Autore: Davide Zhang Indirizzo E-mail: davide.zhang@stud.unifi.it Numero matricola: 7073051

Data consegna: 31/05/2022

Introduzione

Per realizzare le richieste del progetto ho implementato separatamente i vari algoritmi di cifratura e poi li ho messi insieme alla fine. Quindi la prima parte della relazione riguarderà l'implementazione dei singoli metodi di cifratura e le corrispondenti metodi per decifrare, e l'ultima parte riguarderà invece come mettere insieme tutti i metodi per arrivare all'algoritmo finale.

Cifrario a sostituzione

Nel Cifrario di Cesare per ottenere il messaggio crittografato ho bisogno di "traslare" ogni singola lettera del messaggio di una serie di posti a destra o a sinistra a seconda del valore rappresentato dalla costante "SostK": quindi come argomenti di questo primo metodo avrò bisogno di una costante e una stringa da trasformare, la "plaintext".

Il primo passo che ho fatto è quella di caricare dalla memoria il carattere e determinare se è una lettera o no, dopo di ciò ho applicato la cifratura a sostituzione in base al tipo di carattere. All'interno della parte di codice che si occupa della cifratura dei caratteri viene anche svolto il calcolo del modulo, implementato sia per il caso di caratteri maiuscoli che per il caso di caratteri minuscoli con nomi diversi per le etichette. In questo metodo non viene quindi usato la pila.

Per decodificare un messaggio cifrato con sostituzione devo solo semplicemente invertire la "traslazione" delle lettere: a partire da "sostK" devo calcolarne l'opposto e chiamare il metodo del cifrario a sostituzione con questo come l'argomento. Il metodo del calcolo dell'opposto della costante viene implementato in questo modo: visto che i numeri interi in RISC-V sono codificati con complemento a due, per ottenere l'opposto di un dato numero devo solo invertire i bit e aggiungere 1. L'inversione è ottenuta con una semplice istruzione di xor tra il dato contenuto in a0 e -1 in t0 (-1 in complemento a due è rappresentato come 2^n-1, quindi i bit del registro t0 saranno costituito da soli 1).

Cifrario a blocchi

Il cifrario a blocchi è simile al cifrario a sostituzione solo che, a differenza di essa, non ho una sola costante di traslazione ma una serie di costanti che vengono forniti sotto forma di una stringa di caratteri: ad ognuno di questi caratteri corrisponde un numero che verrà usato per cifrare ciascun carattere del testo in chiaro. Una volta che si finiscono i caratteri della stringa fornita con la variabile "blocKey" si ricomincia da capo e quindi si seleziona di nuovo il primo carattere della suddetta stringa. Il processo non termina finché non termina la stringa da codificare.

Per implementare questa procedura ho prima costruito un metodo a parte che mi scorre la stringa "blocKey" e restituisce, data in input l'indice del carattere da caricare e l'indirizzo della stringa, il carattere che mi serve per la codifica e l'indice aggiornato all'elemento successivo (nel caso "blocKey" finisse il metodo restituirà il primo carattere della stringa e imposterà l'indice a 1 (prossimo elemento da selezionare)). Successivamente ho costruito il corpo del metodo che consiste in un ciclo che carica per ogni iterazione un carattere del testo in chiaro, verificando anche che sia diversa da zero (termine stringa) e sommandola con il carattere caricato dalla chiave. Il risultato è poi diminuito modulo 96 prima di sommarla di nuovo con 32 e salvarla in memoria al posto del carattere originale.

Il metodo prende come argomenti l'indirizzo del testo in chiaro e quello della chiave, che vengono subito salvati nella pila per usi successivi insieme agli indici di scorrimento. Il carattere caricato dalla "plaintext" viene temporaneamente salvato in pila prima di chiamare il metodo "metodoScorrimentoRipetuto" che restituisce la codifica di un key. Questi due, come detto prima, vengono poi sommati insieme mentre si salva l'indice di "blocKey" aggiornato nella pila. Quindi si calcola il modulo della somma attraverso un loop, si somma il risultato con 32, si scarica il risultato della cifratura in memoria usando l'indirizzo recuperato dalla pila, si aggiorna anche l'indice del testo in chiaro e, infine, si ripristina in a0 e in a1 gli indirizzi delle variabili in memoria prima di entrare in una nuova iterazione.

La procedura finisce quando si carica dalla memoria un byte nullo: in tal caso si salta all'etichetta fine_cifraturaABlocchi dove si conclude la procedura ripulendo la pila, e restituendo l'indirizzo del testo crittografato.

L'implementazione della procedura inversa è simile: tuttavia bisogna trovare prima un modo per risalire alla codifica iniziale del carattere nel testo in chiaro data quella del carattere nella chiave. L'algoritmo che ho pensato io parte dal

carattere fornito dalla “cypthertext”: si sottrae 32 e la codifica del carattere della chiave prima di aggiungere una serie di 96 finché il numero risultante non sia maggiore o uguale a 32.

Si caricano i caratteri della “plaintext” e della chiave in memoria, si verifica che il carattere del testo in chiaro e quello proveniente dalla “blocKey” non siano di fine stringa, si aggiornano gli indici nella pila e infine si applica il processo di decifratura descritto prima. Se il byte caricato dal testo in chiaro è nullo il metodo termina restituendo l’indirizzo della stringa decifrata, mentre se è nullo il byte da “blocKey” si entra nel blocco di istruzioni “gestisci_fineblocco” che carica il primo carattere del blocco e imposta l’indice di “blocKey” a 1. Prima entrare in una nuova iterazione, dopo aver memorizzato il carattere decifrato in memoria, devo impostare in a0 e in a1 gli indirizzi delle stringhe in maniera che siano già pronte all’uso.

Cifratura occorrenze

Per cifrare un messaggio in questo modo devo trovare le posizioni dei vari caratteri all’interno della stringa, convertirli in caratteri ASCII e inserirli in memoria come tali, elencandoli una dopo l’altra separati da un trattino nella lista di occorrenze del carattere giusto (ogni tipo di carattere che appare nella stringa ha una sua lista di occorrenza). Il primo metodo che ho implementato è quella che converte i numeri interi in cifre ASCII: questo è fatto trovando il resto della divisione per dieci del numero e aggiungendo ad esso 48, mentre il quoziente è riciclato nel ciclo successivo per individuare la cifra che segue. Questo procedimento è ripetuto finché il numero è maggiore di dieci: durante ogni iterazione del ciclo i caratteri ASCII trovati vengono memorizzati nella pila, in modo tale che alla fine della procedura posso recuperare uno alla volta le cifre del numero e inserirle nell’indirizzo indicato negli argomenti del metodo. Per segnalare la fine dei caratteri ho inserito all’inizio della pila uno zero: al momento dell’estrazione questo porterà al termine il processo.

La cifratura ad occorrenze da me implementato è composta da due cicli: uno interno ed uno esterno. Quello esterno seleziona i caratteri della stringa uno alla volta, controllando se sono già stati registrati, mentre quello interno fa la maggior parte del lavoro costruendo le varie liste di occorrenza dei caratteri e chiamando nel processo la procedura di conversione degli interi. Il ciclo esterno è diviso in due parti: la parte iniziale costruisce la lista delle occorrenze del primo carattere della stringa, mentre l’altra parte è costituita da un ciclo che scorre la “plaintext” individuando i caratteri diversi dalla prima e costruendo per ognuna di essi la loro lista di occorrenza attraverso il ciclo interno (il primo carattere della stringa è usato per segnalare che il carattere è già stato esaminato prima). Il ciclo interno per essere chiamato necessita direttamente dell’indirizzo in memoria in cui si è arrivati nella costruzione della “ciphertext”, e quindi prima di chiamarlo si deve trovare quel valore sommando l’indice di scorrimento della stringa in costruzione con l’indirizzo della “ciphertext”. Alla fine del ciclo interno recupero l’indice di scorrimento e lo uso per aggiornare quello del ciclo esterno sommandolo ad esso. Un altro compito importante svolto dal ciclo interno è la modifica dei caratteri della “plaintext”: ogni volta che si costruisce una nuova lista di occorrenze si modificano i caratteri del testo in chiaro in maniera tale che ai cicli esterni successivi i caratteri già considerati (cioè uguali al primo) possano essere ignorati. Alla fine del ciclo interno si aggiunge uno spazio alla “ciphertext” per separare le varie liste di occorrenza che si costruiranno. L’ultima operazione che si fa per concludere la procedura della cifratura ad occorrenze è l’inserimento del carattere di fine stringa.

Anche la procedura per decifrare è costituita da due cicli: uno principale col compito di individuare le varie liste di occorrenza attraverso gli spazi vuoti, e uno interno per inserire i caratteri nelle posizioni indicate. Alla fine del processo, contando i numeri di inserimenti che ho fatto, e a patto che la scrittura crittografata abbia senso posso calcolare dall’indirizzo della “ciphertext” e dal numero di inserimenti fatti la posizione di inserimento del carattere di fine stringa. Per convertire le cifre ASCII in numeri ho costruito una procedura asservita che carica i byte dei caratteri uno alla volta, sottraendo 48, e poi sommandoli alla cifra successiva sottoposta alla stessa trasformazione, dopo che il primo è stato moltiplicato per 10. Questo procedimento si ripete finché non incontro un carattere del trattino o dello spazio. Dopo che ottengo la posizione del carattere sottraggo questo di 1 e lo uso per il suo inserimento nel testo decifrato: questo ciclo continua finché non incontro uno spazio vuoto, dopodiché ritorno al corpo del metodo dove carico il carattere successivo da inserire che quindi chiama di nuovo il ciclo interno.

Il metodo inserisce automaticamente il testo decifrato 1000 byte prima della “ciphertext”, come argomenti il ciclo di inserimenti (ciclo interno) ha l’indice di scorrimento, l’indirizzo della “ciphertext” e il numero inserimenti. Il carattere stesso è invece caricato nel ciclo stesso, prima di saltare alla posizione del primo numero. La pila, sia nel processo di

cifratura che in quella di decifratura, è stata usata per memorizzare i dati prima di chiamare procedure asservite e indirizzi di ritorno in casi di necessità.

Dizionario

prima di tutto provo che per effettuare le trasformazioni richieste per i numeri devo sottrarre a 105 la codifica del numero, mentre per le lettere, sia maiuscole che minuscole, devo sottrarre a 187 la codifica del carattere.

Per i caratteri maiuscoli:

1. Posizione del carattere nell'alfabeto: cod(ci)-65
2. Sottraggo questo numero a 122 (codifica dell'ultimo carattere minuscolo): 122-cod(ci)+65=187-cod(ci)

per i caratteri minuscoli:

1. Posizione del carattere nell'alfabeto: cod(ci)-97
2. Sottraggo questo numero a 90 (codifica dell'ultimo carattere maiuscolo): 90+97-cod(ci)=187-cod(ci)

Per le cifre:

1. Devo sottrarre alla codifica di 9 la codifica del numero ci: 57-cod(ci)
2. Aggiungo 48 per ottenere la codifica ASCII del numero: 57+48-cod(ci)=105-cod(ci)

La funzione implementata carica prima di tutto il carattere da analizzare e attraverso una serie di confronti decide se è un numero o una lettera: se è un numero gli viene applicato la trasformazione dei numeri, se è una lettera quella delle lettere e nulla se non è nessuna dei due. Dopo ciascun tipo di trasformazione viene aggiornato l'indice prima di tornare alla testa del codice per pescare il carattere successivo. L'algoritmo termina quando si incontra la fine della stringa.

La funzione di decifratura è del tutto uguale a quella di cifratura.

Inversione

L'inversione è implementata scorrendo prima la "plaintext" fino a incontrare il byte nullo e poi da lì procedere indietro, copiando i caratteri della "plaintext" nell'indirizzo della "ciphertext" fino ad arrivare ad un indice negativo. Dopodiché fermarsi e inserire il byte nullo nella cipher. L'algoritmo di decodifica è lo stesso di quella di cifratura.

Corpo codice

Dopo aver caricato gli argomenti del metodo nei registri a, ho iniziato a scorrere "mycipher", determinando quale cifratura effettuare in base ad una serie di confronti del byte ASCII con il reg t4, dove ho messo di volta in volta i caratteri corrispondenti al segnale di inizio di vari algoritmi di cifratura. Per ognuna di questi algoritmi c'è una parte di codice precedente alla chiamata vera e propria che salva i dati del metodo principale nella pila e immette nei registri a gli input giusti per quel determinato tipo di algoritmo. A questo punto vengono chiamati i vari tipi di algoritmi che o modificano direttamente la "myplain" o creano una nuova stringa in un indirizzo diverso per il testo crittografato: se siamo nel primo caso non c'è da fare nulla; se siamo nel secondo, prima di tornare al metodo con l'indirizzo di ritorno recuperato dalla pila devo aggiornare la locazione di memoria per la "myplain". Fatto il passo di cifratura salto all'etichetta "passoCifraturaFatto" dove recupero i dati dalla pila e aggiorno l'indice di scorrimento per la "mycipher". A questo punto torno al caricamento dei caratteri dalla "mycipher", che termina quando carico il byte nullo: inizia allora la fase di decodifica. Per fare in modo che "myplain" possa essere stampato alla fine di ogni ciclo di cifratura in maniera separata ho implementato un metodo di stampa a parte: questa copia la stringa da stampare in un indirizzo diverso di memoria, ci aggiunge alla fine il carattere della riga nuova e poi di fine stringa prima di stamparla. Il metodo stampa è quindi sempre chiamato alla fine di ogni algoritmo di cifratura.

La parte del codice dedicata alla decifratura ha una struttura quasi simmetrica alla prima parte: una tra le differenze si trova nei primi passi dove effettuo una inversione senza stampa della mycipher e il calcolo dell'opposto per il sostK. Quindi entro nel ciclo di decifratura, determino quale processo effettuare in base al valore del carattere caricato da "mycipher", memorizzo i dati importanti e applico quel processo alla "myplain". Gli algoritmi di decifratura sono tutti

separati da quelli di cifratura anche se alcuni tra loro possono essere uguali; comunque il processo della chiamata è lo stesso: si memorizza nella pila l'indirizzo di ritorno, si svolge il codice corrispondente al metodo di decifratura, si trasferisce l'indirizzo della "myplain" in a0 (si salva in pila questo valore se è diverso da quello iniziale) e si chiama il metodo della stampadec (del tutto uguale al metodo della stampa) prima di recuperare in ra l'indirizzo di ritorno dalla pila e chiudere. La decifratura finisce quando carico dalla "mycipher" un byte nullo.