L'algoritmo RSA: un approccio algebrico ed astratto

Dott. Ing. Tiziano Binda

Sommario 1. Da rivedere: Il seguente lavoro mira ad esaminare l'algoritmo a chiave pubblica/privata RSA. Per dimostrare la valenza teorica di tale algoritmo si presenteranno il teorema di Eulero/Fermat e il piccolo teorema di Fermat secondo l'approccio di Hernstein. Si illustreranno le basi della teoria dei gruppi, in particolare i gruppi finiti, le classi di resto, laterali destri e teorema di Lagrange per i gruppi finiti. Si faranno poi osservazioni di carattere algoritmico e computazionale per giustificare l'efficacia pratica di questo algoritmo, e metterne in evidenza i limiti.

Ok . Direi che sta in piedi, nonstante la figura "itinerari a new york" sia al solito mancante e l'assenza della bibliografia.

Esporta come pdfLuaTex. Da rivedere i caratteri per il ceiling.

Indice

CAPITOLO 1

La crittografia: una introduzione storica

Una delle prime notizie documentate dell'uso della crittografia, come la intendiamo oggi, è storicamente attribuita a Giulio Cesare. Egli, come evidenziato dalla *Vita dei Cesari* di Svetonio, usava già una sorta di cifratura per inviare i suoi dispacci ai suoi luogotenenti. Questo accadeva quasi 21 secoli fa.

> Per migliaia di anni, re, regine e generali hanno avuto bisogno di comunicazioni efficienti per governare i loro Paesi e comandare i loro eserciti. Nel contempo, essi compresero quali conseguenze avrebbe avuto la caduta dei loro messaggi in mani ostili: informazioni preziose sarebbero state a disposizione delle nazioni rivali e degli eserciti nemici. Fu il pericolo dell'intercettazione da parte degli avvessari a promuovere lo sviluppo di codici e cifre, tecniche di alterazione del messaggio destinate a renderlo comprensibile solo alle persone autorizzate. –Simon Singh, "Codici & Segreti" - Introduzione

1.1. L'esigenza della comunicazione di informazioni riservate

Nell'era dell'informazione non c'è molto da dire sull'esigenza di comunicare in modo sicuro, e segreto, fra due parti. Oggi siamo costantemente collegati con gli altri e spesso abbiamo bisogno di fare comunicazioni in modo riservato, garantendo l'autenticità del contenuto, del mittente e anche del destinatario. L'esempio più emblematico è quello dell'*home banking*, ovvero della gestione diretta attraverso internet del proprio conto corrente.

Un cliente ha la necessità di autenticarsi, ovvero di fornire credenziali che dimostrino il più inequivocabilmente possibile che egli è chi asserisce di essere. Per esempio la mia banca raggiunge questo scopo attraverso una terna di informazioni: Nome Utente, Password, Data Chiave. E' chiaro che *chiunque* avesse a disposizione queste tre informazioni sarebbe a tutti gli effetti identificato come titolare di quel conto corrente, e come tale gli sarebbe garantito libero accesso a tutte le funzioni dispositive, per esempio fare un bonifico di tutto il denaro su un altro conto corrente, magari estero.

Più in piccolo possiamo pensare a una tessera bancomat. Per poter accedere ai servizi bancomat una persona deve fornire, oltre alla tessera, anche un numero che identifichi il titolare.

E' forse meno ovvio che il problema dell'autenticazione può scaturirsi anche al contrario, ovvero anche il destinatario spesso ha bisogno di identificarsi in maniera inequivocabile. Immaginiamo di accedere a uno sportello bancomat truffaldino[1]: dentro il bancomat c'è un tizio che prende la vostra tessera, legge il vostro PIN (Personal Identifier Number) e li immette in un secondo, e vero, bancomat. Voi avete chiesto 100 Euro e lui ne preleva 200, tenendosi 100 Euro come commissione. Benchè buffo questo tipo di *attacco* è teoricamente e praticamente possibile e in

[1]C'è un film (*Killer per caso*) *in* cui Ezio Greggio compie esattamente questa manovra.

nomenclatura va sotto il nome di *Man in the middle (e trova nell'hijacking* la sua miglior realizzazione), ovvero furto di credenziali.

(A) Party Man in The midlle

FIGURA 1.1.1
Party "Man in the Middle"

Per tornare all'esempio dell'*homebanking* se un sito potesse spacciarsi per il sito della nostra banca potrebbe, una volta convinti noi stessi di essere il sito originale, rubarci le credenziali di accesso e divertirsi a piacimento con i nostri dati.

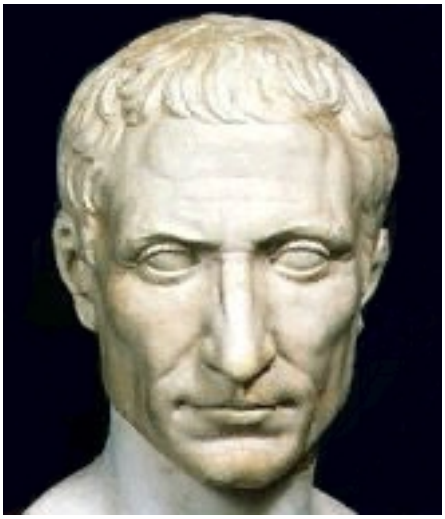
Questo tipo di attacco è possibile anche se tecnicamente complicato da mettere in piedi. Questo spiega perché quando ci colleghiamo per la prima volta al sito della banca, ci appaia una finestra che ci chiede se accettare oppure no un certo certificato di autenticità del sito stesso.

Allo stesso modo, le informazioni che inviamo al sito devono essere protette. Esse devono essere disponibili a noi, alla banca e a nessun altro. Se qualcuno riuscisse a monitorare la connessione e da questa attività estrapolare le informazioni di autenticazione saremmo punto e a capo. Inoltre potrebbe anche curiosare informazioni sensibili, come per esempio i saldi dei nostri conti correnti e le operazioni che eseguiamo. Quindi l'intero traffico di informazioni deve essere in qualche modo protetto fra mittente e destinatario da occhi indiscreti.

1.2. Il "cifrario Cesare"

Agli albori, le informazioni sensibili erano associate più all'attività militare e politica che a quella finanziaria. Il primo uso ampamente documentato di un sistema crittografico risale a Gaio Giulio Cesare. Egli, a partire dalle campagne in Gallia, prese l'abitudine di comunicare i suoi dispacci scrivendoli su pergamena e affidandoli poi a galoppini, una sorta di servizio postale. Dato però che si trovava in territorio nemico, e che anche i galli conoscevano il latino, si presentava l'ovvia necessità di impedire che queste informazioni potessero cadere in mano nemica. Se i galli, catturando o corrompendo un galoppino, avessero potuto sapere in anticipo tattiche, entità, posizioni delle truppe e ordini dello stesso Cesare ne avrebbero tratto un ovvio vantaggio tattico e strategico.

Per ovviare a questo problema, e anche per certificare che l'autore del dispaccio fosse egli stesso, Cesare prese a scrivere i suoi dispacci in modo diverso: prima li scriveva in latino, poi li trascriveva sulla pergamena da affidare al galoppino applicando un trucco. Anzichè scrivere per esempio "A", egli scriveva "D". Al posto di "B" scriveva "E" e così via. Chiaramete al posto di "Z" metteva "C" e al posto di "Y" metteva "B". Ogni lettera veniva, quindi, scambiata con quella che stava tre posizioni più avanti. Inoltre per ovviare al problema delle ultime tre



(A) Gaio Giulio Cesare

FIGURA 1.2.1. Gaio Giulio Cesare

lettere dell'alfabeto egli stabilì che dopo "Z" si dovesse ricominciare con "A" e così via, chiudendo quindi il *protocollo* [2]di cifratura.

Allo stesso modo, invertendo l'associazione (ovvero spostando indietro di tre posizioni nell'alfabeto con l'ovvia precauzione di far precedere "Z" ad "A") si poteva tornare al messaggio originale, ovvero decifrare il messaggio.

Questo tipo di protocollo a noi può far sorridere, e forse far venire il dubbio che i galli non fossero questi maestri di sagacia. Va però detto che questo fu il primo caso di cifratura della storia, e che, almeno secondo alcuni storiografi, da quella volpe che Cesare era, faceva accompagnare i dispacci cifrati da altri in chiaro non del tutto affidabili. Così i galli si tenevano le informazioni in chiaro senza sospettare che invece le altre in versione criptica avessero una qualche valenza.

1.3. Cenni di crittografia simmetrica

Il *cifrario Cesare*, come viene chiamato, è un esempio di crittografia simmetrica. Come è facile notare il sistema di cifratura/decifratura si inverte con facilità. Se indichiamo con CC(3) l'algoritmo che sostituisce a ogni lettera quella che sta tre posizioni più avanti secondo il protocollo discusso sopra, è ovvio che CC(-3) eseguirà la decifratura. Se preferiamo possiamo creare un algoritmo DCC(3)=CC(-3) che esegue la decifratura. La simmetria di questi codici sta nel fatto che l'algoritmo di cifratura è in sostanza lo stesso che si usa in decifratura: basta una banale inversione del parametro di cifratura.

1.3.1. Codici monoalfabetici: crittografia da settimana enigmistica. Ogni tipo di codice che sostituisce "a numero uguale carattere uguale" viene chiamato monoalfabetico. Questo a significare che c'è sempre e solo un'associazione fra un determinato carattere nel testo in chiaro[3] e un determinato carattere del testo cifrato. Quindi, nell'esempio del cifrario Cesare, ogni "A" in chiaro viene

[2]Un protocollo non è altro che un insieme di regole da seguire per rispettare una certa procedura. In questo caso le regole per cifrare e decifrare il messaggio.

[3]Si dice testo in chiaro il testo con il messaggio non codificato

riscritta come "D" nel cifrato. E' evidente che quindi vale anche il viceversa, ovvero ogni "D" nel cifrato dovrà essere ritrasformata in "A" nel testo decifrato.

Per sua stessa costruzione, il cifrario Cesare può essere esteso fino a fornire, usando l'alfabeto inglese, 26 diversi modi di crittare uno stesso messaggio. E' lapalissiano che CC(0)=CC(26) e che applicare CC(0) a un testo produrrà nient'altro che il testo in chiaro stesso. Ci sono quindi 26 diversi codici Cesare costruibili con questo protocollo, dei quali uno banale.

L'algoritmo di cifratura/decifratura rimane identico per tutti e 25 i cifrari cesare. Quello che cambia è l'entità dello spostamento in avanti. In tutti i sistemi di cifratura esistono questi due elementi: l'algoritmo e la *chiave*, ovvero il parametro, che fa lavorare in qualche modo l'algoritmo. Anche se si conosce l'algoritmo, la mancanza della chiave rende comunque difficoltoso risalire al messaggio in chiaro. Questo si ottiene, ovviamente, avendo un'ampia scelta di chiavi ovvero avendo un insieme di chiavi grande. Così da garantire che anche provandole tutte, il tempo necessario per trovare quella giusta sia mediamente improponobile.

Per esempio per *violare* **un** cifrario cesare non ci vuole molta fatica. Una volta che si sa che un messaggio è stato cifrato usando *un* cifrario Cesare basta provarli tutti e 25 per trovare quello giusto ed avere quindi la chiave di decifrazione. Di particolare nota il fatto che non è affatto necessario trasformare tutto il testo in questa fase, ma solo poche parole e verificare se esse hanno senso. Appena trovata una possibile corrispondenza si decifra tutto il testo e "cest voilà tout" il gioco è fatto.

Va da sè che spesso fa comodo aggiungere caratteri al nostro alfabeto: segni di punteggiatura, accenti e gli stessi spazi bianchi sono elementi importanti che devono essere considerati. Quindi si può costruire un alfabeto esteso che contenga tutti i simboli che noi vogliamo nascondere e associare a ogni simbolo un altro nel medesimo alfabeto. Così facendo, e dando il sistema di trascrizione fra un alfabeto e l'altro, si può eseguire la cifratura e la decifratura del messaggio. Siccome si usa un singolo alfabeto, o meglio una singola corrispondenza biunivoca fra l'alfabeto in chiaro e quello cifrato, questi codici vengono anche detti *monoalfabetici*.

Per dare qualche numero, se si dispone di un alfabeto di 10 caratteri, quanti codici monoalfabetici, anche banali, si possono costruire?

Al primo carattere si può associare uno qualsiasi degli altri 10. Al secondo uno degli altri 9 rimasti (uno è andato "perso" dato che è già stato associato al primo carattere). Al terzo uno degli altri 8 rimasti e così via. Dato la ovvia indipendenza della scelta del secondo carattere rispetto alla scelta del primo, stante che non si possono scegliere lo stesso carattere, per il resto non ci sono vincoli: il numero totale di alfabeti diversi è $1 * 2 * 3 * ... * 9 * 10 = 10! = 3628800$. Ora se passassimo a un alfabeto a 26 caratteri, come ad esempio l'inglese, avremmo $26! > 4 * 10^{26}$ che è un numero decisamente enorme.

Questi numeri potrebbero far credere che rompere un codice di questo tipo sia quindi un'impresa titanica. Eppure, come nel racconto "Gli omini danzanti" di Sherlock Holmes, è possibile, e spesso senza nemmeno avere a disposizione grandi potenze di calcolo.

Conoscendo informazioni del messaggio, come per esempio la lingua in cui è scritto, il tema di cui tratta, lo stile dell'autore, è possibile fare delle osservazioni di carattere statistico. Per esempio in una certa lingua come l'inglese, il carattere più spesso ripetuto è la vocale "e". Quindi se disponiamo di un testo cifrato di una certa lunghezza, possiamo ipotizzare che il carattere che occorre più spesso sia quello associato al carattere "e". Andando avanti in questo modo, studiando le parole che si ottengono e scartando le ovvie associazioni che portano a "non sense", si possono indovinare, sempre più in fretta, le altre associazioni fra lettere dell'alfabeto in

chiaro e lettere dell'alfabeto cifrato. E proprio così che questa generazione di codici divenne rapidamente obsoleta e *debole*, nel senso che si poteva facilmente risalire al testo in chiaro anche senza *la chiave*.

1.4. La seconda guerra mondiale e la macchina "Enigma"

L'idea successiva fu quella di non basarsi su un unico alfabeto. In effeti se ne possono usare per esempio due: uno per i caratteri in posizione dispari e uno per quelli nelle posizioni pari. Oppure ne potrei usare 10: uno per il primo carattere, un secondo per il secondo e così via fino al decimo per poi ripartire dal primo alfabeto.

In questo modo anche il numero degli alfabeti diventa un'incognita e non di poco conto.

1.4.1. Codici polialfabetici: crittografia "da prete". Storicamente, uno dei primi a proporre un sistema di questo tipo per cifrare testi fu un abate tedesco: Trithemius, nato nel 1472. Per la loro natura non monoalfabetica, questi codici si dicono *polialfabetici*, e per lungo tempo i crittografi si sentirono frustrati nell'impossibilità di violarli.

Su questo principio venne costruita la macchina enigma, strumento di crittografia usato dai tedeschi durante la seconda guerra mondiale. Convinti di avere uno strumento inviolabile, i tedeschi usarano pesantemente la cifratura e ci si fidarono ciecamente. Eppure uno dei padri dell'informatica, Alan Turing, aveva scoperto delle falle in questo sistema, e mise il suo genio a disposizione del proprio paese, la Gran Bretagna, per violare engima. Usando l'analisi statistica, alcune debolezze dovute ai fattori umani e ad alcune imperfezioni tecniche, egli risucì, in genere, ad aver la meglio su Enigma. E per quando le sue brillanti intuizioni non bastavano, aveva costruito delle "bombe" (veri e propri calcolatori) che provavano brutalmente tutte le possibilità.

1.5. L'informatica e la crittografia

Come si vede il problema di violare la crittografia era ormai diventato un problema di matematica, ingegneria elettrica e sociale, ovvero quello di sfruttare gli errori e la pigrizia umana, e un problema di informatica, nel senso di provare il più rapidamente possibile una quantità di chiavi nel tentativo di isolare quella giusta.

1.5.1. Macchine dell'informazione. I calcolatori, proprio per la loro natura, sono gli stumenti ideali per implementare sistemi crittografici e strumenti di rottura di sistemi crittografici. L'aumento esponenziale dei messaggi scambiati durante l'ultima guerra mondiale ha creato un problema di sovrabbonadanza di informazione che andava immagazzinata e processata. Si è stimato che l'esercito nazista abbia trasmesso ben oltre un milione di dispacci nell'ultima guerra. Le capacità di immagazzinare, processare e fare ricerche è diventata critica, oltre a quella di poter eseguire operazioni aritmetico logico massivamente su enormi moli di dati.

1.5.2. La chiave, forza di un protocollo simmetrico e il problema computazionale. Ora che si è chiarito che la forza di un protocollo sta nella chiave vediamo di chiarire meglio questo punto.

Se le chiavi disponibili fossero poche, basterebbe provarle tutte e analizzare i singoli testi decifrati proposti per trovare poi quello giusto. Quindi va da sè che dobbiamo avere un vasto insieme di chiavi nel quale prendere la nostra, cosicchè la probabilità che scegliendone a caso una si prenda quella giusta sia piccola. Per esempio un insieme di un miliardo di chiavi può sembrare adeguato, ma se si pensa

con quale velocità oggi un computer di media potenza può provare queste combinazioni, ci si rende conto che non sono molte. Per amor di chiacchiera si immagini che un computer possa provare un milione di chiavi al secondo, cifra tutto sommato ragionevole, allora gli basterebbero 1000 secondi, meno di venti minuti, per esaurire tutto lo *spazio* delle chiavi. Una sicurezza probabilmente insufficiente per la maggior parte delle possibili applicazioni. Si tenga poi conto che grandi società e organizzazioni governative possono disporre di ben altre risorse informatiche. Si conclude che per avere delle garanzie da parte del nostro protocollo dobbiamo avere uno spazio delle chiavi *molto* più grande. Si ricordi che in media è sufficiente provare la metà di tutte le chiavi per trovare quella giusta. Inoltre è verosimile che un computer abbastanza veloce possa provare 1000 miliardi di chiavi in un giorno (pari a 10 milioni al secondo circa), che in un anno sono quasi 500 mila miliardi di chiavi.

Si conclude quindi che per avere una chiave che mediamente possa resistere *almeno* un anno, bisognerebbe avere uno spazio di almeno un milione di miliardi di chiavi, sempre che i computer non aumentino di prestazioni nel frattempo.

1.5.3. La chiave, debolezza di un protocollo simmetrico. In tutto questo discorso c'è un unico grosso neo: la chiave. Essa deve essere disponibile sia al mittente che al destinatario. Deve quindi essere in qualche modo distribuita fra di essi in modo *sicuro*. Va da sè che un modo di distribuire dati in modo sicuro è proprio quello che ci manca, e quindi è proprio questo l'anello debole di questa catena. Inoltre ogni chiave è soggetta a un fenomeno di *usura* (invecchiamento o aging in inglese). Più viene adoperata piu lascia tracce di sè. Queste tracce furono utilizzate proprio durante la seconda guerra mondiale a Bletchey Park per trovare sistemi decisamente più rapidi per identificare la chiave che i nazisti usavano quel giorno. Per dirla il più semplicemente possibile, l'unico modo davvero sicuro di crittare un messaggio è quello di usare una chiave grande quanto il messaggio stesso, e di non riutilizzare la chiave mai più. Ogni volta che la si riutilizza si possono usare tecniche e processi statistici per ridurre progressivamente lo spazio delle chiavi. Il risultato è quello di riportare il problema entro valori e tempi accettabili, e fu quello che gli inglesi realizzarono, sia attraverso le bombe di Turnig che Colossus.

Inoltre la distribuzione delle chiavi porta a un problema logistico e organizzativo enorme e estremamente dispendioso, dato che esse devono essere consegnate massivamente e in modo sicuro. Rubare la chiave equivale a rompere il codice a tutti gli effetti.

1.6. La crittografia Asimmetrica

In quest'ottica sono entrati in scena i sistemi asimettrici. Essi usano due chiavi, diverse, una per cifrare e una per decifrare. Quello che deve essere chiaro è che esiste sempre un unico algoritmo di cifratura che chiameremo C. La chiave di cifratura la chiameremo CK (Cipher Key) e quella di decifratura DK(decipher Key). Chiameremo T il testo in chiaro da cifrare, CT (Ciphred Text) il testo cifrato e DT (Dechipred Text) il testo decifrato. Dopo questa mole di definizioni possiamo quindi descrivere l'algoritmo asimmetrico:

Algoritmo 2. *CT=C(T,CK). CT è ottenuto dall'algoritmo C applicato al testo T secondo la chiave CK.*

Viene inviato in modo non sicuro CT al nostro destinatario. Egli calcola quindi DT.

DT=C(CT,DK), applicando l'algoritmo C su CT con la chiave DK. Si deve avere che DT=T e il protocollo si chiude.

Il truco sta nel fatto che il destinatario crea le due chiavi, CK e DK. Si tiene DK per sè, nascosta, e invia, anche in chiaro, CK al mittente. Il mittente applica l'algoritmo C su T con CK e invia CT al destinatario.

Inserire qui un'immagine delle fasi dell'algoritmo

Ora, per ritrasformare CT in DT=T si ha bisogno di DK, che non è stata trasmessa e quindi sta al sicuro dal destinatario. Il fatto di usare due chiavi distinte, di cui una nascosta, garantisce la sicurezza. O almeno sempre che sia *difficile* ricostruire DK da C e CK. Se così non fosse il gioco non starebbe in piedi.

L'idea può essere schematizzata così: Io ho una cassa alla quale applico un mio lucchetto. Mi tengo la chiave e invio la cassa al destinatario. Egli appone un secondo lucchetto, sempre sullo stesso anello, si tiene la chiave e mi rinvia la cassa. Io ricevo la cassa, levo il mio lucchetto con la mia chiave e la rinvio al destinatario che la riceve chiusa con il suo solo lucchetto. Che quindi può levare senza problemi. La cassa ha sempre viaggiato chiusa a chiave, e quindi in modo sicuro.[4]

1.6.1. L'RSA, la chimera della crittografia. Diffie e Hellman teorizzarono per primi questo protocollo nel 1975. Bello in teoria, ma in pratica irrealizzabile per via della procedura "a due lucchetti" che si sposa male con le operazioni che in genere possiamo fare. Infatti quando noi facciamo una sequenza di azioni, per tornare indietro dobbiamo fare le azioni opposte ma in ordine invertito. Immaginate di girarvi a destra e fare 5 passi. poi di girarvi a sinistra e farne altri 5. Per tornare al punto di partenza dovrei fare 5 passi indietro, girarmi a destra, fare altri 5 passi indietro poi girarmi a sinistra. Se invertissi l'ordine delle due rotazioni mi troverei in un posto diverso. La ricerca di una funzione con queste proprietà proseguì per due anni senza nessun successo fino al 1977 quando Rives, Shamir e Adlemann scoprirono come usare i moduli e le operazioni su di essi per implementare il protocollo simmetrico.

Questo protocollo sarà analizzato nel dettaglio nel capitolo 3, dopo aver appreso i rudimenti della teoria dei gruppi, irrinunciabile base teorica a giustificazione dell'efficacia del protocollo.

1.7. Chiave (o chiavi?) e canali sicuri

Per poter cifrare e decifrare si ha quindi bisogno di una informazione comune, la chiave. Essa deve essere la stessa da entrambe le parti, e si pone quindi l'ovvio problema della distribuzione sicura di questa chiave. Se essa cadesse in mani nemiche tutto il sistema sarebbe compromesso. Allo stesso modo se avessi un canale sicuro per trasferire la chiave, perché' non usare lo stesso sistema per trasferire tutto il messaggio? E' il classico cane che si morde la coda.

1.7.1. Il problema computazionale. Al giorno d'oggi è solo un problema di forza bruta, ovvero provare tutte le possibili combinazioni della chiave fino a trovare quella giusta. I computer moderni hanno capacità di calcolo inimmagginabili solo pochi decenni di anni fa. I codici che usiamo non sono intrinsecamente sicuri, ma solo *probabilmente*[5] sicuri. Ovvero oggi ci vorrebbero anni o decine di anni per provare tutte le possibili chiavi. Questo perché' non esistono algoritmi efficienti, ovvero rapidi, per risolvere certi problemi, come per esempio trovare i fattori di un numero intero grande, dell'ordine delle centinaia di cifre decimali. La sicurezza sta *solo* in questo fatto. Quindi, al crescere della potenza di calcolo dei computer, o se si trovasse un procedimento che permettesse di arrivare molto rapidamente alla soluzione, il nostro codice diverrebbe violabile.

[4]Questo bellissimo esempio è preso "as is" direttamente da "Codici & segreti"

[5]Ovvero si ha una probabilità grande che non si possa rompere il codice in un tempo breve. Si noti che la probabilità di non rompere il codice è funzione decrescente del tempo a disposizione.

L'RSA si basa sul fatto che per trovare i fattori di un numero grande ci vuole un tempo proporzionale alla radice del numero stesso, ovvero per trovare i fattori di un numero che vale circa un milione ci vogliono, salvo casi triviali, un migliaio di operazioni. Se il numero ha 10 cifre, ci vogliono un numero con 5 cifre di operazioni (circa 100.000). Per un numero che avesse, per dire, 18 cifre, ci vorrebbero circa un miliardo di operazioni. Per un numero con 100 cifre ci vorrebbero circa 10 alla 50 operazioni, numero decisamente enorme.

CAPITOLO 2

Intermezzo numerico

2.1. Il diffidente uomo inferno

Il protocollo[1] Diffie-Hellman permette di negoziare fra due parti un numero intero e positivo. Usando canali non protetti e facendolo *passare* sotto il naso a potenziali spioni.

Un numero può sembrare poca cosa, ma esso è sufficiente per mettere in piedi una chiave sicura per un sistema di comunicazione simmetrico. Specie se il numero prodotto è di dimensione sufficiente (ovvero la scelta sia ampia).

Chiamiamo al solito Alice (A) e Bob (B) i nostri due soggetti.

Alice produce 3 numeri interi e positivi: a il suo numero segreto (scelto casualmente e grande); g un numero che chiamiamo generatore; p un numero primo di discreta dimensione (passando il tempo diventa sempre più grande: diciamo ad almeno 2000 bit). Da questa terna (a, g, p) Alice produce un nuovo numero A:

$$A := g^a \% p$$

e invia (A,g,p) a Bob come meglio crede, anche con un piccione viaggiatore[2]. O urlandoglieli in mezzo a una piazza.

Bob li riceve. Prepara il suo numero segreto b (scelto casualmente e grande) e da questo calcola

$$B := g^b \% p$$

con g,p che ha mandato Alice. Egli rimanda in qualsiasi modo B ad Alice, probabilmente con lo stesso piccione viaggiatore di partenza[3].

Dalla sua parte Bob calcola

$$K_B = A^b \% p$$

mentre Alice calcolerà

$$K_A = B^a \% p$$

E' evidente che $B^a = \left(g^b\right)^a = g^{ba} = g^{ab} = \left(g^a\right)^b = A^b$ a meno di lasciar cadere i segni di modulo. Non facciamoli cadere allora!

$$K_A = B^a \% p = \left(g^b \% p\right)^a \% p = g^{ba} \% p = g^{ab} \% p = \left(g^a \% p\right)^b \% p = A^b \% p = K_B$$

Il secondo, il terzo, il quinto e sesto $=$ sono validi in virtù del teorema di idempotenza dell'esponenziazione modulare 4 che dimostreremo a breve.

Quello che è importante è che $K_A = K_B = K$ è la chiave che servirà ad Alice e Bob per avviare il loro sistema di comunicazione simmetrico. Il tutto funziona perché ricavare a, b dai valori A, B, g, p ha bisogno di moltissime operazioni: il logaritmo nel gruppo moltiplicativo modulo p[4] non si può fare direttamente (se trovate un algoritmo rapido vincerete un gran bel premio!) e quindi per un tempo

[1] E' evidente che contiene anche un algoritmo, ma il termine protocollo resta più appropriato

[2] Forse una email in chiaro sarebbe meno crudele verso gli animali.

[3] Così da non rimandarlo indietro a vuoto

[4] Il valore x tale che $y = a^x \% p$ si chiama logartimo discreto di y in base a modulo p.

ragionevolmente lungo possiamo immaginare che solo Alice e Bob sapranno quanto vale K. Dopo un po' potranno negoziare un nuovo valore, se necessario.

2.2. Approccio ingenuo all'algebra modulare

In tutto questo contesto useremo numeri interi moltiplicati fra loro e poi soggetto all'operazione di resto modulo m. Ovvero faremo la moltiplicazione e poi calcoleremo il resto rispetto alla divisione per m.

E' evidente che il risultato sarà un valore compreso fra 0 e m-1. Useremo poi dire che due valori sono congrui modulo m se questi valori divisi per m danno lo stesso resto, ovvero $a \equiv b\%m \iff a\%m = b\%m$. Continuando a moltiplicare numeri *modulo m* è evidente che avremo sempre e solo risultati che saranno numeri compresi fra 0 e m-1. Lo studio delle proprietà di questi valori al variare di m sarà fondamentale nel prosieguo.

2.2.1. Ancora sulle classi di resto. Non solo. Ci sono altre propietà, analoghe, che sono equivalenti alla congruenza:

$$a \equiv b\%m \iff m|(a-b) \iff (a-b) \in [m]_{\mathbb{Z},+}$$

Spieghiamo i geroglifici. a e b sono congruenti modulo m se danno lo stesso resto per la divisione di m. Questo è equivalente a dire che la loro differenza è multipla di m, ovvero che m divide a-b. L'ultima è analoga a quella appena espressa, stando il geroglifico $[m]_{\mathbb{Z},+}$ rappresentare il sottogruppo additivo modulo m. (Vedremo più avanti cos'è).

Ora. se $a \equiv b\%m \implies a\%m = b\%m = r$ allora $a = q_a m + r, b = q_b + r$ e quindi $a - b = q_a m + r - (q_b m + r) = q_a m - q_b m$ cioè $a - b = (q_a - q_b)m$ ovvero a-b è multiplo intero di m.

Ora continuiamo con i seguenti teoremi.

2.3. Idempotenza della moltiplicazione modulare

TEOREMA 3. *di idempotenza della moltiplicazione modulare.*

$$(a * b)\%m \equiv (a\%m \ * b\%m)\%m$$

Sembra una sciocchezza, ma in realtà ci sarà molto utile

DIMOSTRAZIONE. $a * b = qm + r$ dove $q = (a * b)//m, r = (a * b)\%m$[5]. A loro volta scriviamo $a = q_a m + r_a, q_a = a//m, r_a = a\%m$[6] e analogamente $b = q_b m + r_b, q_b = b//m, r_b = b\%m$. Calcoliamo $a*b = (q_a m + r_a) * (q_b m + r_b) = ... = (q_a q_b m + q_a r_b + q_b r_a)m + r_a r_b$. E' evidente che allora $a * b\%m \equiv r_a r_b\%m$. Ma per costruzione $r_a = a\%m, r_b = b\%m \implies a * b\%m \equiv r_a r_b\%m \equiv (a\%m)(b\%m)\%m$. □

2.4. Idempotenza della esponenziazione modulare

Nello stesso spirito del teorema precedente

TEOREMA 4. *di idempotenza della esponenziazione modulare. Dati* $a, e, m \in \mathbb{N}$ *allora*

$$a^e\%m \equiv (a^{e-1}\%m) * (a\%m)\%m$$

[5]Diretta conseguenza dell'algortimo della divisione intera

[6]Qui, e spesso nel seguito, è usata la notazione *Python* in cui l'operatore // indica il quoziente della divisione intera

La dimostrazione non serve. E' un corollario del teorema precedente fatte le opportune sostituzioni e procedendo per induzione.

La cosa importante è che lavorando per induzione possiamo scrivere

$$a^2 \% m \equiv ((a\%m) * (a\%m)) \% m$$

$$a^3 \% m \equiv \left(a^2 \% m * a\%m\right) \% m \equiv (((a\%m) * (a\%m)) \% m * a\%m) \% m$$

e così via. E' apparentemente un modo molto strampalato di procedere, ma ha il grosso pregio che a ogni passaggio il valore da trattare è minore di m, cosa che permette di rendere le operazioni di moltiplicazione più rapide e tenere il consumo di memoria sotto controllo.

2.5. Esponenziazione modulare veloce

Il teorema 4 permette di eseguire l'esponenziazione modulare più velocemente. Il fatto che permetta di tenere sotto controllo la dimensione del numero man mano che lo si calcola (così da non eccedere mai m^2) permette di risparmiare molto tempo. Ma si può fare di meglio. *Molto meglio.*

Sia e il nostro esponente ed e_b la sua espressione binaria corripondente, ovvero $e_b = e_n e_{n-1}..e_1 e_0$ siano gli (n+1) bit che rappresentano il numero, intero e positivo, e. Ovvero

$$e = \sum_{i=0}^{n} e_i 2^i$$

[7]

Ora

$$b^e = b^{\sum_{i=0}^{n} e_i 2^i} = b^{e_n 2^n} * b^{e_{n-1} 2^{n-1}} * .. * b^{e_0 2^0} = \prod_{i=0}^{n} b^{e_i 2^i}$$

Che è il prodotto di n+1 termini e non e. Inoltre c'è un legame fra $b^{2^{i+1}}$ e b^{2^i}: $b^{2^{i+1}} = b^{2^i * 2} = (b^{2^i})^2$. Ogni termine ulteriore nella produttoria è il quadrato del precedente, cosa che permette di calcolarlo con un'unica operazione. I termini e_i sono bit e come tali valgono 1 o 0. Nel primo caso significa che il corrispndente b^{2^i} entra nel prodotto per il calcolo di b^e, altrimenti azzera l'esponente e quindi manda il contributo di $b^{2^i} = b^0 = 1$ che non cambia il valore del prodotto[8].

Applicando e reiterando il teorema 3 di idempotenza del prodotto modulare ora possiamo calcolare

$$b^e \% m = \left(\prod_{i=0}^{n} b^{e_i 2^i}\right) \% m = \left(\prod_{i=0}^{n} \left(b^{e_i 2^i} \% m\right)\right) \% m$$

Facendo n+1 moltiplicazioni, n quadrati e (2n+1) operazioni di modulo. **Essendo** $n = \lceil \log_2 e \rceil$, il calcolo, benché appesantito dai moduli, richiede un numero di operazioni proporziale al quadruplo del logaritmo dell'esponente. Cosa che lo rende decisamente più veloce.

---

[7] A un informatico è probabile che verrebbe più spontaneo scrivere

$$e = \sum_{i=n}^{0} e_i 2^i$$

con l'indice che va a decrescere anziché crescere. Solo per avere il bit più pesante per primo (in realtà molti di voi obietterebbero che è più spontaneo continuare a usare la notazione precedente come un ciclo for a incremento).

[8] Questo non significa che possiamo fare a meno di calcolarlo però! Anche se b^{2^i} non ci interessa, è necessario per calcolare $b^{2^{i+1}}$.

Per capirci se e fosse un numero a 2000 bit ($> 10^{600}$) basterebbero circa 8000 operazioni per calcolare il risultato.

Se e fosse un numero a 4000 bit ($> 10^{1200}$) basterebbero circa 16000 operazioni per calcolare il risultato.

Esercizio. Realizzare, diciamo in Python, un programma che esegua l'esponenziazione intelligente e valutarne le prestazioni.

2.6. RSA

Per implementare l'RSA abbiamo bisogno di due grandi numeri primi (che chiameremo p e q), il loro prodotto $n = p * q$ un numero e coprimo con n e il suo *gemello* d, anch'esso coprimo con n ma soprattutto tale che $ed\%\varphi(n) \equiv 1$.[9] Ovvero e, d sono in qualche modo inversi rispetto alla moltiplicazione modulo $\varphi(n)$ (e quindi n come vedremo fra un tantinello).

Allora

Teorema 5. *[RSA]*

$$c = t^e\%n; t = c^d\%n$$

Sembra l'analogo del Diffie Hellman e sostanzialmente le operazioni sono le stesse, se non per la scelta di e e quindi il calcolo di d .

Come trovare però i numeri (e,d)? Un attimo di pazienza e ancora un po' di lavoro.

2.7. Coprimalità ed algoritmo di Euclide

L'algoritmo di Euclide prevede una via molto rapida per il calcolo del massimo comun divisore fra due interi, e quindi anche un test rapido ed efficiente per determinare se due interi sono fra loro coprimi. In particolare ci permette di trovare il più grande intero che divide sia a che b.

Si procede calcolando la seguente successione (dando per scontato che a>b):

Tabella 1. Algoritmo di Euclide

i passo	q quoziente	r resto
0	—	a
1	—	b
2	$a//b=$ $r_0//r_1$	$a\%b=$ $r_0\%r_1=$ $r_0 - q_2 r_1$
...		
i	$r_{i-2}//r_{i-1}$	$r_{i-2}\%r_{i-1}=$ $r_{i-2} - q_i r_{i-1}$
...		
n	$r_{n-2}//r_{n-1}$	$r_{n-2}\%r_{n-1}=0$

[9] La $\varphi(x)$ di Eulero conta quanti sono i numeri compresi fra 1 e x che siano coprimi con x. Più avanti le studieremo più approfonditamente.

$$\begin{gathered} a = bq_2 + r_2 \\ b = r_2q_3 + r_3 \\ r_2 = r_3q_4 + r_4 \\ ... \\ r_{n-3} = r_{n-2}q_{n-1} + r_{n-1} \\ r_{n-2} = r_{n-1}q_n + 0 \end{gathered}$$

Che produce la successione $r_i = (a, b, r_2, r_3, .., r_{n-1}, r_n)$ dove r_{n-1} è l'ultimo valore di resto non nullo calcolato dalla sequenza sopra. La sequenza è decrescente, per via dell'algoritmo della divisione fra interi, e come tale l'algoritmo di Euclide termina in un tempo finito. Essendo $r_n = 0$, r_{n-1} è un divisore di r_{n-2}. Da questo fatto possiamo ricalcolare facendo le opportune sostituzioni

$$r_{n-3} = r_{n-2}q_{n-1} + r_{n-1} = (r_{n-1}q_n)q_{n-1} + r_{n-1} = r_{n-1}(q_nq_{n-1} + 1)$$

tenuto presente che al passo i possiamo scrivere la relazione

$$r_i = r_{i+1}q_{i+2} + r_{i+2}$$

Se $r_{i+1} = r_{n-1}k_{i+1}$ e $r_{i+2} = r_{n-1}k_{i+2}$ [10] allora

$$r_i = r_{n-1}k_{i+1} + r_{n-1}k_{i+2} = r_{n-1}(k_{i+1} + k_{i+2})$$

Ovvero ogni resto della sequenza è multiplo intero del resto r_{n-1} divisore di r_{n-2} e come calcolato anche di r_{n-3}, allora per induzione r_{n-1} sarà divisore di r_0 e quindi di b e a. Ergo r_n è divisore sia di a che di b.

Resta da verificare che non esista un numero più grande di r_n con questa prorietà. Chiamiamo d un valore tale che $d|a, d|b$. Ma quindi possiamo scrivere $a = a'd, b = b'd$ allora $d|r_n$. Per assicurarsene basta vedere che $d|r_0$. Ma $a'd = b'dq + r_0 \implies r_0 = (a' - b'q)d \implies d|r_0$. Con la stessa costruzione, dato che b è multiplo di d come r_0, allora lo sarà anche r_1, in quanto differenza di multipli di d. La conseguenza è una discesa infinita dove per ogni i $r_i : d|r_i$ fino a r_n. Così r_n è divisibile $\forall d : d|a, d|b$ e quindi è il più grande divisore sia di a che di b.[11]

```
def GCD(a,b):
    '''Ritorna il massimo comun divisore fra gli interi a,b'''
    if a<b:
        a,b = b,a #scambia a,b tale che a>b
    r = a % b
    while r>0:
        a = b
        b = r
        r = a%b
        #a, b, r = b, r, a%b
    return b
```

[10] Applicando l'induzione.

[11]

TEOREMA. *Se $d|a$ e $d|b$ allora $\forall n \in \mathbb{N} : n = a \pm b \implies d|n$.*

DIMOSTRAZIONE. $a = da', b = db' \implies n = da' \pm db' = d(a' \pm b') \implies d|n$. □

La dimostrazione precedente, che è quella classica, è decisamente antiintuitiva e un po' oscura in certi passi. Proponiamo allora questàaltra **in alternativa e procediamo per induzione**.[12] A ogni passo calcoliamo il quoziente $q_i = r_{i-2}//r_{i-1}$ e il resto $r_i = r_{i-2} - q_i r_{i-1}$, ovviamente per valori di i non minori di 2. Notiamo che MCD(a,b)=m significa che m è divisore sia di a che di b.

Dimostrazione. Calcoliamo il secondo resto:

$$r_2 = r_0 - q_2 r_1 = a - q_2 b$$

Essendo composto dalla differenze di due termini ciascuno multiplo di m deve essere a sua volta multiplo di m.[13] E quindi la base di induzione è valida. Diciamola valida per tutti i passi fino a $i-1$ e proviamola per il passo i.

$$q_i = r_{i-2}//r_{i-1}$$

$$r_i = r_{i-2} - q_i r_{i-1}$$

Come per il caso r_2 questo resto è la differenza di due valori multipli di m (per ipotesi induttiva r_{i-2} e r_{i-1} lo sono) e quindi a sua volta è multiplo di m. Questo vale anche per l'ultimo resto $r_n = 0$. Per sua natura 0 è divisibile, internamente, per qualsiasi numero, dato che darà sempre resto nullo. □

2.8. Identità di Bezout

Dati due interi a, b e il loro $MCD(a,b) = m$ allora esistono[14] due interi x, y tali che

$$ax + by = m$$

Questa uguaglianza è detta identità di Bezout.

L'esistenza è garantita dal fatto che d è un divisore sia di a,b e quindi essi sono multipli interi di d. Cioè:

$$\exists f : a = fm; \exists g : b = gm$$

applicando all'identitàdi Bezout otteniamo

$$ax + by = fmx + gmy = m(fx + gy) = mz$$

che quindi è un multiplo di m, e se f,g sono coprimi fra loro, z può assumere al minimo anche il valore 1.

Altro intermezzo. Siano a,b coprimi. Usandol'algoritmo di Euclide, m=(a,b)=(b,a%b)=(a%b,b%(a%b)$)$=...

Una dimostrazione formale all'identità di Bezout la omettiamo. Comunque si vedrà che se l'MCD vale 1, allora è possibile costruire i due coefficienti incriminati con l'algoritmo successivo, che quindi è a tutti gli effetti una dimostrazione. Anzi. E' *la* dimostrazione. Rispetto a dimostrazioni per assurdo o per induzione essa produce anche il valore cercato, e come tale viene considerata migliore delle altre da tutti i matematici.

---

[12] Questa dimostrazione mi piace di più e mi sembra molto più diretta. Inoltre spiana la strada all'algoritmo esteso di Euclide.

[13] Infatti scriviamo , come visto in 2.2.1. E dovendo valere per qualsiasi divisore, deve valere anche per il più grande!

[14] almeno, ne esistono infiniti

2.9. Algoritmo di Euclide esteso e coefficienti di Bezout

https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Questo algoritmo oltre a calcolare il MCD fra due numeri, trova anche i coefficienti di Bezout[15]. In particolare quindi se l'MCD vale 1, si avrà

$$ax + by = 1$$

che implica che

$$ax\%b \equiv 1\%b$$

Cioè x è l'inverso di a nel gruppo moltiplicativo modulo b [16].

Si prenda in considerazione la tabella che segue. Le prime due righe sono compilate di autorità con i valori assegnati.

Si verifichi che su ogni riga i vale la relazione $ax_i + by_i = r_i$[17]

TABELLA 2. Algoritmo esteso di Euclide

i passo	q quoziente	r resto	x	y
0	—	a	1	0
1	—	b	0	1
2	$a//b=$ $r_0//r_1$	$a\%b=$ $r_0\%r_1=$ $r_0 - q_2 r_1$	1	$-q_2$
...				
i	$r_{i-2}//r_{i-1}$	$r_{i-2}\%r_{i-1}=$ $r_{i-2} - q_i r_{i-1}$	$x_{i-2} - q_i x_{i-1}$	$y_{i-2} - q_i y_{i-1}$
...				
n-1	$r_{n-3}//r_{i-2}$	$r_{i-3}\%r_{i-2}=$ $r_{i-3} - q_{i-1} r_{i-2} = MCD$	$x_{i-3} - q_{i-1} x_{i-2}$	$y_{i-3} - q_{i-1} y_{i-2}$
n	$r_{n-2}//r_{n-1}$	$r_{n-2}\%r_{n-1}$=0		

Si osservi come il calcolo alternativo di r abbia la stessa forma del calcolo delle successioni degli x e y. Però il quoziente resta lo stesso su tutta la riga, quindi non si può usare il % fra i valori omogenei delle ultime due colonne. La dimostrazione è il modo migliore di capire perché funziona!

ESERCIZIO. Si scriva un programma che presi in input due valori (saranno a,b) trovi:

1) Se a,b sono coprimi, ovvero il loro MCD

2) Se $MCD(a, b) == 1$ allora si trovi con l'algoritmo esteso di Euclide l'inverso di a modulo b.

SOLUZIONE.

```
# Compute the extended euclidean algorithm, returns the GCD and the
def ext_euclid(a, b):
    a, b = abs(a), abs(b)
    s0, s1, t0, t1 = 1, 0, 0, 1
```

[15]Ammesso che esistano. Perché?

[16]Anche se lo vedremo nel seguito è intuitivo che se due numeri moltiplicati insieme danno come risultato 1, anche se modulo qualcosa, si comportano un po' come degli inversi per la moltiplicazione usuale.

[17]Anche qui si proceda per induzione come per la dimostrazione alternativa dell'algortimo di Euclide. E' evidente che vale per le righe 0 e 1. La si dimostri per la seconda e quindi per una generica i.

```
    while b != 0:
        q, a, b = a // b, b, a % b
        s0, s1 = s1, s0 - q * s1
        t0, t1 = t1, t0 - q * t1
    return a, s0, t0

def ext_euc_std(a,b):
  a, b = abs(a), abs(b)
  a, b = max(a,b), min(a,b)
  r, q, s, t = [a, b], ['nan'], [1, 0], [0, 1]
  i = 1
  while r[-1]>0:
    r.append( r[i-1] % r[i] )
    q.append( r[i-1] // r[i] )  #q has got one less element than other lists

    s.append( s[i-1]- q[i]*s[i] )
    t.append( t[i-1]- q[i]*t[i] )
    print(f'r = {r[-1]}\tq = {q[-1]}\ts = {s[-1]}\tt = {t[-1]}\t')
    i = i+1
  return r[-2], s[-2], t[-2]

def main():
    print("——— Extended Euclidean Algorithm ———\n")

    # Get user input
    while True:
        try:
            a = int(input("Insert the first integer: "))
            b = int(input("Insert the second integer: "))
            break
        except ValueError:
            print("\nYou must enter two integer.\n")

    # Execute the extended euclidean algorithm and print the results
    gcd, x, y = ext_euclid(a, b)
    print("\nGreatest Common Divisor (GCD):", gcd)
    print("Bezout Coefficients (x, y):", "(", x, ",", y, ")","\n")
    gcd, x, y = ext_euc_std(a, b)
    print("\nGreatest Common Divisor (GCD):", gcd)
    print("Bezout Coefficients (x, y):", "(", x, ",", y, ")")


if __name__ == '__main__':
    main()
```

CAPITOLO 3

I gruppi dell'algebra astratta. Le basi

Le nozioni che seguono sono un modo per dimostrare il Piccolo Teorema di Fermat, cardinde della dimostrazione del funzionamento dell'RSA.

3.1. Definizione di gruppo

Qui e nel prosieguo faremo riferimento a un insieme G, finito o no che sia. Esso è composto da vari elementi. Fra questi elementi definiamo una certa operazione binaria, ovvero un'applicazione che presi come argomenti due elementi dell'insieme G, restituisca un elemento ancora di G. Chiameremo questa operazione "+"[1], anche se non vorremo nè *dovremmo* confonderla con la somma ordinaria.

$$\forall a, b \in G, \exists c \in G : c = a + b$$

Nel linguaggio delle applicazioni, o funzioni, dovremmo scrivere $c = +(a, b)$[2] dove la funzione "+" ha per argomenti i valori rappresentati dalle variabili a e b e restituisce il valore c. Va da sé che il valore di c è unico, ovvero ogni volta che facciamo $a + b$ otteniamo sempre e solo un certo c [3].

La propietà summonenzionata viene detta di **chiusura**. Ovvero, computando attraverso "+" una qualsiasi coppia di elementi di G, si ottiene sempre un elemento di G; non si può "scappare" da G attraverso "+".

E' forse meno ovvio che non è detto che $a + b = b + a$. Questo è vero per l'usuale somma, ma se intendiamo una funzione con due argomenti, diventa più spontaneo immaginare che lo scambio degli argomenti possa portare in generale a un valore diverso[4]. Per esempio[5] si pensi all'elevamento a potenza, dove a può essere l'esponente e b la base o viceversa. Ovviamente lo scambio della base con l'esponente in genere porta ad un risultato diverso.

Più semplicemente basta considerare la sottrazione dove $a - b = b - a$ è vera se e solo se $a = b$.

Una seconda importante proprietà di questa operazione è l'associatività:

$$\forall a, b, c \in G : (a + b) + c = a + (b + c)$$

dove il significato delle parentesi è quello naturale di regolare la precedenza nell'applicazione dell'operazione. Vediamo come diventerebbe in forma di funzione:[6]

---

[1] In realtà, almeno per gruppi astratti, potrebbe essere più sagace usare l'operazione "-" che anche nell'aritmetica ordinaria non è commutativa. Solo si perderebbe la capacità di definire il monoide $(\mathbb{N}, +)$.

[2] Questo modo di rappresentare l'operazione è molto informatica: è esattamente come la scriveremmo come una funzione in quasi tutti i linguaggi.

[3] Va da sè che se esistesse un d diverso da c tale che $d = +(a, b) = c$, per la transitività dell'uguaglianza $d = c$ contro l'ipotesti che siano distinti.

[4] Per esempio i generale $f(a, b) \neq f(b, a)$

[5] L'esempio è macchinoso, la sottrazione si presterebbe decisamente meglio!!!

[6] Da notare che siccome +() produce come risultato un elemento di G allora è perfettamente lecito che +() possa essere considerato come un argomento di un'altra operazione +().

$$\forall a, b, c \in G : +(\ +(a, b)\ , c) = +(a,\ +(b, c)\ )$$

ovvero anche eseguendo l'ordine delle operazioni diversamente, il risultato non cambia.

Ogni insieme dotato di operazione che gode delle proprietà enunciate sopra, **chiusura ed associatività**, è detto *semigruppo*.

Se inoltre in G, semigruppo, esiste un elemento, che chiameremo e, tale che

$$\forall a \in G, a + e = e + a = a$$

chiameremo e elemento neutro di "+" in G, e diremo che $(G, +)$ è un *monoide*.

Se in più

$$\forall a \in G, \exists b \in G : a + b = b + a = e$$

diremo che b è *elemento inverso* di a e in generale lo indicheremo con $\overline{a}$. Se ogni elemento di G è invertibile, ovvero ammette elemento inverso, diremo che (G,+) ha la struttura di un *gruppo*.

Riassumendo:

(1) Un insieme G dotato di un'operazione, detta "+", chiusa e associativa si dirà che è un (una struttura di) *semigruppo*.
(2) Se (G,+) è semigruppo ed ha elemento neutro, diremo che (G,+) ha la struttura di un *monoide*.
(3) Se (G,+) è un monoide e ogni elemento di G ammette inverso, diremo che (G,+) ha la struttura di un *gruppo*.
(4) Se (G,+) è un gruppo e l'operazione "+" è commutativa, diremo che (G,+) ha la struttura di un gruppo commutativo (o abeliano).

Il numero di elementi dell'insieme G, spesso indicato come *supporto* della struttura algebrica in esame, porta poi ad un'altra distinzione: gruppi finiti e gruppi non finiti.

3.2. Esempi

Esempio 6. L'usuale operazione di somma e l'insieme dei numeri naturali $\mathbb{N}$. Consideriamo quindi la struttura $(\mathbb{N}, +)$.

La somma di due numeri naturali è sempre un numero naturale. Come ben sappiamo fin dalle elementari, l'ordine in cui si sommano gli addendi non cambia il risultato finale, e quindi la somma ordinaria è associativa. Il numero 0 è il nostro elemento neutro, dato che aggiunto a qualsiasi numero non cambia il numero stesso.

Per l'inverso abbiamo qualche problema: il numero da sommare a 4 per farlo diventare 0 è -4, e non è un naturale.

Quindi $(\mathbb{N}, +)$ è un monoide, ma non un gruppo.

Se considerassimo l'insieme dei numeri interi (ovvero $\mathbb{Z}$) allora avremmo sì un gruppo. Gruppo fra l'altro commutativo e infinito.

Esempio 7. Dell'orologio. Immaginiamo di avere un orologio da polso analogico, ovvero con le lancette. Sul quadrante compaiono i numeri dall'1 al 12. Se ora sono le ore 4, fra 5 ore saranno le ore 9. Se passano altre 4 ore saranno le ore 1. La somma delle ore su un orologio è quindi un'operazione chiusa e associativa, come nell'esempio precedente. L'elemento neutro è il valore 12 (ovvero un giro completo delle lancette). Vediamo se riusciamo a trovare gli elementi inversi.

L'inverso di un'ora è il numero di ore da sommargli per arrivare alle ore 12. Quindi l'inverso di 1 è 11, di 2 è 10, di 3 è 9 e così via. La cosa più buffa di questo gruppo è che l'inverso di 12 è 12 (che in realtà non è per nulla strano, essendo 12 l'elemento neutro *allora* deve essere inverso di sè stesso come vedremo nel prossimo paragrafo). L'operazione, come sopra, è pure commutativa.

Abbiamo quindi che l'insieme $G = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ con la somma sul quadrante dell'orologio è un gruppo abeliano. Ed è finito, come si vede.

Fin qui abbiamo visto solo gruppi, o monoidi, commutativi, cosa che potrebbe portarci a credere che la commutatività sia un qualcosa di scontato. Un bell'esempio di gruppi non commutativi potrebbero essere il gruppo delle permutazioni, il gruppo delle matrici quadrate con la moltiplicazione. Un altro bell'esempio più facile da immaginare è quello degli itinerari: immaginatevi a New York con tutte le strade che si incrociano ad angolo retto. Immaginate un itinerario del tipo "svolta a destra e poi a sinistra". Immaginiamo di invertire l'ordine delle svolte e come si vede dal disegno il punto che si raggiunge non è esattamente lo stesso.

Esempio 8. Del gruppo moltiplicativo modulo p

Sia p un numero primo e $G = \{x \in \mathbb{N} : x > 0, x < p\}$. Sia * l'operazione tale che $a * b = (ab)\%p$. Provare che G è composto di p-1 elementi. 1 è il suo elemento neutro. $\forall g \in G, \exists \overline{g} \in G : g * \overline{g} = 1$, ovvero ogni elemento è invertibile. Concludere che questa struttura algebrica è un gruppo (abeliano).

Esercizio. Provare a trovare un gruppo come nel caso precedente, ma con p non necessariamente numero primo. Quali elementi compresi fra 0 e p-1 vanno esclusi? E perché? Quanti (e quali) ne restano?

3.3. Lemmi

Per lemma intendiamo una sorta di teorema, ovvero una verità dimostrata con un procedimento logico deduttivo a partire dai nostri assiomi. La lieve differenza con un teorema è che il lemma viene enunciato e poi dimostrato, all'opposto dei teoremi. Spesso però si tratta solo di quisquiglie e differenze marginali. In realtà quello che indichiamo come teoremi sono spesso delle verità logicamente dedotte ma di interesse primario. I lemmi invece sono, spesso, usati a supporto delle dimostrazioni stesse e in qualche misura più banali, fors'anche ovvi, dei teoremi. Ciò nonostante qui ne elencheremo e dimostreremo alcuni dei più fondamentali nella teoria dei gruppi.

Lemma 9. *L'elemento neutro di un gruppo è unico*

Dimostrazione. Consideriamo $(G, +)$. Per definizione, e è un elemento neutro, ma non è stato assunto che sia anche l'unico. Immaginiamo quindi che ne esista almeno un secondo che chiameremo f **distinto** da e. Essendo entrambi elementi neutri possiamo scrivere che $e + f = f$ e anche $e + f = e$. Per quanto detto nel paragrafo 1, questo porta a concludere $e = f$ contro l'ipotesi iniziale. Avendo ottenuto un assurdo significa che l'ipotesi è falsa e che quindi non esiste un secondo elemento con le stesse proprietà di e che quindi è unico in $(G, +)$.

Riassumendo

$$\exists e, f : e \neq f : \forall a \in G \implies e{+}a = a{+}e = f{+}a = a{+}f = a \implies e = e{+}f = f \implies !!$$

□

Quanto appena detto è vero per un generico gruppo. Ovvero vale per tutti i gruppi.

Lemma 10. *Leggi di cancellazione. Dato un* ***gruppo*** *(G,+) e tre elementi, diciamoli a, x, y appartenenti a G, se $a + x = a + y$ allora $x = y$, e viceversa, se $x + a = y + a \implies x = y$.*

Dimostrazione. Nel gruppo (G,+) esiste l'elemento neutro e. Ovviamente esiste anche un elemento b inverso di a, ovvero tale che $b + a = e$. Quindi $x =$

$e + x = (b + a) + x = b + (a + x) = b + (a + y) = (b + a) + y = e + y = y$, che conclude la dimostrazione. □

Ovvero si può cancellare a dalle equazioni. Attenzione però. L'elemento uguale si può cancellare solo se si presenta sullo stesso lato dell'operazione, oppure se l'operazione è commutativa.

Si noti che in questa dimostrazione abbiamo usato l'ovvio assioma *cose* logicamente equivalenti (uguali) possono essere interscambiate senza alterare il valore di un'espressione.

LEMMA 11. *Unicità dell'inverso*

$$\exists! b \in G : b + a = a + b = e$$

DIMOSTRAZIONE. Come prima neghiamo la tesi e vediamo dove ci conduce, ovvero dato $a \in G$ esistano $b, c \in G$ entrambi inversi di a e tali che $b \neq c$. Per le proprietà dell'elemento inverso, possiamo scrivere $e = a + b$ ed $e = a + c$ da cui $a + b = a + c$. Ma per il lemma precedente siamo quindi obbligati a concludere che $b = c$ contro l'ipotesi iniziale, che quindi è falsa. Quod Erat Demostrandum, l'unica alternativa è che non esistano due elementi distinti inversi di uno stesso elemento, ovvero che l'elemento inverso sia quindi unico. □

LEMMA 12. *L'inverso dell'inverso è l'elemento stesso*

$$\forall a \in G : \overline{(\overline{a})} = a$$

DIMOSTRAZIONE. Chiariamo cosa vogliamo dimostrare: $\forall a \in G : \overline{(\overline{a})} = a$. Definiamo allora $b = \overline{a}$ e calcoliamone l'elemento inverso. $\overline{b} : \overline{b} + b = e$ è equivalente a scrivere $\overline{b} + \overline{a} = e$. Ora aggiungiamo a a destra di entrambe le espressioni ottenendo $\overline{b} + \overline{a} + a = e + a \iff \overline{b} + e = a \iff \overline{b} = a$. Riscrivendo $b = \overline{a}$ si ottiene $\overline{(\overline{a})} = a$. □

LEMMA 13. $\overline{(a + b)} = \overline{b} + \overline{a}$.

DIMOSTRAZIONE. Basta provare che $\overline{b} + \overline{a}$ è l'inverso di $a + b$. Vediamo subito che

$$(\overline{b} + \overline{a}) + (a + b) = \overline{b} + (\overline{a} + (a + b)) = \overline{b} + (\overline{a} + a) + b = \overline{b} + e + b = \overline{b} + b = e$$

□

D'ora in avanti parleremo sempre di gruppi, e per non appesantire la notazione scriveremo spesso G intendo con esso la struttura (G,+) che è gruppo. L'abuso di linguaggio sarà comunque sempre accompagnato dalle parole, ed evitato quando il contesto non sia immediatamente esplicito.

3.3.1. Ordine di gruppi. Un caso in cui si può "confondere" G con (G,+) è quando parliamo dell'ordine di un gruppo. Esso non è altro che la cardinalità[7] del supporto del gruppo G. Va da sè che in generale ha senso parlare della cardinalità di un insieme quando essa è finita, e così anche per l'ordine di un gruppo. Arriviamo quindi alla seguente definizione

DEFINIZIONE. L'ordine di un gruppo *finito* (G,+), che indicheremo semplicemente con o(G), invece di o((G,+)), è la cardinalità dell'insieme G a supporto del gruppo.

[7]Ricordiamo che per cardinalità intendiamo il numero di elementi, se finito, di un insieme

3.3.2. Gruppi ciclici. Di tutti i gruppi possibili ce n'è una categoria particolare che va sotto il nome di gruppo ciclico. Questi gruppi particolari, *finiti*, hanno almeno un elemento che è in grado, a furia di essere computato da solo, di generare tutti gli altri. L'esempio più semplice è quello del gruppo delle ore visto prima. Se ci si pensa le ore 1, per somme successive, possono generare tutte le altre ore. Non bisogna però credere che questa proprietà sia naturale del numero 1. Infatti il numero 1 non è un generatore del gruppo $(\mathbb{Z}, +)$. Come si verifica facilmente, ogni numero positivo, cioè *naturale*, si può scomporre come somma *finita* di unità. Ma non c'è modo di ottenere un numero *negativo* come somma di numeri positivi.

Per i gruppi finiti invece le cose non vanno esattamente così.

PROBLEMA 14. Trovare tutti i generatori del gruppo delle ore dell'orologio.

Enunciamo quindi un classico teorema di teoria dei numeri:

TEOREMA 15. *Un gruppo finito con p elementi dove p è un numero primo allora è un gruppo ciclico.*

Da notare che il viceversa non è vero. Dato un elemento si può costruire un gruppo di ordine qualsiasi che ammette quell'elemento come generatore.

Inoltre si può facilmente costruire un gruppo di ordine 4 che non sia ciclico. Si pensi al gruppo G che abbia come supporto l'insieme di 4 elementi $G = \{e, a, b, c, \}$ dove e è l'elemento neutro. Posto che $a+a = b+b = c+c = e$ e posto, per esempio, $a + b = c$; $a + c = b$; $b + c = a$ è il gruppo, commutativo, cercato.

La dimostrazione al momento si omette. Sarà poi vista come un corollario dell'importantissimo *teorema di Lagarange* per i gruppi finiti.

3.4. Sottogruppi

DEFINIZIONE. Un sottogruppo di un gruppo, i.e. (G,+), è un sottoinsieme di G che è comunque gruppo rispetto alla medesima operazione + definitia in (G,+).

Sia quindi $A \subset G$ tale che A sia sottogruppo di G. Scriveremo d'ora in poi che $A \in sg(G, +)$ dove per $sg(G, +)$ intendiamo l'insieme di tutti i possibili sottogruppi di (G,+).

FATTO. *Ogni gruppo ha sempre almeno due sottogruppi (che chiameremo* ***banali****): sé stesso e il gruppo che contiene solo l'elemento neutro.*

3.4.1. relazione di congruenza per sottogruppi. In questo paragrafo indicheremo sempre con $H \in sg(\ G, +)$ un sottogruppo di (G,+).

Si prenda un elemento $a \in G$ e si cerchino gli elementi $b \in G$ che soddisfino la proprietà che $a + \overline{b} \in H$. Questo insieme, d'ora in avanti lo chiameremo *classe di congruenza modulo H*, degli elementi $b \in G$ che rispettano la condizione di pocanzi gode di alcune interessanti proprietà:

DEFINIZIONE. La classe di congruenza di $a \in G$ modulo H viene definita come l'insieme degli elementi $b \in G$ tali che $a + \overline{b} \in H$ e scriveremo che $a \equiv b \mod H = \{b \in G : a + \overline{b} \in H\}$.

La classe di congruenza è quindi un sottoinsieme di G. Elenchiamo alcune ovvie proprietà:

(1) $a \equiv a \mod H$. Per assurdo, come potrebbe essere il contrario?
(2) Se $a \equiv b \mod H \iff b \equiv a \mod H$.
(3) Se $a \equiv b \mod H$ e $b \equiv c \mod H$ allora $a \equiv c \mod H$.
(4) Se $a \neq b \mod H \iff b \neq a \mod H$.