggere/scrivere) a dati legati al tuo account presso un altro servizio
        - in passato se un servizio voleva accedere ai tuoi dati presso un altro servizio, dovevi fornirgli le tue credenziali
            - seconda coppia di credenziali che il servizio deve tenere sicura
            - autorizzazioni non granulari
        - con oauth deleghi un'applicazione a fare delle azioni per conto tuo **without giving them your credentials**
            - nel fornire le autorizzazioni l'utente si autentica, ma il RP non riceve mai le tue credenziali
            - inoltre, le autorizzazioni fornite possono venire controllate in maniera **granulare**
    - Entità in gioco
        - utente (resource owner)
        - authorization server (in generale distinto dal Resource server)
        - Relying party (Oauth Client)
    - Token in gioco
        - Authorization code
            - ottenuto una volta che l'utente ha fornito autorizzazioni al Client
            - è la prova che l'utente ha concesso delle autorizzazioni al Client
        - Access token
            - ottenuto una volta che il RP si è identificato presso AS mostrato il client secret e Authorization Code
            - token da presentare al Resource Server per ottenere accesso a dei dati
    - Authorization Server e Client si conoscono a priori e tra loro vi è già stabilità una relazione di fiducia
        - client ID e client secret sono dati già scambiati in precedenza ai flussi di autorizzazione

        - By keeping the client secret, secret, **this is how the Authorization Server can identify the Client** and make sure not to give an access token to an impostor

34. OIDC

- OAuth 2.0 is designed only for authorization
    - granting access to data from one application to another.
    - per ottenere anche autenticazione si potrebbe pensare usare sempre oauth predisponendo un endpoint del tipo /userinfo con cui recuperare l'identità dell'utente... meglio usare OIDC
- OIDC is a thin layer that sits on top of OAuth that adds identity information about the person who is logged in.
    - permette sia la gestione delle autorizzazioni che delle autenticazioni
    - autorizzazioni vengono concesse in maniera analoga ad OAuth (thin layer that sits on top)
    - se serve autenticazione, invece di accedere a dati generici il Client accede all'identità digitale
- The OpenID Connect flow **looks the same as OAuth**.
    - le entità in gioco sono le stesse
    - ci sono sempre authorization code ed access token
- The only differences are
    - in the initial request, a **specific scope of openid** is used (Client vuole accedere all'identità digitale)
    - and in the final exchange the Client receives both an Access Token and an **ID Token**
    - l'id token è un jwt (quindi firmato) contenente l'identità digitale dell'utente
- Con le informazioni presenti nell'id token (firmate dall'IdP) il Client può registrare l'utente (se è la prima volta) e il resource owner può finalmente accedere al servizio del RP
    - è qui che avviene l'autenticazione! (identificazione dell'utente)
    - notare che come prima non è stata condivisa la pwd dell'utente con il RP
- Riassumendo:
    - l’access token autorizza l’accesso alle risorse
    - mentre l’ID token autentica l’utente e trasmette in modo sicuro la sua identità al Client

35. Blockchain

- blockchain come tecnologia per risolvere trust problem

    - pensa a pagamenti online
    - acquirente e venditore in generale non si fidano l'uno dell'altro
    - si fa affidamento ad una terza parte fidata centrale (banca) che mi conferma che il pagante ha i soldi necessari per completare la transazione
        - la terza parte deve essere online altrimenti non riesco a pagare
        - la terza parte sa cosa compro/vendo
    - con blockchain
        - possiamo validare le transazioni, e più in generale ottenere consenso in un sistema decentralizzato, in un contesto non fidato
        - **eliminando la terza parte fidata centrale**, mantenendo così totale anonimità

- Consenso
    - Distributed processes that have to agree on a single value (e.g., new status of the system).
    - **blockchain è una soluzione al problema del consenso nel contesto bizantino (nodi che possono mentire) di internet**
    - vogliamo che nodi in un sistema decentralizzato, che non si fidano tra di loro, e con la presenza di nodi malevoli, riescano a mettersi daccordo sullo stato delle transazioni senza la terza parte fidata che detta legge
- struttura blockchain
    - A blockchain is an **append-only list of (effectively) IMMUTABLE records (ledger)**, called **blocks (che sono quindi dati di qualsisi natura)**, that are linked together using hash-pointers.
        - è quindi una semplice **struttura dati**
    - Blockchains are typically **managed by a decentralized peer-to-peer network** for use as a publicly distributed ledger (DB distribuito).
        - **distributed on many nodes of a P2P network (everyone has a copy of the ledger)**
        - e **decentralized (nobody owns it)**
    - The block consists of two main parts:
        - Header: contains the hash of the previous block, the root hash of the Merkle tree (data hash) and other information
        - Merkle tree (data):
            - how the data is structured inside a block
            - the leaves of the tree are the transactions (notare il plurale) contained in the block.
- Peers who generate new blocks are called miners.
    - **A miner receives transactions from clients**.
    - After **validating them**, it can start the process of generating a new block.
    - To discourage byzantine miners and DoS attacks, **Bitcoin requires miners to solve a "puzzle" (mining) in order to generate a new block**
        - This is the so-called **Proof of Work**.
        - PoW = biglietto d'ingresso costoso per proporre un blocco.
            - I nodi scartano automaticamente i blocchi senza PoW valida → spam inutile.
        - Solving the puzzle in a short time requires a **large amount of computational power**.
        - Also, if you solve the puzzle, **you get a reward in Bitcoin**.
            - qusto incentiva i miner a produrre nuovi blocchi il che risolve le fork
    - **Perché serve la Proof of Work (PoW) per aggiungere blocchi?**
        - chiunque potrebbe dire: "Ho creato il nuovo blocco, accettatelo!"
            - **La PoW introduce un costo computazionale per proporre un blocco**.
            - per essere preso sul serio, un blocco deve dimostrare che il suo creatore ha consumato risorse reali (energia, tempo, hardware).
        - **Come scoraggia i nodi bizantini?**

        - Un nodo bizantino potrebbe voler immettere blocchi invalidi o manipolati.
        - Ma per ogni blocco deve risolvere il puzzle → costa soldi ed energia.
        - Se la maggioranza (>50% della potenza di calcolo) è onesta, **la catena valida crescerà più velocemente di quella malevola** e per nakamoto consensus gli altri nodi seguiranno la catena onesta.
            - è difficile che un attaccante abbia > 50% della potenza computazionale della rete

- **Nakamoto consensus**

    - the algorithm used in Bitcoin network to achieve **trustless consensus** riguardo al prossimo blocco contenente transazioni da aggiungere.
        - l'unica assunzione è che il 50% +1 dei nodi della rete sia non malevolo (altrimenti la maggioranza si mette daccordo su quello che vuole)
    - Two possible scenarios:
        - The miner solves the puzzle, adds the block to its chain and sends it to all other nodes (broadcast)
        - The miner before solving the puzzle receives a block from another node. In this case, it stops searching for the solution and adds the block to its chain.
    - **NB**: Actually, a new block may arrive at any moment (asynchrony). **What happens if a miner receives a block when it has already added one to its chain?**
        - **The miner keeps both blocks** as if they are both valid, resulting in a **fork**.
        - The rule is that **the longest chain is the one that individual nodes accept as a valid** version of the blockchain.
            - **NB**: questa è la regola di consenso
        - blocks inside the other shorter paths of the fork are called “orphans” and transactions in them are considered invalid.

- **La blockchain è tamper-resistant**

    - in generale se modifico un blocco devo anche modificare tutti i blocchi successivi per aggiustare gli hashpointer
    - questo lo posso anche fare sulla mia copia locale ma poi devo raggiungere il consenso del resto della rete
        - devo fare una o più PoW per aggiungere i miei blocchi modificati (che produrranno una fork)
        - intanto il resto della rete continua ad aggiungere blocchi il che peggiora la mia situazione
        - anche se riesco a fare le PoW, genero una fork e la mia catena deve diventare quella più lunga (vedi nakamoto consensus)
    - per questo motivo i dati nella blockchain sono **effectively immutable**
        - non basta che io modifica la mia copia locale della blockchain!
        - **i fatti veri sono dettati dalla maggiornaza dei nodi**

- **Double spending** come esempio di tamper-resistance della blockchain

    - Double-spending is the risk that a cryptocurrency can be used twice or more.
    - What an attacker might do is send two different transactions to two different miners.

            - i miner aggiungono alla loro catena personale le due transazioni
            - quindi sembra che gli stessi soldi siano stati spesi due volte ...
        - But no problem! **Eventually one of the two blocks will be considered invalid**
            - abbiamo che i due miner aggiungono ognuno il loro nodo -> producono una fork nella catena
            - dipendentemente da quale dei due percorsi diventa il più lungo, una delle due transazioni diventa invalida
            - The rule per avere certezza che una transazione sia valida, is to wait for the block in which the transaction is present to be “buried” by other blocks (about 6).
                - dopo 6 è difficile che catene concorrenti superino la propria
            - nakamoto consensus previene il double spending invalidando uno dei due blocchi

    - Double spending con 51%

        - It is possible to exploit the fork mechanism to double-spend, if one entity owns 51% of the computational power of the entire network.
        - Se l’attaccante controlla più della metà della potenza di calcolo totale della rete, **questo significa che può generare blocchi più velocemente di tutti gli altri miner messi insieme.**
        - L'attaccante può comprare qualcosa e aspettare che il venditore invii il bene (magari egli ha anche aspettato 6 blocchi per sicurezza)
        - Una volta ricevuto il bene, l'attaccante può attivare i suoi miner (> 50%) e cominciare una catena alternativa in cui la transazione verso il venditore non è mai avvenuta
        - Siccome l'attaccante possiede la maggior parte della potenza computazionale della reta, prima o poi la sua catena diventerà quella più lunga e gli altri nodi saranno costretti ad accetarla come valida
        - Il venditore perde sia i beni consegnati che i bitcoin.

    - Considerazioni su nakamoto consensus

        - Agreement:
            - probabilistic.
            - If we stop at any time, we can always have forks.
        - Termination:
            - to have deterministic agreement we cannot stop!
        - The main drawback of Bitcoin (Nakamato Consensus really) is that **it can never stop in order to work**.
            - beh potrei fermarmi e scartare tutti i blocchi presenti in una fork...
            - però in effetti questa lista potrebbe essere arbitrariamente lunga
        - Inoltre, con nakamoto consensus:
            - risolvo fork e double spending
            - ... ma devo aspettare un'ora per comprare qualcosa

36. Smart contract

    - una blockchain è una struttura dati distribuita in un sistema decentralizzato
    - uno smart contract è un programma salvato su blockchain ed eseguito dai peer della rete
        - Stored on the blockchain = Each node in the network has a copy of it.

- **Come viene eseguito?**
  - Each contract has an address that uniquely identifies it, functions that can be invoked, and an internal state (e.g. the value of a variable at a given moment in time).
  - To execute a function it is necessary **to perform a transaction to the address of the SC**, specifying the function name and arguments.
  - Each node independently executes the operations defined by the contract on its local copy.
  - **All of them have to get the same result to reach the consensus**. Otherwise the transaction is not valid and is rejected.
- **Cosa ottengo**
  - esecuzione distribuita, fault tolerant
    - non ho un server centralizzato su cui esegue il codice
    - tutti lo eseguono
  - dati immutabili
  - tracing completo
    - ho l'intera storia dell'esecuzione del programma
  - disaster recovery
    - in eventualità di crash lo stato precedente è riproducibile in quanto salvato su blockchain sotto forma di transazioni