# Integrità

Gregorio D'Agostino

16 Aprile 2021

Riepilogo di quanto svolto sulla Confidenzialità

Integrità

Controllo ciclico di ridondanza (Cyclic redundancy Check)

# Attività svolte

- Steganografia e crittografia

# Attività svolte

- Steganografia e crittografia
- Crittografia antica: scitala, Cesare, Bellasio, Vigenere

# Attività svolte

- Steganografia e crittografia
- Crittografia antica: scitala, Cesare, Bellasio, Vigenere
- Metodi base: trasposizione e sostituzione

# Attività svolte

- Steganografia e crittografia
- Crittografia antica: scitala, Cesare, Bellasio, Vigenere
- Metodi base: trasposizione e sostituzione
- Teoria dei numeri: Aritmetica, Sistemi Diofantei, Anelli, Fermat, Eulero, Gauss, Carmichael. Criteri di primalità e pseudo-primalità.

# Attività svolte

- Steganografia e crittografia
- Crittografia antica: scitala, Cesare, Bellasio, Vigenere
- Metodi base: trasposizione e sostituzione
- Teoria dei numeri: Aritmetica, Sistemi Diofantei, Anelli, Fermat, Eulero, Gauss, Carmichael. Criteri di primalità e pseudo-primalità.
- Scrittografia moderna asimmetrica. RSA.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.
- Garantire la confidenzialià cioè l'esclusività dell'accesso all'informazione da parte dei soggetti autorizzati.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.
- Garantire la confidenzialià cioè l'esclusività dell'accesso all'informazione da parte dei soggetti autorizzati.
- Tutte le combinazione di queste potenzialità.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.
- Garantire la confidenzialià cioè l'esclusività dell'accesso all'informazione da parte dei soggetti autorizzati.
- Tutte le combinazione di queste potenzialità.
- La crittografia che si realizza di chiama a "chiave asimmetrica" perché la cifratura non coincide e non richiede le stesse conoscenze della decifratura.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.
- Garantire la confidenzialià cioè l'esclusività dell'accesso all'informazione da parte dei soggetti autorizzati.
- Tutte le combinazione di queste potenzialità.
- La crittografia che si realizza di chiama a "chiave asimmetrica" perché la cifratura non coincide e non richiede le stesse conoscenze della decifratura.
- L'algoritmo non richiede alcun intermediario e l'informazione può viaggiare su un canale non protetto visibile a chiunque.

# Riepilogo delle applicazioni dell'algoritmo RSA

- Cifrare un testo o in generale una seguenza di bit come ad esempio un codice.
- Garantire l'origine (e dunque in parte la "non ripudiazione") di un testo.
- Garantire la confidenzialià cioè l'esclusività dell'accesso all'informazione da parte dei soggetti autorizzati.
- Tutte le combinazione di queste potenzialità.
- La crittografia che si realizza di chiama a "chiave asimmetrica" perché la cifratura non coincide e non richiede le stesse conoscenze della decifratura.
- L'algoritmo non richiede alcun intermediario e l'informazione può viaggiare su un canale non protetto visibile a chiunque.
- L'applicazione più diffusa è la cosiddetta "end to end security" adesso attiva su tutti i dispositivi di comunicazione mobile.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.
- Trovare il periodo di un elemento o il suo inverso.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.
- Trovare il periodo di un elemento o il suo inverso.
- Risolvere un sistema diofanteo. Ad esempio un esercizio equivalente è trovare una coppia $k$ e $\alpha$ di chiavi in RSA conoscendo la fattorizzazione del modulo.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.
- Trovare il periodo di un elemento o il suo inverso.
- Risolvere un sistema diofanteo. Ad esempio un esercizio equivalente è trovare una coppia $k$ e $\alpha$ di chiavi in RSA conoscendo la fattorizzazione del modulo.
- Trovare un generatore o tutti i generatori.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.
- Trovare il periodo di un elemento o il suo inverso.
- Risolvere un sistema diofanteo. Ad esempio un esercizio equivalente è trovare una coppia $k$ e $\alpha$ di chiavi in RSA conoscendo la fattorizzazione del modulo.
- Trovare un generatore o tutti i generatori.
- Trovare il massimo periodo in un gruppo moltiplicativo.

# Esercizi proposti

Non serve conoscere le dimostrazioni dei teoremi, ma bisogna saperli applicare.

- Scrivere un numero in formato binario o ottale o esadecimale.
- Dato un $n$ decomporlo in fattori primi con l'algoritmo di Fermat.
- Verificare la pseudoprimalità di un numero.
- Trovare il periodo di un elemento o il suo inverso.
- Risolvere un sistema diofanteo. Ad esempio un esercizio equivalente è trovare una coppia $k$ e $\alpha$ di chiavi in RSA conoscendo la fattorizzazione del modulo.
- Trovare un generatore o tutti i generatori.
- Trovare il massimo periodo in un gruppo moltiplicativo.
- Dire se un gruppo è ciclico. Trovare la funziona di Carlmichael $\lambda$ e trovare un elemento di massima periodicità.

# Integrità

- Abbiamo trascorso un periodo per studiare le basi della teoria dei numeri. Una prima finalizzazione è stata verso la tutela della confidenzialità tramite l'algoritmo di "RSA".

# Integrità

- Abbiamo trascorso un periodo per studiare le basi della teoria dei numeri. Una prima finalizzazione è stata verso la tutela della confidenzialità tramite l'algoritmo di "RSA".
- I tre cardini della Sicurezza informatica dei dati sono la confidenzialità, la disponibilità e l'integrità. Cercheremo di alternarli. Oggi vedremo alcuni punti sull'integrità.

# Integrità

- Abbiamo trascorso un periodo per studiare le basi della teoria dei numeri. Una prima finalizzazione è stata verso la tutela della confidenzialità tramite l'algoritmo di "RSA".
- I tre cardini della Sicurezza informatica dei dati sono la confidenzialità, la disponibilità e l'integrità. Cercheremo di alternarli. Oggi vedremo alcuni punti sull'integrità.
- Il primo punto da osservare è che qualunque sistema informatico presenta delle imperfezioni e può dunque dare luogo ad errori. Come sempre oltre alla sorgente casuale di errori bisogna pensare all'azione volontaria di un soggetto ostile che può deliberatamente modificare le sequenza di dati che custodiamo, diffondiamo o eleboriamo.

# Duplicazione

- La verifica esatta dell'integrità di una sequenza informativa presuppone la disponibilità della sequenza originale.

# Duplicazione

- La verifica esatta dell'integrità di una sequenza informativa presuppone la disponibilità della sequenza originale.
- La ridondanza integrale dei dati consiste nelle operazioni di duplicazione (o moltiplicazione) dei dati su supporti locali o remoti. La scrittura può avvenire con speciali dispositivi in tempo reale all'immagazzinamento dei dati o in una fase successiva. I principali meccanismi sono il RAID e il backup che vedremo in seguito. La ridondanza integrale è la soluzione al problema della disponibilità dei dati nel paradigma della resilienza.

# Duplicazione

- La verifica esatta dell'integrità di una sequenza informativa presuppone la disponibilità della sequenza originale.
- La ridondanza integrale dei dati consiste nelle operazioni di duplicazione (o moltiplicazione) dei dati su supporti locali o remoti. La scrittura può avvenire con speciali dispositivi in tempo reale all'immagazzinamento dei dati o in una fase successiva. I principali meccanismi sono il RAID e il backup che vedremo in seguito. La ridondanza integrale è la soluzione al problema della disponibilità dei dati nel paradigma della resilienza.
- Ma anche la sequenza originale è suscettibile di deterioramenti o alterazioni volontarie ostili. Inoltre la duplicazione dei dati, il loro trasporto ed il loro confronto consumano risorse di memoria e di tempo. Per tali ragioni si utilizzano criteri parziali per la verifica dell'integrità. Per criteri parziali si intende condizioni necessarie per l'integrità dei dati, ma non sufficienti.

# Ridondanza

- In ogni caso anche per fornire tali parziali garanzie di integrità è necessario allocare risorse suppletive di memoria e tempo di elaborazione.

# Ridondanza

- In ogni caso anche per fornire tali parziali garanzie di integrità è necessario allocare risorse suppletive di memoria e tempo di elaborazione.
- La ridondanza è, per definizione, una allocazione di risorse di memoria e tempo di elaborazione che eccede le risorse minimali alla elaborazione, trasmissione o immagazzinamento dei dati. In alcuni casi la ridondanza è puramente computazionale, in altri puramente spaziale.

# Ridondanza

- In ogni caso anche per fornire tali parziali garanzie di integrità è necessario allocare risorse suppletive di memoria e tempo di elaborazione.
- La ridondanza è, per definizione, una allocazione di risorse di memoria e tempo di elaborazione che eccede le risorse minimali alla elaborazione, trasmissione o immagazzinamento dei dati. In alcuni casi la ridondanza è puramente computazionale, in altri puramente spaziale.
- Il concetto di ridondanza può essere applicato anche alle risorse di calcolo e di memoria allocate ad una attività. Esempio: molti strumenti diagnostici sono interfacciati a camputer tramite connesioni seriali, ethernet o altro. Se lo strumento è molto costoso si possono tenere computer o schede dati di riserva. Altro esempio: i dati di un computer possono essere copiati in due dischi rigidi. Un dispositivo può essere connesso ad una rete tramite diverse interfaccia, etc.

# Funzioni di verifica "hash functions"

- Data una sequenza di lunghezza variabile, una funzione di verifica ("hash function") è una applicazione che fornisce una sequenza di lunghezza prefissata univocamente determinata dalla sequenza iniziale.

# Funzioni di verifica "hash functions"

- Data una sequenza di lunghezza variabile, una funzione di verifica ("hash function") è una applicazione che fornisce una sequenza di lunghezza prefissata univocamente determinata dalla sequenza iniziale.
- Il termine "hash" letteralmente significa "tagliare a dadini" o sminuzzare, tritare o macinare. In pratica si riduce la sequenza iniziale in poltiglia per poi creare una polpetta, un goulash, uno spezzatino etc.

# Funzioni di verifica "hash functions"

- Data una sequenza di lunghezza variabile, una funzione di verifica ("hash function") è una applicazione che fornisce una sequenza di lunghezza prefissata univocamente determinata dalla sequenza iniziale.
- Il termine "hash" letteralmente significa "tagliare a dadini" o sminuzzare, tritare o macinare. In pratica si riduce la sequenza iniziale in poltiglia per poi creare una polpetta, un goulash, uno spezzatino etc.
- In informatica si fa una cosa simile si prendono elementi di una sequenza e si costruisce una sequenza di lunghezza prefissata. Sia $S$ lo spazio delle sequenze di lunghezza arbitraria e $H$ lo spazio delle sequenze di lunghezza fissata $k$. Se per semplicità fissiamo un limite $n$ alle sequenze di $S$: in questo caso $|S| = 2^n$, mentre $|H| = 2^k$ . In cui $k$ è la lunghezza della sequenza di verifica (ovvero in inglese "hash values, hash codes, hash sums, o anche solamente hashes").

# Funzioni di verifica - proprietà matematiche

- Definizione formale

$$h : S \rightarrow H;$$

cioè:

$$\forall s \in S : \exists h(s) \in H : s- > h(s).$$

# Funzioni di verifica - proprietà matematiche

- Definizione formale

$$h : S \rightarrow H;$$

cioè:

$$\forall s \in S : \exists h(s) \in H : s- > h(s).$$

- L'output dell'algoritmo su una sequenza è detto anche il "digest".

# Funzioni di verifica - proprietà matematiche

- Definizione formale

$$h : S \rightarrow H;$$

cioè:

$$\forall s \in S : \exists h(s) \in H : s- > h(s).$$

- L'output dell'algoritmo su una sequenza è detto anche il "digest".
- La prima proprietà è la suriettività: tutte le sequenze di lunghezza $k$ devono essere ottenibili:

$$\forall hs \in H : \exists s \in S : hs = h(s).$$

Ovviamente se la sequenza originale è più lunga della hash ($n > k$) esisteranno diverse sequenze con lo stesso hash.

# Funzioni di verifica - proprietà matematiche

- Definizione formale

$$h : S \rightarrow H;$$

cioè:

$$\forall s \in S : \exists h(s) \in H : s- > h(s).$$

- L'output dell'algoritmo su una sequenza è detto anche il "digest".
- La prima proprietà è la suriettività: tutte le sequenze di lunghezza $k$ devono essere ottenibili:

$$\forall hs \in H : \exists s \in S : hs = h(s).$$

Ovviamente se la sequenza originale è più lunga della hash $(n > k)$ esisteranno diverse sequenze con lo stesso hash.

- Essendo solitamente lo spazio $H$ molto più piccolo di $S$ l'applicazione $h$ non è invertibile.

# Funzioni di verifica - proprietà matematiche

- Definizione formale

$$h : S \rightarrow H;$$

cioè:

$$\forall s \in S : \exists h(s) \in H : s- > h(s).$$

- L'output dell'algoritmo su una sequenza è detto anche il "digest".
- La prima proprietà è la suriettività: tutte le sequenze di lunghezza $k$ devono essere ottenibili:

$$\forall hs \in H : \exists s \in S : hs = h(s).$$

Ovviamente se la sequenza originale è più lunga della hash $(n > k)$ esisteranno diverse sequenze con lo stesso hash.

- Essendo solitamente lo spazio $H$ molto più piccolo di $S$ l'applicazione $h$ non è invertibile.
- Per essere resistenti rispetto ad azioni deliberate ostili si aggiungono altri requisiti.

# Robustezza delle Funzioni di verifica "hash function"

- "Resistenza alla ricerca di una preimmagine". Deve essere computazionalmente difficile trovare una sequenza $s$ che abbia una hash function prefissata.

# Robustezza delle Funzioni di verifica "hash function"

- "Resistenza alla ricerca di una preimmagine". Deve essere computazionalmente difficile trovare una sequenza $s$ che abbia una hash function prefissata.
- "Resistenza alla ricerca di un'altra preimmagine (detta seconda preimmagine)". Data una sequenza, deve essere computazionalmente difficile trovare un'altra sequenza $s'$ che abbia la stessa hash function.
  In questo caso si dice che l'algoritmo di verifica gode della "Sicurezza Debole".

# Robustezza delle Funzioni di verifica "hash function"

- "Resistenza alla ricerca di una preimmagine". Deve essere computazionalmente difficile trovare una sequenza $s$ che abbia una hash function prefissata.
- "Resistenza alla ricerca di un'altra preimmagine (detta seconda preimmagine)". Data una sequenza, deve essere computazionalmente difficile trovare un'altra sequenza $s'$ che abbia la stessa hash function.
  In questo caso si dice che l'algoritmo di verifica gode della "Sicurezza Debole".
- "Resistenza alle collisioni". In generale deve essere computazionalmente difficile trovare due sequenze con la stesso hash, indipendentemente dal suo valore.
  In questo caso si dice che l'algoritmo di verifica gode della "Sicurezza Forte".

# Applicazione alla firma digitale

- Supponiamo di avere un testo lungo che non vogliamo criptare integralmente, ma vogliamo garantirne la paternità. Possiamo procedere come segue:

# Applicazione alla firma digitale

- Supponiamo di avere un testo lungo che non vogliamo criptare integralmente, ma vogliamo garantirne la paternità. Possiamo procedere come segue:
- Rendiamo disponibile il testo in chiaro. Calcoliamo la sua hash tramite una funzione di hash anch'essa pubblica. Corrediamo il documento in chiaro con la sua hash (cioè del suo "digest") criptata con la nostra chiave privata. Quest'ultima parte si chiama la firma e l'operazione si chiama mettere la firma "digitale".

# Applicazione alla firma digitale

- Supponiamo di avere un testo lungo che non vogliamo criptare integralmente, ma vogliamo garantirne la paternità. Possiamo procedere come segue:
- Rendiamo disponibile il testo in chiaro. Calcoliamo la sua hash tramite una funzione di hash anch'essa pubblica. Corrediamo il documento in chiaro con la sua hash (cioè del suo "digest") criptata con la nostra chiave privata. Quest'ultima parte si chiama la firma e l'operazione si chiama mettere la firma "digitale".
- Si noti che diversamente dalle firme su carta la "firma digitale" cambia da documento a documento.

# Applicazione alla firma digitale

- Supponiamo di avere un testo lungo che non vogliamo criptare integralmente, ma vogliamo garantirne la paternità. Possiamo procedere come segue:
- Rendiamo disponibile il testo in chiaro. Calcoliamo la sua hash tramite una funzione di hash anch'essa pubblica. Corrediamo il documento in chiaro con la sua hash (cioè del suo "digest") criptata con la nostra chiave privata. Quest'ultima parte si chiama la firma e l'operazione si chiama mettere la firma "digitale".
- Si noti che diversamente dalle firme su carta la "firma digitale" cambia da documento a documento.
- Chiunque può decifrare la nostra firma con la nostra chiave pubblica. Poi può calcolare il digest del documento in chiaro con la funzione di verifica nota a tutti e verificare che coincidono.

# Applicazione alla firma digitale

- Supponiamo di avere un testo lungo che non vogliamo criptare integralmente, ma vogliamo garantirne la paternità. Possiamo procedere come segue:
- Rendiamo disponibile il testo in chiaro. Calcoliamo la sua hash tramite una funzione di hash anch'essa pubblica. Corrediamo il documento in chiaro con la sua hash (cioè del suo "digest") criptata con la nostra chiave privata. Quest'ultima parte si chiama la firma e l'operazione si chiama mettere la firma "digitale".
- Si noti che diversamente dalle firme su carta la "firma digitale" cambia da documento a documento.
- Chiunque può decifrare la nostra firma con la nostra chiave pubblica. Poi può calcolare il digest del documento in chiaro con la funzione di verifica nota a tutti e verificare che coincidono.
- Questo metodo è meno sicuro e più semplice che criptare tutto il documento iniziale con la mostra chiave privata, ma la lettura è molto più semplice.

# Verifica di errori

- Le hash function consentono di verificare eventuali errori molto velocemente. Ad esempio aggiungere un "non" in un testo è semplicissimo, mentre è molto difficile rileggendo rendersi conto della modifica. Ma la hash function cambia radicalmente. Il limite delle funzioni di verifica è che non consentono di ripristinare la forma originale, ma solo evidenziare la difformità.

# Verifica di errori

- Le hash function consentono di verificare eventuali errori molto velocemente. Ad esempio aggiungere un "non" in un testo è semplicissimo, mentre è molto difficile rileggendo rendersi conto della modifica. Ma la hash function cambia radicalmente. Il limite delle funzioni di verifica è che non consentono di ripristinare la forma originale, ma solo evidenziare la difformità.
- L'applicazione più comune è nella trasmissione dei dati. Normalmente si applica una ridondanza ad ogni blocco di dati trasmessi per fare una verifica della loro integrità. In questi casi si parla di "controllo ridondante" o "controllo per ridondanza".

# Verifica di errori

- Le hash function consentono di verificare eventuali errori molto velocemente. Ad esempio aggiungere un "non" in un testo è semplicissimo, mentre è molto difficile rileggendo rendersi conto della modifica. Ma la hash function cambia radicalmente. Il limite delle funzioni di verifica è che non consentono di ripristinare la forma originale, ma solo evidenziare la difformità.
- L'applicazione più comune è nella trasmissione dei dati. Normalmente si applica una ridondanza ad ogni blocco di dati trasmessi per fare una verifica della loro integrità. In questi casi si parla di "controllo ridondante" o "controllo per ridondanza".
- Applicando tali controlli la banda effettiva di trasmissione si riduce. Più è specifico il controllo e più si riduce la banda effettiva.

# Verifica di errori - Bit di parità

- Supponiamo di volere "essere sicuri" (leggasi verificare con una hash) che un numero a 64 bit venga trasferito correttamente (ad esempio dal processore al bus pci).

# Verifica di errori - Bit di parità

- Supponiamo di volere "essere sicuri" (leggasi verificare con una hash) che un numero a 64 bit venga trasferito correttamente (ad esempio dal processore al bus pci).
- Se ci accontentiamo di trasferire dati da 63 bit, possiamo mandare i 63 bit e come primo bit lo "xor" (cioè) l'or disgiuntivo dei rimanenti 63 bit. Siano $b_1, b_2, \ldots b_n$ i bit della sequenza che si intende trasmettere. Il bit di controllo aggiuntivo sarà dato da:

$$b_c \stackrel{def}{=} b_1 \oplus b_2 \oplus b_3 \cdots \oplus b_m;$$

in cui il simbolo $\oplus$ indica il connettivo logico "aut" cioè "o esclusivo". Per la proprietà associativa l'ordine di esecuzione non conta.

# Verifica di errori - Bit di parità

- Supponiamo di volere "essere sicuri" (leggasi verificare con una hash) che un numero a 64 bit venga trasferito correttamente (ad esempio dal processore al bus pci).
- Se ci accontentiamo di trasferire dati da 63 bit, possiamo mandare i 63 bit e come primo bit lo "xor" (cioè) l'or disgiuntivo dei rimanenti 63 bit. Siano $b_1, b_2, \ldots b_n$ i bit della sequenza che si intende trasmettere. Il bit di controllo aggiuntivo sarà dato da:

$$b_c \stackrel{def}{=} b_1 \oplus b_2 \oplus b_3 \cdots \oplus b_m;$$

  in cui il simbolo $\oplus$ indica il connettivo logico "aut" cioè "o esclusivo". Per la proprietà associativa l'ordine di esecuzione non conta.
- La grandezza su definita è anche detta "parità". La sequenza inviata sarà dunque:

$$b_1, b_2, b_3 \cdots, b_m, b_c.$$

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:
  - leggerà i primi $n = 63$ bit;

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:
  - leggerà i primi $n = 63$ bit;
  - calcolerà la loro parità;

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:
  - leggerà i primi $n = 63$ bit;
  - calcolerà la loro parità;
  - e la confronterà con l'ultimo bit ricevuto.

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:
  - leggerà i primi $n = 63$ bit;
  - calcolerà la loro parità;
  - e la confronterà con l'ultimo bit ricevuto.
- Se la parità ricevuta e quella calcolata coincidono, la sequenza è accettata, altrimenti bisognerà riottenerla.

# Verifica di errori - Bit di parità

- Il ricevente eseguirà le operazioni in ordine inverso:
  - leggerà i primi $n = 63$ bit;
  - calcolerà la loro parità;
  - e la confronterà con l'ultimo bit ricevuto.
- Se la parità ricevuta e quella calcolata coincidono, la sequenza è accettata, altrimenti bisognerà riottenerla.
- Ovviamente basta cambiare due bit contemporaneamente ed i due errori si cancellano. La verifica positiva non garantisce l'integrità della sequenza.

# Le funzioni di verifica semplici (a blocchi)

- Fissata la lunghezza $n$ della hash, ad esempio 128bit, ma può essere qualsiasi valore.

# Le funzioni di verifica semplici (a blocchi)

- Fissata la lunghezza $n$ della hash, ad esempio 128bit, ma può essere qualsiasi valore.
- Si suddivide la sequenza da verificare in maniera continua in $m$ blocchi da $n$ bit. Tipicamente la lunghezza totale della sequenza è 1 kB (1024 byte=8192 bit) quindi ad esempio $m = 64$ e $n = 128$.

# Le funzioni di verifica semplici (a blocchi)

- Fissata la lunghezza $n$ della hash, ad esempio 128bit, ma può essere qualsiasi valore.
- Si suddivide la sequenza da verificare in maniera continua in $m$ blocchi da $n$ bit. Tipicamente la lunghezza totale della sequenza è 1 kB (1024 byte=8192 bit) quindi ad esempio $m = 64$ e $n = 128$.
- Si decompone il flusso di dati in gruppi da 63 blocchi da 128bit: $b_{1,1}, b_{1,2}, \ldots, b_{1,n}, b_{2,1}, b_{2,2}, \ldots, b_{2,n}, \ldots, b_{m,n}, b_{m,1}, b_{m,2}, \ldots, b_{m,n}$

# Le funzioni di verifica semplici (a blocchi)

- Fissata la lunghezza $n$ della hash, ad esempio 128bit, ma può essere qualsiasi valore.
- Si suddivide la sequenza da verificare in maniera continua in $m$ blocchi da $n$ bit. Tipicamente la lunghezza totale della sequenza è 1 kB (1024 byte=8192 bit) quindi ad esempio $m = 64$ e $n = 128$.
- Si decompone il flusso di dati in gruppi da 63 blocchi da 128bit: $b_{1,1}, b_{1,2}, \ldots, b_{1,n}, b_{2,1}, b_{2,2}, \ldots, b_{2,n}, \ldots, b_{m,n}, b_{m,1}, b_{m,2}, \ldots, b_{m,n}$
- Si calcolano le parità (xor sequenziali) di 63 messaggi:

$$b_{ci} \stackrel{def}{=} b_{1i} \oplus b_{2i} \oplus b_{3i} \cdots \oplus b_{63i};$$

ottenendo così un altro blocco da 128bit.

# Le funzioni di verifica semplici (a blocchi)

- Fissata la lunghezza $n$ della hash, ad esempio 128bit, ma può essere qualsiasi valore.
- Si suddivide la sequenza da verificare in maniera continua in $m$ blocchi da $n$ bit. Tipicamente la lunghezza totale della sequenza è 1 kB (1024 byte=8192 bit) quindi ad esempio $m = 64$ e $n = 128$.
- Si decompone il flusso di dati in gruppi da 63 blocchi da 128bit: $b_{1,1}, b_{1,2}, \ldots, b_{1,n}, b_{2,1}, b_{2,2}, \ldots, b_{2,n}, \ldots, b_{m,n}, b_{m,1}, b_{m,2}, \ldots, b_{m,n}$
- Si calcolano le parità (xor sequenziali) di 63 messaggi:

$$b_{ci} \stackrel{def}{=} b_{1i} \oplus b_{2i} \oplus b_{3i} \cdots \oplus b_{63i};$$

  ottenendo così un altro blocco da 128bit.
- Si inviano tutti i 64 blocchi da 128bit che formano un messaggio da 1kB. In questo caso la ridondanza è $1/63$ e la banda effettiva cambia di poco. In generale è $1/(m-1)$.

# Le funzioni di verifica semplici (a blocchi) - miglioramenti

- Come abbiamo visto i codici ascii delle lettere sono piccoli (da 65 a 122) e quindi non arrivano a coprire tutti i gruppi da 8 bit (il primo bit sarà sempre zero). Quindi anche i blocchi da 128 bit all'inizio avranno sempre 0 se la sequenza codifica lettere.

# Le funzioni di verifica semplici (a blocchi) - miglioramenti

- Come abbiamo visto i codici ascii delle lettere sono piccoli (da 65 a 122) e quindi non arrivano a coprire tutti i gruppi da 8 bit (il primo bit sarà sempre zero). Quindi anche i blocchi da 128 bit all'inizio avranno sempre 0 se la sequenza codifica lettere.
- Per questo motivo ad ogni passo si fa ruotare ciclicamente la hash temporanea. Cioè si spostano tutti i bit a dx di uno e l'ultimo diviene il primo. In questo modo si migliora l'algoritmo:

$$b_{ci} \stackrel{def}{=} b_{1i} \oplus b_{2(i+1)} \oplus b_{3(i+2)} \cdots \oplus b_{63(i+62)};$$

ottenendo così un altro blocco da 128bit che non inizia necessariamennte con un bit nullo.

# Le funzioni di verifica semplici (a blocchi) - miglioramenti

- Come abbiamo visto i codici ascii delle lettere sono piccoli (da 65 a 122) e quindi non arrivano a coprire tutti i gruppi da 8 bit (il primo bit sarà sempre zero). Quindi anche i blocchi da 128 bit all'inizio avranno sempre 0 se la sequenza codifica lettere.
- Per questo motivo ad ogni passo si fa ruotare ciclicamente la hash temporanea. Cioè si spostano tutti i bit a dx di uno e l'ultimo diviene il primo. In questo modo si migliora l'algoritmo:

$$b_{ci} \stackrel{def}{=} b_{1i} \oplus b_{2(i+1)} \oplus b_{3(i+2)} \cdots \oplus b_{63(i+62)};$$

  ottenendo così un altro blocco da 128bit che non inizia necessariamennte con un bit nullo.
- Anche con questa modifica l'algoritmo è poco costoso computazionalmente: due operazioni (uno shift ed uno xor) per ogni blocco anziché una. Essendo operazioni sequenziali di fatto non cambia nulla.

# Le funzioni di verifica semplici (a blocchi) - miglioramenti

- Come abbiamo visto i codici ascii delle lettere sono piccoli (da 65 a 122) e quindi non arrivano a coprire tutti i gruppi da 8 bit (il primo bit sarà sempre zero). Quindi anche i blocchi da 128 bit all'inizio avranno sempre 0 se la sequenza codifica lettere.
- Per questo motivo ad ogni passo si fa ruotare ciclicamente la hash temporanea. Cioè si spostano tutti i bit a dx di uno e l'ultimo diviene il primo. In questo modo si migliora l'algoritmo:

$$b_{ci} \stackrel{def}{=} b_{1i} \oplus b_{2(i+1)} \oplus b_{3(i+2)} \cdots \oplus b_{63(i+62)};$$

  ottenendo così un altro blocco da 128bit che non inizia necessariamennte con un bit nullo.
- Anche con questa modifica l'algoritmo è poco costoso computazionalmente: due operazioni (uno shift ed uno xor) per ogni blocco anziché una. Essendo operazioni sequenziali di fatto non cambia nulla.
- La ridondanza di controllo è sempre la stessa e quindi la banda di trasmissione effettiva si riduce di $1/64 - esimo$.

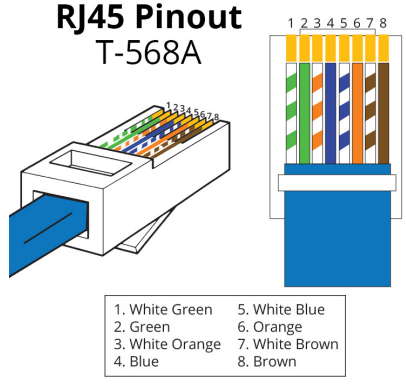
# Connessioni ethernet

- Ethernet è uno standard di comunicazione definito inizialmente nel 1980 definito dallo IEEE (Institute of Electrical and Electronics Engineers) nel 1983 col nome di IEEE 802.3. Si realizza tramite delle apposite schede (in genere sul bus pci) e dei cavi. Ethernet prevede tutti i protocolli per le reti; vedremo in seguito come funzionano le reti, qui ci limitiamo ad alcune considerazioni sulla connessione diretta tra due dispositivi.

# Connessioni ethernet

- Ethernet è uno standard di comunicazione definito inizialmente nel 1980 definito dallo IEEE (Institute of Electrical and Electronics Engineers) nel 1983 col nome di IEEE 802.3. Si realizza tramite delle apposite schede (in genere sul bus pci) e dei cavi. Ethernet prevede tutti i protocolli per le reti; vedremo in seguito come funzionano le reti, qui ci limitiamo ad alcune considerazioni sulla connessione diretta tra due dispositivi.
- I cavi sono terminati con spinotti RJ45:

Un connettore ethernet. Il cavo è composto da 4 doppini di rame incrociati (twisted pairs). Ogni coppia è in identificata da un colore: uno dei cavetti è a colorazione uniforme l'altro alternata con il bianco.



# Connessioni ethernet - cont

- Due dei quattro doppini sono allocati alla spedizione e due alla ricezione delle sequenze binarie.

# Connessioni ethernet - cont

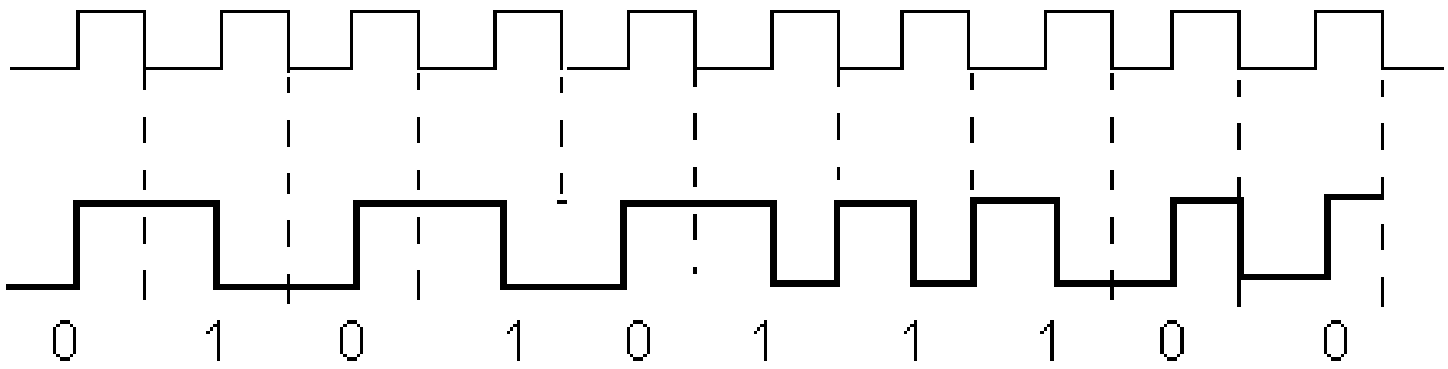
- Due dei quattro doppini sono allocati alla spedizione e due alla ricezione delle sequenze binarie.
- Nel caso della spedizione uno dei doppini trasporta il segnale del clock cioè un'onda quadra alla frequenza della scheda. Mentre l'altro trasporta il segnale effettivo: una sequenza binaria viene rappresentata con un livello nullo della tensione per lo zero ed un livello prefissato per il valore unitario.

# Connessioni ethernet - cont

- Due dei quattro doppini sono allocati alla spedizione e due alla ricezione delle sequenze binarie.
- Nel caso della spedizione uno dei doppini trasporta il segnale del clock cioè un'onda quadra alla frequenza della scheda. Mentre l'altro trasporta il segnale effettivo: una sequenza binaria viene rappresentata con un livello nullo della tensione per lo zero ed un livello prefissato per il valore unitario.
- Analogamente gli altri due doppini trasportano il segnale in ricezione.

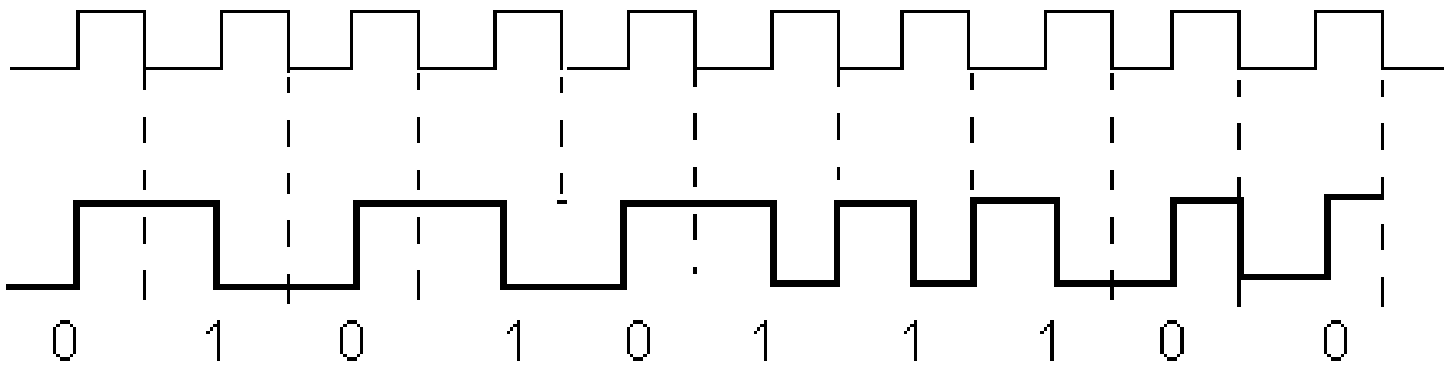
# Connessioni ethernet - cont

- Esempio di trasmissione del segnale



# Connessioni ethernet - cont

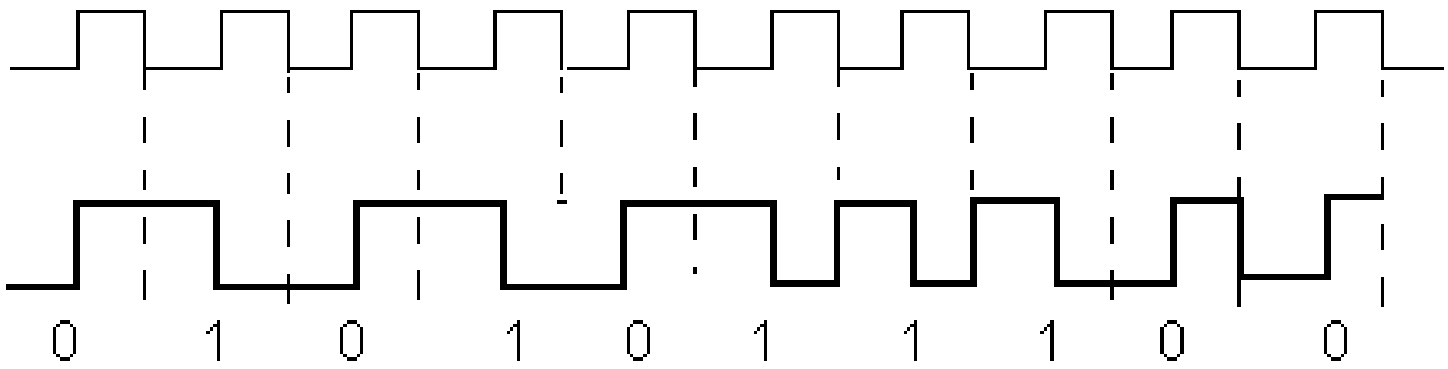
- Esempio di trasmissione del segnale




- Il raddoppiamento dei cavi consente la sincronia dei segnali.

# Connessioni ethernet - cont

- Esempio di trasmissione del segnale




- Il raddoppiamento dei cavi consente la sincronia dei segnali.
- Per definire l'inizio e la fine della trasmissione si usa una sequenza di 16F in formato esadecimale cioè 64 uno di seguito. Nel caso in cui si necessiti di inviare veramente una sequenza di 64 uno si invieranno 15F e poi 8bit nulli, a seguire ancora 8 uno. Gli otto bit nulli dopo 15F vengono ignorati.

# Controllo di integrità ciclico

- Supponiamo di aver inviato una sequenza (vedremo che si chiama "blocco") di dati.

# Controllo di integrità ciclico

- Supponiamo di aver inviato una sequenza (vedremo che si chiama "blocco") di dati.
- La verifica ciclica di integrità consiste nell'allocare una parte di lunghezza prefissata del blocco (collocata tipicamente alla fine) al digest ottenuto tramite una predefinita hash function. La hash function è assolutamente nota a tutti, lo scopo è solo evitare gli errori, non difendersi dagli origliamenti (eavesdropping) o manipolazioni (tampering).

# Controllo di integrità ciclico

- Supponiamo di aver inviato una sequenza (vedremo che si chiama "blocco") di dati.
- La verifica ciclica di integrità consiste nell'allocare una parte di lunghezza prefissata del blocco (collocata tipicamente alla fine) al digest ottenuto tramite una predefinita hash function. La hash function è assolutamente nota a tutti, lo scopo è solo evitare gli errori, non difendersi dagli origliamenti (eavesdropping) o manipolazioni (tampering).
- Supponiamo che la lunghezza del digest sia fissata in $n$ bit collocati alla fine della sequenza trasmessa di lunghezza totale $L + n$. Dal punto di vista matematico la lunghezza della sequenza infomativa è $L$, mentre $n$ è la componente di ridondanza. Sia $m$ il numero binario corrispondente alla sequenza informativa composto da $L$ bit $(b_1, b_2, \ldots, b_L)$:

$$m \stackrel{def}{=} \sum_{i=1,L} b_i \cdot 2^{L-i}.$$

# Controllo di integrità ciclico -cont

- Inserire in coda alla sequenza codificante un digest di controllo equivale a moltiplicare per $2^n$ (traslare di n passi a sinistra la sequenza condificante) ed aggiungere un intero $d$ (il digest) a $n$ bit ($0 \leq d < 2^n$):

$$m' \stackrel{def}{=} m \cdot 2^n + d.$$

# Controllo di integrità ciclico -cont

- Inserire in coda alla sequenza codificante un digest di controllo equivale a moltiplicare per $2^n$ (traslare di n passi a sinistra la sequenza condificante) ed aggiungere un intero $d$ (il digest) a $n$ bit ($0 \leq d < 2^n$):

$$m' \stackrel{def}{=} m \cdot 2^n + d.$$

- Per definire il digest bisogna specificare la funzione di hash.

$$d \stackrel{def}{=} f(m);$$

per costruzione $d$ è minore di $2^n$.

# Esempio di controllo di parità a blocchi

- un pacchetto di 8 blocchi da 4 bit in cui l'ultimo blocco è di controllo: $A3179CFX$:

| $A$ | 3 | 1 | 7 | 9 | $C$ | $F$ | $x$ |
|---|---|---|---|---|---|---|---|
| 1010 | 0011 | 0001 | 0111 | 1001 | 1100 | 1111 | $x$ |

$$
\begin{array}{lcc}
A & \oplus & \\
3 & \oplus & \\
1 & \oplus & \\
7 & \oplus & \text{cioè} \\
9 & \oplus & \\
C & \oplus & \\
F & = & \\
x & & 
\end{array}
\qquad
\begin{array}{lc}
1010 & \oplus \\
0011 & \oplus \\
0001 & \oplus \\
0111 & \oplus \\
1001 & \oplus \\
1100 & \oplus \\
1111 & = \\
0101 &
\end{array}
$$

# Esempio di controllo di parità a blocchi

- un pacchetto di 8 blocchi da 4 bit in cui l'ultimo blocco è di controllo: $A3179CFX$:

| $A$ | 3 | 1 | 7 | 9 | $C$ | $F$ | $x$ |
|---|---|---|---|---|---|---|---|
| 1010 | 0011 | 0001 | 0111 | 1001 | 1100 | 1111 | $x$ |

$$
\begin{array}{lcc}
A & \oplus & \\
3 & \oplus & \\
1 & \oplus & \\
7 & \oplus & \text{cioè} \\
9 & \oplus & \\
C & \oplus & \\
F & = & \\
x & & 
\end{array}
\qquad
\begin{array}{lc}
1010 & \oplus \\
0011 & \oplus \\
0001 & \oplus \\
0111 & \oplus \\
1001 & \oplus \\
1100 & \oplus \\
1111 & = \\
0101 &
\end{array}
$$

- X=5, quindi la sequenza completa è: $A3179CF5$

# Esempio di controllo di parità ciclizzata

- un pacchetto di 8 blocchi da 4 bit in cui l'ultimo blocco è di controllo: $A3179CFX$
  [t trasforma una sequenza ciclicamente: t(wxyz)=zwxy]:

| $A$ | 3 | 1 | 7 | 9 | $C$ | $F$ | $x$ |
|---|---|---|---|---|---|---|---|
| 1010 | 0011 | 0001 | 0111 | 1001 | 1100 | 1111 | $x$ |

$$
\begin{array}{rcc}
A & \oplus & \\
t(3) & \oplus & \\
t(t(1)) & \oplus & \\
t(t(t(7))) & \oplus & \\
t(t(t(t(9)))) & \oplus & \\
t(t(t(t(t(C))))) & \oplus & \\
t(t(t(t(t(t(F)))))) & = & \\
X & & 
\end{array}
\quad \text{cioè} \quad
\begin{array}{rcc}
1010 & \oplus & \\
1 \to\ 0011 & \oplus & \\
2 \to\ 0001 & \oplus & \\
3 \to\ 0111 & \oplus & \\
4 \to\ 1001 & \oplus & \\
5 \to\ 1100 & \oplus & \\
6 \to\ 1111 & = & \\
X & & 
\end{array}
\quad \text{cioè} \quad
\begin{array}{rcc}
1010 & \oplus & \\
1001 & \oplus & \\
0100 & \oplus & \\
1110 & \oplus & \\
1001 & \oplus & \\
0110 & \oplus & \\
1111 & = & \\
1001 & &
\end{array}
$$

# Esempio di controllo di parità ciclizzata

- un pacchetto di 8 blocchi da 4 bit in cui l'ultimo blocco è di controllo: $A3179CFX$
  [t trasforma una sequenza ciclicamente: t(wxyz)=zwxy]:

| $A$ | 3 | 1 | 7 | 9 | $C$ | $F$ | $x$ |
|---|---|---|---|---|---|---|---|
| 1010 | 0011 | 0001 | 0111 | 1001 | 1100 | 1111 | $x$ |

$$
\begin{array}{rcc}
A & \oplus & \\
t(3) & \oplus & \\
t(t(1)) & \oplus & \\
t(t(t(7))) & \oplus & \text{cioè} \\
t(t(t(t(9)))) & \oplus & \\
t(t(t(t(t(C))))) & \oplus & \\
t(t(t(t(t(t(F)))))) & = & \\
X & &
\end{array}
\qquad
\begin{array}{rcc}
1010 & \oplus & \\
1 \rightarrow 0011 & \oplus & \\
2 \rightarrow 0001 & \oplus & \\
3 \rightarrow 0111 & \oplus & \text{cioè} \\
4 \rightarrow 1001 & \oplus & \\
5 \rightarrow 1100 & \oplus & \\
6 \rightarrow 1111 & = & \\
X & &
\end{array}
\qquad
\begin{array}{rcc}
1010 & \oplus & \\
1001 & \oplus & \\
0100 & \oplus & \\
1110 & \oplus & \\
1001 & \oplus & \\
0110 & \oplus & \\
1111 & = & \\
1001 & &
\end{array}
$$

- X=9, quindi la sequenza completa è: $A3179CF9$

# Messaggio

- L'integrità rappresenta un importante requisito per i dati e per i dispositivi atti a gestirli ed immagazzinarli.

# Messaggio

- L'integrità rappresenta un importante requisito per i dati e per i dispositivi atti a gestirli ed immagazzinarli.
- Garantire l'integrità non è possibile in assoluto ma è possibile ridurre la probabilità che si presenti.

# Messaggio

- L'integrità rappresenta un importante requisito per i dati e per i dispositivi atti a gestirli ed immagazzinarli.
- Garantire l'integrità non è possibile in assoluto ma è possibile ridurre la probabilità che si presenti.
- I meccanismi che aumentano la garanzia di integrità dei dati richiedono ridondanza ovvero allocazione non minimale di risorse.

# Messaggio

- L'integrità rappresenta un importante requisito per i dati e per i dispositivi atti a gestirli ed immagazzinarli.
- Garantire l'integrità non è possibile in assoluto ma è possibile ridurre la probabilità che si presenti.
- I meccanismi che aumentano la garanzia di integrità dei dati richiedono ridondanza ovvero allocazione non minimale di risorse.
- La ridondanza si contrappone all'efficienza nella gestione dei dati e delle risorse, quindi è necessario valutare la giusta percentuale di risorse suppletive da allocare alla ridondanza strutturata che aumenta l'integrità dei dati.

# Messaggio

- L'integrità rappresenta un importante requisito per i dati e per i dispositivi atti a gestirli ed immagazzinarli.
- Garantire l'integrità non è possibile in assoluto ma è possibile ridurre la probabilità che si presenti.
- I meccanismi che aumentano la garanzia di integrità dei dati richiedono ridondanza ovvero allocazione non minimale di risorse.
- La ridondanza si contrappone all'efficienza nella gestione dei dati e delle risorse, quindi è necessario valutare la giusta percentuale di risorse suppletive da allocare alla ridondanza strutturata che aumenta l'integrità dei dati.
- Per difendere dalle manipolazioni intenzionali dei dati le hash function devono essere "sicure".

# Esempio Esercizio su RSA

- Usando la cifratura:

$$c = m^k (\ mod\ n).$$

con $n = 7957$ e chiave pubblica $k = 341$, trovare la funzione di decifrazione, cioè una chiave segreta $\alpha$:

$$t = c^\alpha (\ mod\ n).$$

# Esempio Esercizio su RSA

- Usando la cifratura:

$$c = m^k (\ mod\ n).$$

con $n = 7957$ e chiave pubblica $k = 341$, trovare la funzione di decifrazione, cioè una chiave segreta $\alpha$:

$$t = c^\alpha (\ mod\ n).$$

- Ricavare la decomposizione in fattori di n usando il metodo di Fermat (problema del decrittatore). Quanti passi servono?

# Esempio Esercizio su RSA

- Usando la cifratura:

$$c = m^k (\ mod\ n).$$

con $n = 7957$ e chiave pubblica $k = 341$, trovare la funzione di decifrazione, cioè una chiave segreta $\alpha$:

$$t = c^\alpha (\ mod\ n).$$

- Ricavare la decomposizione in fattori di n usando il metodo di Fermat (problema del decrittatore). Quanti passi servono?
- La chiave 341 per cifrare, è ben scelta? Se ne poteva usare una meno grande? Quante operazioni servono per cifrare e decifrare?

# Esempio Esercizio su RSA

- Usando la cifratura:

$$c = m^k (\ mod\ n).$$

con $n = 7957$ e chiave pubblica $k = 341$, trovare la funzione di decifrazione, cioè una chiave segreta $\alpha$:

$$t = c^\alpha (\ mod\ n).$$

- Ricavare la decomposizione in fattori di n usando il metodo di Fermat (problema del decrittatore). Quanti passi servono?
- La chiave 341 per cifrare, è ben scelta? Se ne poteva usare una meno grande? Quante operazioni servono per cifrare e decifrare?
- Trovare tutte le possibili chiavi segrete. Quante sono minori di $n$?

# Soluzione Esercizio - Decrittatore

- Usiamo il metodo di Fermat:

$$k_0 = [\sqrt{n}] = [\sqrt{7957}] + 1 = [89.2..] + 1 = 89 + 1 = 90.$$

# Soluzione Esercizio - Decrittatore

- Usiamo il metodo di Fermat:

$$k_0 = [\sqrt{n}] = [\sqrt{7957}] + 1 = [89.2..] + 1 = 89 + 1 = 90.$$

- Iniziamo con $k_0$:
  90**2=8100; 8100-7957=143: non è un quadrato perfetto.
  Incrementiamo di 1:
  $91^2 \equiv 143 + 90 * 2 + 1 = 324 = 18^2$: è un quadrato perfetto!
  [per passare da $k$ a $k+1$ basta sommare $2k+1$:
  $(k+1)^2 - k^2 = 2k + 1$]

# Soluzione Esercizio - Decrittatore

- Usiamo il metodo di Fermat:

$$k_0 = [\sqrt{n}] = [\sqrt{7957}] + 1 = [89.2..] + 1 = 89 + 1 = 90.$$

- Iniziamo con $k_0$:
  90**2=8100; 8100-7957=143: non è un quadrato perfetto.
  Incrementiamo di 1:
  $91^2 \equiv 143 + 90 * 2 + 1 = 324 = 18^2$: è un quadrato perfetto!
  [per passare da $k$ a $k+1$ basta sommare $2k+1$:
  $(k+1)^2 - k^2 = 2k + 1$]
- La semisomma s è 91, la semidifferenza d è 18. Quindi p= s+d=(91+18=109); q=s-d=91-18=73.

# Soluzione Esercizio - Decrittatore

- Usiamo il metodo di Fermat:

$$k_0 = [\sqrt{n}] = [\sqrt{7957}] + 1 = [89.2..] + 1 = 89 + 1 = 90.$$

- Iniziamo con $k_0$:
  90**2=8100; 8100-7957=143: non è un quadrato perfetto.
  Incrementiamo di 1:
  $91^2 \equiv 143 + 90 * 2 + 1 = 324 = 18^2$: è un quadrato perfetto!
  [per passare da $k$ a $k+1$ basta sommare $2k+1$:
  $(k+1)^2 - k^2 = 2k + 1$]
- La semisomma s è 91, la semidifferenza d è 18. Quindi p= s+d=(91+18=109); q=s-d=91-18=73.
- Bastavano 2 passi di Fermat...

# Soluzione Esercizio - Decrittatore

- Usiamo il metodo di Fermat:

$$k_0 = [\sqrt{n}] = [\sqrt{7957}] + 1 = [89.2..] + 1 = 89 + 1 = 90.$$

- Iniziamo con $k_0$:
  90**2=8100; 8100-7957=143: non è un quadrato perfetto.
  Incrementiamo di 1:
  $91^2 \equiv 143 + 90 * 2 + 1 = 324 = 18^2$: è un quadrato perfetto!
  [per passare da $k$ a $k+1$ basta sommare $2k+1$:
  $(k+1)^2 - k^2 = 2k + 1$]
- La semisomma s è 91, la semidifferenza d è 18. Quindi p= s+d=(91+18=109); q=s-d=91-18=73.
- Bastavano 2 passi di Fermat...
- In generale si scrivono i quadrati perfetti minori di n; in questo caso: 1, 4=1+3, 9=4+5, 16=9+7, 25, 36, 49, 64, 81, 100, 121, 144, 169, 289, 324, 361, 400, 441, 484, 529=484+45, 576=529+47, 625, ... Ma qui bastano due passi...

# Soluzione Esercizio - Decifratore

- Cerchiano il massimo periodo $\lambda$ (Carmichel):

$$\lambda = mcm(72, 108) = mcm(9 * 8, 27 * 4) = 27 * 8 = 216.$$

# Soluzione Esercizio - Decifratore

- Cerchiano il massimo periodo $\lambda$ (Carmichel):

$$\lambda = mcm(72, 108) = mcm(9 * 8, 27 * 4) = 27 * 8 = 216.$$

- Trasformiamo l'equazione della chiave segreta $\alpha$:
$341\alpha = 1 \mod(216)$ in un sistema equivalente:

$$\begin{cases} 341\alpha = 1 \mod(27), \\ 341\alpha = 1 \mod(8); \end{cases}$$

$$\begin{cases} 17\alpha = 1 \mod(27), \\ 5\alpha = 1 \mod(8); \end{cases} \quad \begin{cases} 17(5 + j \cdot 8) = 1 \mod(27), \\ \alpha = 5 \mod(8); \end{cases}$$

$$85 + j \cdot 136 = 1 \mod(27); \quad j = 1 - 4 = -3 \mod(27)$$

$$\alpha = 5 - 3 \cdot 8 = -19 = 216 - 19 = 197.$$

# Soluzione Esercizio - Decifratore cont.

- La chiave pubblica 341 poteva essere sostituita con 125: $125 = 341 - \lambda = 341 - 216$. Le chiavi pubbliche equivalenti sono 37: $\{125, 341, 557, ..., 4877, 5093, 5309, 5525, ..., 7901\}$ Scegliendo a caso un numero servono mediamente $216 log_2(125) = 1512$ operazioni di prodotti seguiti da resto mod(n). [Usando QuickPower]

# Soluzione Esercizio - Decifratore cont.

- La chiave pubblica 341 poteva essere sostituita con 125: $125 = 341 - \lambda = 341 - 216$. Le chiavi pubbliche equivalenti sono 37: $\{125, 341, 557, ..., 4877, 5093, 5309, 5525, ..., 7901\}$ Scegliendo a caso un numero servono mediamente $216 log_2(125) = 1512$ operazioni di prodotti seguiti da resto mod(n). [Usando QuickPower]
- Analogamente le chiavi segrete equivalenti sono 36 e differiscono tutte per multipli di 216: $\{197, 413, ..., 7541, 7757\}$

# Soluzione Esercizio - Decifratore cont.

- La chiave pubblica 341 poteva essere sostituita con 125: $125 = 341 - \lambda = 341 - 216$. Le chiavi pubbliche equivalenti sono 37: $\{125, 341, 557, ..., 4877, 5093, 5309, 5525, ..., 7901\}$ Scegliendo a caso un numero servono mediamente $216 log_2(125) = 1512$ operazioni di prodotti seguiti da resto mod(n). [Usando QuickPower]
- Analogamente le chiavi segrete equivalenti sono 36 e differiscono tutte per multipli di 216: $\{197, 413, ..., 7541, 7757\}$
- Per cifrare (con la chiave minima) servono circa $4(log_2(125)) = 24$ operazioni (12 prodotti, 6 resti e 6 if) e 7 memorie di interi per cifrare; 28 operazioni e 8 memorie di interi per decifrare. [Usando QuickPower] con chiave pubblica 341 servono circa $4(log_2(341)) = 32$ operazioni: solo 8 operazioni in più.
  Attenzione: non si deve scegliere sistematicamente la chiave minima perché l'attaccante avrebbe una stima dell'ordine di grandezza del valore di $\lambda$