Progetto Assembly RISC-V per il Corso di Architetture degli Elaboratori – A.A. 2019/2020 –

Messaggi Cifrati

Versione 3 del documento, aggiornata il 8/4/2020. Modifiche rispetto alle precedenti versioni:

- Condivisione su MOODLE
- Chiarimenti (input attesi cifrature, modalità di esame, dimesione plaintext, cifratura occorrenze, newline)
- Modifica testo cifrature in modo da lavorare solo con caratteri c tali che $32 \leq cod(c) \leq 127$

Crittografia e Cifratura

I codici sono un modo per alterare un messaggio testuale per nasconderne il significato originale. Questo permette di crittografare stringhe e dati generici, per nasconderne il significato a terzi che intercettano il messaggio cifrato, e che non hanno la chiave per decodificarlo. Uno degli esempi più famosi è quello della *macchina Enigma* (suggerisco la visione di The Imitation Game, per chi non l'avesse visto!), un dispositivo elettromeccanico per cifrare e decifrare messaggi che fu ampiamente utilizzata dal servizio delle forze armate tedesche durante il periodo nazista e della seconda guerra mondiale. In molti casi e di solito richiedono una **parola chiave** per essere interpretati. I cifrari sono algoritmi applicati a un messaggio che nascondono o criptano le informazioni trasmesse. Questi cifrari vengono quindi **invertiti** per tradurre o decifrare il messaggio.

Macchina Enigma versione G

Ad esempio, il testo in chiaro (plain text) AMO AssEMbLY potrebbe essere modificato tramite una funzione di cifratura che sostituisce ogni lettera con la successiva nell'alfabeto, ottenendo la parola criptata (cyphertext) BNP BttFNcMZ. Solo chi conosce una apposita funzione di de-cifratura riesce ad interpretare il testo. In questo caso, la funzione di de-cifratura sostituisce ciascuna lettera con quella che la precede nell'alfabeto, riottenendo AMO AssEMbLY.

Più rigorosamente, dato un plaintext pt, una funzione di cifratura fc, una funzione di decifratura fd, si ha

$$\text{cyphertext } ct = fc(pt), \qquad pt = fd(ct)$$

In generale, utilizzando una serie di n funzioni di cifratura $FC = \{fc_1, ..., fc_n\}$ e di decifratura $FD = \{fd_1, ..., fd_n\}$, dove *ogni fd_i decifra il messaggio critpato dalla funzione fc_i*, si ha che

$$pt = fd_n(fd_{n-1}(...fd_1(\, fc_1(fc_2(...fc_n(pt))))))$$

Ovvero che applicando **in sequenza le funzioni di cifratura** $fc_n, fc_{n-1}, ... fc_1$, si riottiene il plaintext applicando le **funzioni di decifratura in ordine inverso** $fd_1, fd_2, ... fd_n$

Alcuni Codici di Cifratura

Cifrario a Sostitizione / Cifrario di Cesare

(da Wikipedia) Il cifrario di Cesare è uno dei più antichi algoritmi crittografici di cui si abbia traccia storica. È un cifrario a sostituzione monoalfabetica in cui **ogni lettera del testo in chiaro è sostituita nel testo cifrato dalla lettera che si trova un certo numero di posizioni dopo nell'alfabeto**. Questi tipi di cifrari sono detti anche cifrari a sostituzione o cifrari a scorrimento a causa del loro modo di operare: la sostituzione avviene lettera per lettera, scorrendo il testo dall'inizio alla fine. Una possibile traduzione potrebbe essere la seguente: il codice ASCII standard su 8 bit di ciascun carattere del messaggio di testo viene modificato sommandoci una costante intera *K*.

Ai fini del nostro progetto, la sostituzione viene eseguita solo per lettere maiuscole e minuscole (A-Z e a-z), che vengono codificate unicamente in altre lettere. Si lasciano gli altri simboli (numeri, spazi, caratteri speciali) invariati. Maiuscole e minuscole vengono preservate.

Esempio

Si veda la pagina precedente con il plaintext AMO AssEMbLY (K=1)
Inoltre, supponendo K = -2, si ottengono cod(A) = Y, cod(d) = b, cod(@) = @, cod(z) = x, cod(1) = 1

Cifrario a Blocchi

(Da Wikipedia) Un algoritmo di cifratura a blocchi è composto da due parti, una che cifra,utilizzando *m* caratteri per il blocco da cifrare e *k* caratteri per la chiave *key* da utilizzare durante la cifratura, restituendo *m* caratteri di uscita per il cyphertext. Una versione semplice per la cifratura a blocchi si può intendere come segue: **la parola viene partizionata in nb blocchi**, ottenuti come nb = m/k arrotondato all'intero superiore. Ogni blocco in B = $\{b_1, b_2, \ldots b_{nb}\}$ contiene al più k elementi consecutivi della stringa da cifrare. **Ogni elemento di ciascun blocco viene cifrato sommando la codifica ASCII di un carattere della chiave alla codifica ASCII del carattere del blocco** come segue

$$\text{For each } b_i \text{ in B } (1 \le i \le nb),\ cb_i = cod(b_{ij}) + cod(key_j),\ 1 \le j \le k,$$

ottenendo il cyphertext ct = $\{cb_1, cb_2, \ldots cb_{nb}\}$, composto da nb blocchi cifrati. *Tutti i caratteri della stringa di partenza vengono cifrati con questa codifica.* Vista la restrizione per l'alfabeto di input di caratteri c con $32 \le cod(c) \le 127$ sia per il plaintext sia per la key, la cifratura diventa:

$$\text{For each } b_i \text{ in B } (1 \le i \le nb),\ cb_i = \{[(cod(b_{ij}) - 32) + (cod(key_j) - 32)] \,\%\, 96\} + 32,\ 1 \le j \le k,$$

Esempio

pt = LAUREATO_1, key = OLE

Si calcola Cod(O) = 79, Cod(L) = 76, Cod(E) = 69 andando a consultare la tabella ASCII e poi da ciascun Cod(ct) si trova il carattere corrispondente nella parte di tabella ASCII per ottenere il cyptertext ct.

Pt	L	A	U	R	E	A	T	O	_	1
Cod(pt)	76	65	85	82	69	65	84	79	95	49
Key	O	L	E	O	L	E	O	L	E	O
Cod(key)	79	76	69	79	76	69	79	76	69	79
Cod(ct)	**123**	**109**	**122**	**33**	**113**	**102**	**35**	**123**	**36**	**96**
ct	**{**	**m**	**z**	**!**	**q**	**f**	**#**	**{**	**$**	**'**

Cifratura Occorrenze

A partire dal primo carattere del plaintext (alla posizione 1), **il messaggio viene cifrato come una sequenza di stringhe separate da esattamente 1 spazio** (ASCII 32) in cui ciascuna stringa ha la forma "*x*-p_1-...-p_k", dove *x* è la prima occorrenza di ciascun carattere presente nel messaggio, p_1...p_k sono le k posizioni in cui il carattere *x* appare nel messaggio (con p_1<...<p_k), ed in cui ciascuna posizione è preceduta dal carattere separatore '-' (per distinguere gli elementi della sequenza delle posizioni).

Note:

- non c'è un ordine prestabilito per le lettere una volta codificata la stringa
- la codifica usa due separatori: lo spazio (ASCII 32) ed il trattino (ASCII 45). Ciò che sta tra due trattini deve essere sempre un numero che indica l'occorrenza di una lettera nella stringa di base, mentre ciò che segue lo spazio è sempre il carattere di riferimento, eccetto il caso in cui il carattere di riferimento sia lo spazio in sè (si veda l'esempio sotto).
- il cypertext ottenuto con questa codifica ha generalmente una lunghezza maggiore del plaintext di partenza

Esempio

Pt = "sempio di messaggio criptato -1"

La cifratura con questo algoritmo produrrà un cyphertext ct = "e-2-12 s-1-13-14 m-3-11 p-4-24 i-5-9-18-23 o-6-19-28 -7-10-20-29 d-8 a-15-26 g-16-17 c-21 r-22 t-25-27 --30 1-31".

- Nella stringa " -7-10-20-29" il carattere in codifica è lo spazio (' ', ASCII decimale 32), che appare nelle posizioni 7, 10, 20 e 29 del messaggio.
- Nella stringa "--30" il carattere in codifica è '-' (il secondo carattere '-' è il carattere separatore fra gli elementi della sequenza), che appare nella posizione 30 del messaggio.
- Nella stringa "1-31" il carattere in codifica è '1', che appare nella posizione 31 del messaggio.
- In memoria, la prima parte del cyphertext "e-2-12 " dovra apparire su 7 byte, uno per carattere, come cod(e), cod(-), cod(2), cod(-), cod(1), cod(2), cod( ) -> 101, 45, 50, 45, 49, 50, 32

Dizionario

Ogni possibile simbolo ASCII viene mappato con un altro simbolo ASCII secondo una certa funzione, che riportiamo di seguito definita per casi.

- Se il carattere *ci* è una lettera minuscola (min), viene sostituito con l'equivalente maiuscolo dell'alfabeto in ordine inverso es. Z = ct(a), A = ct(z).
- Se il carattere *ci* è una lettera maiuscola (mai), viene sostituito con l'equivalente minuscolo dell'alfabeto in ordine inverso es. z = ct(A), y = ct(B), a =ct(Z).
- Se il carattere *ci* è un numero (num), ct(ci) = ASCII(cod(9)-num)
- In tutti gli altri casi (sym), *ci* rimane invariato, ovvero ct(ci) = ci

Esempio

Pt = myStr0ng P4ssW_

Pt	m	y	S	t	r	0	n	g		P	4	s	s	W	_
Tipo ci	min	min	mai	min	min	num	min	min	sym	mai	num	min	min	mai	sym
ct	**N**	**B**	**h**	**G**	**I**	**9**	**M**	**T**		**k**	**5**	**H**	**H**	**d**	**_**

Nota: Trasformazione dello 0: ASCII(cod(9)-0) = ASCII(57-0) = ASCII(57) = 9,

Altre Cifrature

Altre 2 possibili cifrature, basate su modifiche della stringa di caratteri, sono le seguenti.

- **Inversione**: il cyphertext è rappresentato dalla stringa invertita es. pt = BUONANOTTE, ct = ETTONANOUB. Si noti come nel caso di parole palindrome, pt = ct.
- Cifrario a **sostituzione differenziando vocali-consonanti-simboli**. Una estensione della sostituzione dove invece di avere un unico valore di k si hanno 4 valori: kv (per le vocali), kc (per le consonanti), kn (per i numeri), e ks (per i simboli, ovvero spazi, trattini etc).
  Si identificano i seguenti alfabeti circolari (dichiarabili come stringhe a loro volta nel campo . data, per agevolare lo scorrimento):
  - Dizv - Vocali: A E I O U Y A E ...
  - Dizc - Consonanti: B C D F G H J K L M N P Q R S T V W X Z B C D ...
  - Dizn - Numeri 0 1 2 3 4 5 6 7 8 9 0 1 2 ...
  - Dizs - Simboli in tabella sotto.

Cod. ASCII	32	33	35	36	37	38	39	42	43	44	45	46	47	58	59	60	61	62	63	64	32
Simbolo		!	#	$	%	&	‘	*	+	,	-	.	/	:	;	<	=	>	?	@	

La codifica funziona come la sostituzione, ma **usando il k ϵ {kv, kc, kn, ks} appropriato in base alla classe** {vocale, consonante, numero, simbolo} di ciascun carattere del plaintext, **utilizzando l’alfabeto circolare corretto, ed invertendo maiuscole e minuscole per vocali e consonanti**. Ad esempio, la codifica di una vocale non può essere altro che una vocale. *Quindi se io ho la vocale A, e kc = -2, otterrò cyphertext u (minuscolo, ogni volta maiuscole e minuscole sono invertite)*. Simboli che non risultano in nessuna delle 4 categorie sopra rimangono invariati (si veda _ nell’esempio sotto).

Pt = myStr0ng P4ssW_

Kv = -3, Kc = 2, Kn = 12, ks = 10

Pt	**m**	**y**	**S**	**t**	**r**	**0**	**n**	**g**		**P**	**4**	**s**	**s**	**W**	**_**
Tipo (c, v, n, s)	c	v	c	c	c	n	c	c	s	c	n	c	c	c	same
ct	**P**	**I**	**v**	**W**	**T**	**2**	**Q**	**J**	**-**	**r**	**6**	**V**	**V**	**z**	**_**

Progetto Assembly

Costruendo su quanto detto sopra, il progetto di AE 19/20 riguarda la progettazione e la scrittura di un codice assembly RISC-V che simuli alcune funzioni di cifratura e decifratura di un messaggio di testo, interpretato come sequenza di caratteri ASCII.

In particolare il programma dovrà consentire di **cifrare** e **decifrare** un messaggio di testo (plaintext) fornito dall'utente come variabile *myplaintext* di tipo stringa (.string in RIPES). Diverse funzioni di cifratura, e le conseguenti di decifratura, dovranno essere implementate come di seguito.

A. Cifrario a Sostituizione, configurabile tramite variabile sostK, che indica lo shift alfabetico.
B. Cifrario a blocchi, con la chiave che è una stringa da considerare come variabile blocKey
C. Cifratura Occorrenze
D. Dizionario

Nel caso in cui il gruppo sia costituito da 2 o 3 persone, il gruppo dovrà implementare anche alcune funzioni di cifratura e decifratura ***addizionali***, come descritto sotto. Per gruppi di **due persone**, se ne potrà scegliere **una delle due**, mentre i gruppi di **tre persone** dovranno realizzarle **entrambe**.

E. Inversione
F. Sostituzione differenziata, configurabile tramite variabili kv, kc, kn, ks, che indicano gli shift per le vocali, consonanti, numeri, simboli.

Oltre alla variabile myplaintext (**dimensione massima 100 caratteri, caratteri c che possono essere unicamente tali che 32 ≤ cod(c) ≤ 127 per evitare caratteri ASCII speciali**), il programma richiede un input addizionale che specifica le *modalità di applicazione delle cifrature.* Tale variabile *mycypher* è una stringa S = "$S_1..S_n$" formata da al massimo 5 caratteri (quindi con $1 \leq n \leq 5$), in cui ciascun carattere S_i (con $1 \leq i \leq n$) corrisponde ad uno fra i caratteri 'A', 'B', 'C', 'D', 'E', 'F', ed identifica l' i-mo cifrario da applicare al messaggio. **L'ordine delle cifrature è quindi stabilito dall'ordine in cui appaiono i caratteri nella stringa. Inltre, ogni cifrario restituisce un cyphertext che è una sequenza di caratteri c t.c. 32 ≤ cod(c) ≤ 127**

A titolo di esempio, si riportano alcuni possibili parole chiave: "C", oppure "AEC", oppure "DEDD", ... Ad esempio, la cifratura del messaggio di testo con la parola chiave "AEC" determinerà l'applicazione dell'algoritmo A, poi dell'algoritmo E (sul messaggio già cifrato con A) ed infine dell'algoritmo C (sul messaggio già cifrato prima con A e poi con E).

Il programma dovrà considerare le variabili d'ambiente myplaintext e mycypher, oltre ai vari parametri messaggio da cifrare, e produrre in output a video, i vari cypthertext ottenuti dopo l'applicazione di ciascun singolo passaggio di cifratura, separati da un newline. Allo stesso modo, si dovranno applicare le funzioni di decifratura *a partire dal messaggio precedentemente cifrato* in ordine inverso rispetto alle cifrature. **L'ultimo messaggio stampato a video dovrà corrispondere al plaintext di partenza**.

Note Generali del Progetto

- "newline" ha codice ASCII 10, e può essere usato per differenziare i 3 output del programma
- RIPES non gestisce bene stringe in input con dimensione divisibile per 4 (non mette il fine stringa). Quindi, si usi un input che è sempre una stringa lunga n caratteri, dove *n non è divisibile per 4*.

Note e Modalità di Consegna

Note

- Seguire fedelmente tutte le specifiche dell'esercizio (incluse quelle relative ai nomi delle variabili e al formato del loro contenuto).
- Rendere il codice **modulare** utilizzando ove opportuno **chiamate a procedure** e **rispettando le convenzioni fra procedura chiamante/chiamata**. La modularità del codice ed il rispetto delle convenzioni saranno aspetti fondamentali per ottenere un'ottima valutazione del progetto. Si richiede in particolare di *implementare ogni cifrario (ciascun algoritmo A-B-C-D-E, e le loro inversioni per la decifratura) come una procedura separata.*
- Commentare in modo significativo (non commentare *addi s3, s3, 1* con "sommo uno ad s3"....).

Modalità di Esame

- Per sostenere l'esame è necessario consegnare preventivamente il codice e una relazione PDF sul progetto assegnato. Il progetto può essere svolto in gruppo, **minimo 1 massimo 3 persone per gruppo**, ma ogni gruppo di lavoro dovrà consegnare un'unica relazione.
- Il codice consegnato deve essere funzionante sul simulatore RIPES, usato durante le lezioni.
- La scadenza esatta della consegna verrà resa nota di volta in volta, in base alle date dell'appello.
- Discussione e valutazione: la discussione degli elaborati avverrà contestualmente all'esame orale e **prevede anche domande su tutti gli argomenti di laboratorio trattati a lezione**.
- Membri dello stesso gruppo faranno orali individuali sul progetto, senza nessun vincolo di presentarsi assieme all'esame. Comunque, *il progetto deve essere consegnato sempre prima che qualunque membro del gruppo si iscriva all'esame*.

Struttura della Consegna

La consegna dovrà consistere di un unico achivio contenente 3 componenti. L'archivio dovrà essere caricato sul sito moodle del corso seguendo il link che verrà reso disponibile alla pagina del corso.

- Un archivio contenente il **codice** assembly
- un **breve video** (max 5 minuti) dove si registra lo schermo del dispositivo durante l'esecuzione del programma, commentandone il funzionamento in base a 2-3 combinazioni di input diverse
- la **relazione** in formato PDF, strutturata come segue.
    1. **Informazioni** su autori, indirizzo mail, matricola e data di consegna
    2. **Descrizione** della soluzione adottata, trattando principalmente i seguenti punti:
        a. Descrizione ad alto livello di ciascun algoritmo di cifratura/decifratura, di altre eventuali procedure e del main, in linguaggio naturale, con flow-chart, in pseudo-linguaggio, etc
        b. Uso dei registri e memoria (stack, piuttosto che memoria statica o dinamica)
    3. **Test di corretto funzionamento**, per fornire evidenze del corretto funzionamento del programma.
    4. **Codice assembly** implementato e commentato in modo chiaro ed esauriente.
        - Nota: alla fine della relazione va inserito **TUTTO** il codice così come appare nel sorgente

Tabella ASCII ridotta (interessano solo caratteri da ASCII 32 a ASCII 127)

Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	`
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(	01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41	)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledge	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[	01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	|
00011101	29	Group separator	00111101	61	=	01011101	93	]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del