ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής & Τηλεπικοινωνιών
2η Εργασία – Τμήμα Περιττών Αριθμών
K22: Λειτουργικά Συστήματα – Χειμερινό Εξάμηνο '09
Ημερομηνία Ανακοίνωσης: Τετάρη 18 Νοεμβρίου 2009
Ημερομηνία Υποβολής: Τετάρτη 9 Δεκεμβρίου 2009 και Ωρα 21:00

Εισαγωγή στην Εργασία:

Στα πλαίσια αυτής της άσκησης θα πρέπει να γράψετε ένα πρόγραμμα που κρυπτογραφεί (και αποκρυπτογραφεί) αρχεία κειμένου ASCII, χρησιμοποιώντας τον αλγοριθμο ICEBERG (ο οποίος παρουσιάζεται εκτενώς παρακάτω).

Η διαδικασία της (απο)κρυπτογράφησης θα πρέπει να γίνεται από πολλές διεργασίες, κάθε μία εκ των οποίων θα (απο)κωδικοποιεί ένα μέρος μόνο του αρχείου εισόδου, με διαφορετικό όμως κλειδί, ώστε να παρέχεται μεγαλύτερη ασφάλεια. Οι διεργασίες αυτές θα είναι παιδιά (children) του βασικού προγράμματός σας που θα δημιουργούνται με την κλήση συστήματος fork() και θα πρέπει να επικοινωνούν μεταξύ τους με απλά σήματα (signals). Λεπτομέρειες για τους ρόλους των διεργασιών παρουσιάζονται παρακάτω.

Για την άσκηση αυτή θα χρειαστείτε:

1. Τη κλήση συστήματος fork() για την δημιουργία διεργασιών.
2. Απλό συγχρονισμό με signals (signal(), kill(), pause()) .
3. Τις exec* κλήσεις συστήματος για την αντικατάσταση/εκτέλεση διάφορων προγραμμάτων (οι Workers θα πρέπει υποχρεωτικά να την εκτελέσουν αφού δημιουργηθούν).
4. Διάφορες άλλες κλήσεις συστήματος όπως: getpid(), getppid(), wait(), exit(), sleep().

Διαδικαστικά:

◇ Το πρόγραμμα σας θα πρέπει να γραφτεί σε C (ή C++ αν θέλετε, χωρίς όμως την χρήση STL) και να τρέχει στις μηχανές Linux του τμήματος.
◇ Παρακολουθείτε την ιστοσελίδα του μαθήματος για επιπρόσθετες ανακοινώσεις στο URL: www.di.uoa.gr/~ad/OSdir/index.html και την λίστα mailman του μαθήματος.
◇ Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις, αξιολόγηση, βαθμολόγηση, κλπ.) είναι ο κος Παναγιώτης Λιάκος (grad0990[at]di.uoa.gr) και ο κος Γιώργος Μπαλατσούρας (std04139[at]di.uoa.gr).

Η ιεραρχία και ο ρόλος των διεργασιών:

Parent Process: Η διεργασία γονέας θα πρέπει αρχικά να δημιουργήσει το αρχείο που θα περιέχει τελικά το κρυπτογραφημένο (ή αποκρυπτογραφημένο αντίστοιχα) μήνυμα. Έπειτα, θα δημιουργεί n Workers/διεργασίες παιδιά (με τη κλήση συστήματος fork()). Αφού έχουν δημιουργηθεί όλοι οι Workers , στέλνει το σήμα $SIGUSR1$ στον τελευταίο (n -οστή διεργασία παιδί), ειδοποιώντας τον πως πλέον όλες οι διεργασίες έχουν δημιουργηθεί και μπορεί να αρχίσει την (απο)κρυπτογράφηση. Η parent process μετά περιμένει την ολοκλήρωση των Workers που δημιούργησε (κλήση συστήματος wait). Θα πρέπει, ωστόσο, να ελέγχει κάθε φορά το status της wait ώστε σε περίπτωση που ένας Worker δεν τερμάτισε επιτυχώς να αναλάβει να τερματίσει όλους τους Workers που τρέχουν ακόμα (στέλνοντας το signal *SIGKILL*) και να διαγράψει το αρχείο που δημιούργησε.

Child Process i (Worker): Ο Worker αρχικά θα πρέπει να περιμένει να λάβει το σήμα *SIGUSR1* . Όταν το λάβει, στέλνει με τη σειρά του το *SIGUSR1* στην Child Process i-1. (Με αυτό το τρόπο, το *SIGUSR1* θα

μεταδοθεί μέχρι και την Child Process 1 όπου και θα σταματήσει). Για να γίνει αυτό, θα πρέπει η διεργασία αυτή να γνωρίζει το pid της προηγούμενης διεργασίας που δημιουργήθηκε (i-1) , δηλαδή να περαστεί σαν παράμετρος όταν γίνει η κλήση exec. Αφού στείλει το *SIGUSR1* , κάθε Worker αρχίζει το δικό του μέρος της (απο)κρυπτογράφησης. Τα μόνα που χρειάζεται γι' αυτό στις παραμέτρους της (exec) είναι το όνομα του αρχείου εισόδου που περιέχει το μήνυμα, το *offset* από το οποίο αρχίζει το δικό της μέρος (ώστε να γίνει κάποιο fseek εκεί) και το *length* που ορίζει το πόσα *bytes* της αντιστοιχούν (ξεκινώντας από το *offset*). Επίσης, χρειάζεται το όνομα του αρχείου εξόδου (κοινό για όλους) στο οποίο θα γράφεται το νέο κρυπτογραφημένο μήνυμα (ή αποκρυπτογραφημένο αντίστοιχα) που θα προκύπτει. Για το αρχείο αυτό θα ισχύουν τα ίδια *length* και offset με το αρχείο εισόδου.

Η Είσοδος του Προγράμματός σας:

Το κύριο πρόγραμμά σας θα πρέπει να παίρνει από την γραμμή εντολής τα ορίσματα της λειτουργίας του ως εξής:

prompt> iceberg [-e/-d] -i input_file -o output_file -c configuration_file

Τα ορίσματα μπορούν να εισαχθούν με οποιαδήποτε σειρά.

1. *iceberg* : το executable του προγράμματος σας.
2. *[-e/-d]* : είναι το mode of operation του αλγορίθμου (encyption/decryption).
3. *input_file* : το όνομα του text αρχείου εισόδου.
4. *output_file* : το όνομα του αρχείου εξόδου που θα παραχθεί.
5. *configuration_file* : το όνομα του αρχείου που περιέχει όλες τις απαραίτητες ρυθμίσεις και τα κλειδιά που θα χρησιμοποιήσει η κάθε process. Η μορφή του αρχείου αυτού είναι η εξής:

   $< integer\ N >$
   $< key\ k_1 > < integer\ l_1 >$
   $< key\ k_2 > < integer\ l_2 >$
   ...
   $< key\ k_n > < integer\ l_n >$
   Ο ακέραιος N καθορίζει το πλήθος των γύρων του αλγορίθμου ICEBERG. Το πλήθος των υπολοίπων γραμμών καθορίζει τον αριθμό των workers που πρέπει να δημιουργηθούν. Το k_i είναι το κλειδί που θα χρησιμοποιεί ο worker i για να κρυπτογραφήσει τα μηνύματα που του αντιστοιχούν. Τα κλειδιά θα δίνονται στή δεκαεξαδική τους μορφή. Τέλος ο ακέραιος l_i είναι το πλήθος των μηνυμάτων που θα πρέπει να κρυπτογραφήσει ο worker i. Όπως αναλύεται στην περιγραφή του αλγορίθμου, κάθε μήνυμα έχει μέγεθος 64 bits που αντιστοιχούν σε 8 χαρακτήρες ASCII. Οπότε ο worker i θα πρέπει να κρυπτογραφήσει l_i*8 χαρακτήρες ASCII. Το configuration file που θα χρησιμοποιηθεί σε μια κρυπτογράφηση πρέπει να είναι ίδιο με το αρχείο που θα χρησιμοποιηθεί στην αντίστοιχη αποκρυπτογράφηση.

Τι Πρέπει να Παραδοθεί:

1. Ένα tar-file με όλη σας τη δουλειά δηλ. source files, header files, output files.
2. Ένα Readme.txt που θα περιέχει μια σύντομη και περιεκτική εξήγηση για τις επιλογές που έγιναν στο design του προγράμματός σας και ό,τι άλλο εσείς κρίνετε απαραίτητο (1-2 σελίδες ASCII κειμένου).
3. Ένα Makefile που θα χρησιμοποιείται για να γίνει 'αυτόματα' compiled το πρόγραμμα σας.

Άλλες Σημαντικές Παρατηρήσεις:

1. Οι εργασίες ειναι ατομικές.
2. Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux της σχολής αλλιώς δεν θα είναι δυνατόν να βαθμολογηθεί. Σε καμία περίπτωση τα MS-Windows δεν είναι διαθέσιμη επιλογή πλατφόρμας για την παρουσίαση αυτής της άσκησης.
3. Αντιγραφή κώδικα σημαίνει μηδενισμό για οποια-/οποιονδήποτε έχει ανάμειξη. Αντιγραφή κώδικα απλά σας δίνει μηδέν στο μάθημα.

Αλγόριθμος ICEBERG

Ο ICEBERG είναι ένας κώδικας μπλοκ (block cipher) ο οποίος κρυπτογραφεί μηνύματα μεγέθους 64-bit χρησιμοποιώντας κλειδιά μεγέθους 128-bit. Το κείμενο που θα χειρίζεστε θα είναι σε μορφή ASCII ενώ το κλειδί σε δεκαεξαδική μορφή. Οι συναρτήσεις του αλγορίθμου όμως μπορούν να υλοποιηθούν ευκολότερα με το κείμενο και το κλειδί σε δυαδική μορφή, οπότε το πρόγραμμά σας θα πρέπει να κάνει αυτή τη μετατροπή πριν χρησιμοποιήσει τον αλγόριθμο.

Ένα παράδειγμα μετατροπής είναι αυτό:

'Usurper.' (ASCII)→ '01010101011100110111010101110010011100000110010101110010001011 10' (binary)

S-boxes & P-boxes

Ο ICEBERG αποτελείται κυρίως από λειτουργίες αντικατάστασης και μετάθεσης, οι οποίες περιγράφονται από S-boxes και P-boxes αντίστοιχα.

Όπως φαίνεται και στο Σχήμα 1, ένα S-box είναι ένας πίνακας αντικατάστασης ο οποίος αντιστοιχίζει τιμές εισόδου σε τιμές εξόδου. Έτσι για το εικονιζόμενο S-box (S0) αν η είσοδος ήταν 'b' (δεκαεξαδικό), η έξοδος θα ήταν 'e'.

Αντίθετα, σε ένα P-box γίνεται αντιμετάθεση των bits της εισόδου. Έτσι αν στο εικονιζόμενο P-box (P4) η είσοδος ήταν '1011' (δυαδικό), η έξοδος θα ήταν '0111'. Θεωρούμε πως η θέση 0 αντιστοιχεί στο λιγότερο σημαντικό bit.

S-box: S0

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
d	7	3	2	9	a	c	1	f	4	5	e	6	0	b	8

```
//Example of an S-box
S0(a){
    switch(a){
        case '0': temp = 'd'; break;
        case '1': temp = '7'; break;
        case '2': temp = '3'; break;
        case '3': temp = '2'; break;
        case '4': temp = '9'; break;
        case '5': temp = 'a'; break;
        case '6': temp = 'c'; break;
        case '7': temp = '1'; break;
        case '8': temp = 'f'; break;
        case '9': temp = '4'; break;
        case 'a': temp = '5'; break;
        case 'b': temp = 'e'; break;
        case 'c': temp = '6'; break;
        case 'd': temp = '0'; break;
        case 'e': temp = 'b'; break;
        case 'f': temp = '8'; break;
    }
    return temp;
}
```

P-box: P4

0	1	2	3
1	0	3	2

```
//Example of a P-box
P4(a[4])
{
    temp[0]=a[1];
    temp[1]=a[0];
    temp[2]=a[3];
    temp[3]=a[2];
    return temp;
}
```

S-box: hex input
P-box: binary input

Σχήμα 1: Παράδειγμα υλοποίησης ενός S-box κι ενός P-box.

Όλα τα S-boxes και τα P-boxes του αλγορίθμου παρατίθονται στο τέλος του αρχείου.

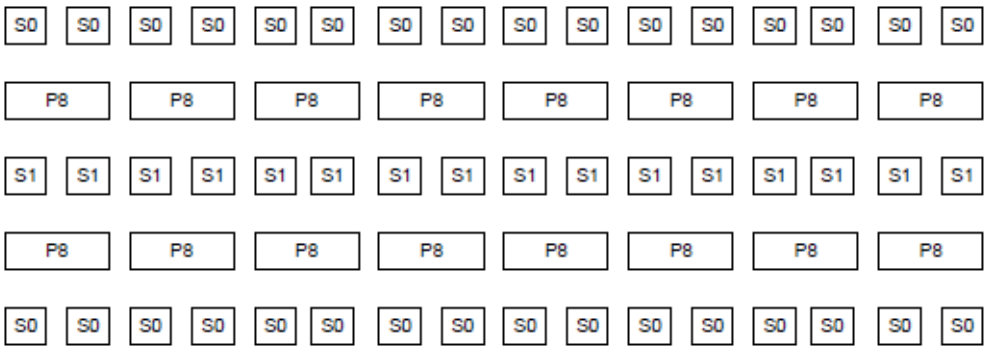
Ανάλυση του αλγορίθμου:

Ο αλγόριθμος αποτελείται από δυο επιμέρους κομμάτια. Τη συνάρτηση γύρου και τον προγραμματισμό του κλειδιού. Το πρώτο κομμάτι αναλαμβάνει ουσιαστικά όλο το έργο της κρυπτογράφησης ενώ το δεύτερο φροντίζει να αλλάζει το κλειδί που χρησιμοποιείται έτσι ωστε ο κώδικας να παρέχει μεγαλύτερη ασφάλεια. Τα κομμάτια αυτά επαναλαμβάνονται πολλές φορές για κάθε μήνυμα έτσι ώστε να παρέχεται μεγαλύτερη ασφάλεια.

Συνάρτηση γύρου:

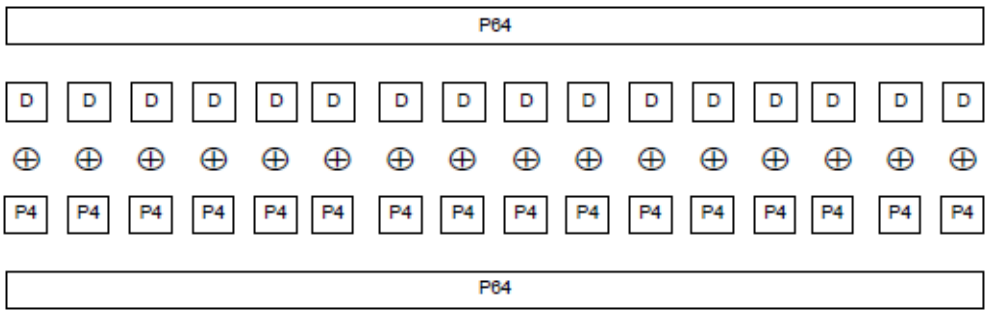
Για το επίπεδο αυτό θα πρέπει να υλοποιήσετε τρεις συνολικά συναρτήσεις οι οποίες θα αναλυθούν σε αυτή την ενότητα.

1. **gama(a[64]){}**: Η συνάρτηση αυτή δέχεται 64 bits και εφαρμόζει σε αυτά ένα πλήθος από S-boxes και P-boxes. Όπως φαίνεται στην παρακάτω εικόνα, αρχικά στα 64 bits, εφαρμόζεται ένα επίπεδο από S-boxes τύπου S0. Παρατηρούμε ότι υπάρχουν συνολικά 16 τέτοια κουτιά, καθώς κάθε ένα χειρίζεται 4 bits. Αν υποθέσουμε ότι η είσοδος είναι αυτή του παραδείγματος μετατροπής σε δυαδική μορφή, τότε στο δεξιότερο S-box της εικόνας η είσοδος θα ήταν τα τελευταία τέσσερα bits δηλαδη τα 1110 που αντιστοιξούν στο 'e' στο δεκαεξαδικό. Έτσι η έξοδος θα ήταν η 'b' (1011 στο δυαδικό).
   Τα καινούργια 64 bits που θα παραχθούν από αυτά τα κουτιά θα οδηγηθούν σε ένα επίπεδο από P-boxes τύπου P8. Αυτή τη φορά τα κουτιά είναι 8 καθώς κάθε ένα χειρίζεται 8 bits. Ακολουθεί ένα επίπεδο με κουτιά αντικατάστασης τύπου S1, ένα επίπεδο με κουτιά αντιμετάθεσης τύπου P8 και τέλος ένα επίπεδο με κουτιά αντικατάστασης τύπου S0. Η έξοδος της συνάρτησης είναι τα 64 bits που εξέρχονται από το τελευταίο αυτό επίπεδο.
   Παράδειγμα:
   Input: 1fbf2fbf0a6aeb3b (HEX)→ Output: f1e368e31accc667 (HEX)



Σχήμα 2: Συνάρτηση gama().

2. **EK(a[64],b[64]){}**: Όπως φαίνεται και στο Σχήμα 3, η συνάρτηση αυτή αρχικά εφαρμόζει στα 64 bits της εισόδου a ένα P-box τύπου P64. Έπειτα οδηγεί την έξοδο του κουτιού αυτού σε ένα επίπεδο με S-boxes τύπου D. Στο επόμενο βήμα εκτελείται ένα bitwise XOR μεταξύ της εξόδου του πιο πάνω επιπέδου και της εισόδου b. Το επίπεδο αυτό είναι η τρίτη συναρτηση που περιγράφεται σε αυτή την ενότητα, με όνομα SK. Η είσοδος b χρησιμοποιείται μόνο από αυτή τη συν·αρτηση. Ακολουθούν άλλα δύο επίπεδα με κουτιά αντιμετάθεσης.
   Παράδειγμα:
   Input: f1e368e31accc667, da52cf52ecca5165 (HEX)→ Output: a70c76dd3224fa65 (HEX)



Σχήμα 3: Συνάρτηση EK().

3. **SK(a[64],b[64]){}**: Η συνάρτηση αυτή απλώς επιστρέφει το bitwise XOR των εισόδων της.
   Παράδειγμα:
   Input: 552e502e3a207570 (HEX)→ Output: 4a917f91304a9e4b (HEX)

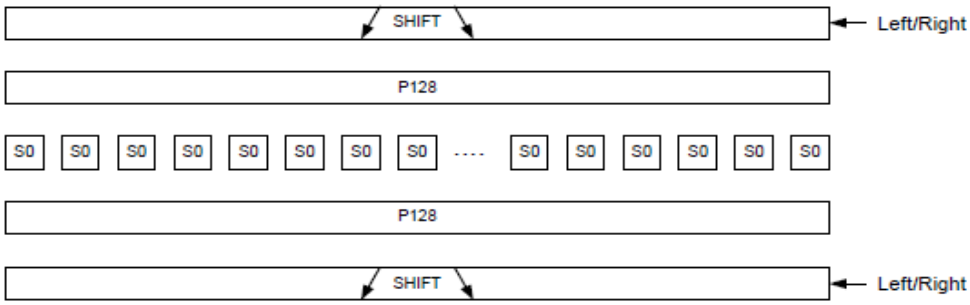
Προγραμματισμός κλειδού:

Για το επίπεδο αυτό θα πρέπει να υλοποιήσετε τέσσερις συνολικά συναρτήσεις οι οποίες θα αναλυθούν σε αυτή την ενότητα. Όπως αναφέρθηκε αρχικά ο ICEBERG χρησιμοποιεί κλειδιά μεγέθους 128 bit. Οπότε οι παρακάτω συναρτήσεις χειρίζονται 128 bits και όχι 64 όπως οι προηγούμενες.

1. **TC(a[128],C){}**: Η συνάρτηση αυτή χρησιμοποιείται από τη συνάρτηση BC() που θα αναλυθεί παρακάτω. Η TC πραγματοποιεί ολίσθηση (αριστερή ή δεξιά ανάλογα με την τιμή του C) στα 128 bits της εισόδου. Η ολίσθηση περιγράφεται από τον παρακάτω κώδικα.

```
TC(a[128],C)   // Shift left/right
{
  for(i=0;i<128;i++)
    {
      if(C==1){temp[i]=a[(i+8)mod 128];}
      if(C==0){temp[i]=a[(i-8)mod 128];}
    }
return temp;
}
```

   Παράδειγμα:
   Input: 6563796f4a2079622073657373796c55, 0 (HEX)
   Output: 63796f4a2079622073657373796c5565 (HEX)
2. **BC(a[128],C){}**: Όπως φαίνεται και στην παρακάτω εικόνα η συνάρτηση αυτή αρχίζει και τελείωνει πραγματοποιώντας ολίσθηση στο κλειδί. Χρησιμοποιεί δηλαδή δύο φορές την παραπάνω συνάρτηση. Ανάμεσα στις ολισθήσεις εφαρμόζει στο κλειδί δύο επίπεδα με κουτιά αντιμετάθεσης P128 και ένα επίπεδο με κουτιά αντικατάστασης S0.
   Παράδειγμα:
   Input: 6563796f4a2079622073657373796c55, 0 (HEX)
   Output: 1532fbd78a0dfb9538808500f9ebdfce (HEX)



Σχήμα 4: Συνάρτηση BC().

3. **E(a[128]){}**: Η συνάρτηση αυτή αναλαμβάνει να συμπιέσει το κλειδί σε 64 bits. Αυτό το επιτυγχάνει κρατώντας μόνο τα περιττά bytes του κλειδιού. Θεωρούμε ότι η αρίθμηση των bytes αρχίζει από το μηδέν, δηλαδή το πρώτο περιττό byte αντιστοιχεί στα bits {8-15}.
   Παράδειγμα:
   Input: 6563796f4a2079622073657373796c55 (HEX)→ Output: 65794a792065736c (HEX)
4. **f(a[64],sel){}**: Η συνάρτηση αυτή δεχεται τα 64 bits του συμπιεσμένου πλέον κλειδιού και για κάθε τετράδα από bits (0-3,4-7,...) κάνει τα παρακάτω:

```
f(a[64],sel)
//For each of the 4-bit groups of the input (0-3,4-7,...) do
if(sel == 0)
{
  temp[0]=a[0] XOR a[1];
  temp[1]=a[1];
  temp[2]=a[2] XOR a[3];
  temp[3]=a[3];
}
else if(sel == 1)
{
  temp[0]=a[0] XOR a[1] XOR a[2];
  temp[1]=a[1] XOR a[2];
  temp[2]=a[2] XOR a[3] XOR a[0];
  temp[3]=a[3] XOR a[0];
}
return temp;
```

   Παράδειγμα:
   Input: 65794a792065736c, 1 (HEX)→ Output: 4a917f91304a9e4b (HEX)

Κρυπτογράφηση και αποκρυπτογράφηση:

Έχοντας υλοποιήσει τις παραπάνω συναρτήσεις η κρυπτογράφηση και η αποκρυπτογράφηση πραγματοποιούνται από τις συναρτήσεις iceberg_encrypt και iceberg_decrypt οι οποίες σας δίνονται έτοιμες (θα χρειαστεί μόνο να ορίσετε τους τύπους δεδομένων που θα επιλέξετε να χρησιμοποιήσετε για να αναπαραστήσετε το

μήνυμα και το κλειδί, για τους οποίους δεν υπάρχει περιορισμός). Η είσοδος N είναι ο αριθμός των γύρων του αλγορίθμου. Ο αριθμός αυτός πρέπει να είναι άρτιος και μεγαλύτερος του 2 και η προκαθορισμένη τιμή του να είναι το 16.

Ένα παράδειγμα κρυπτογράφησης είναι το εξής (για N = 16):

m = 'U.P.: up' (ASCII)
key = 6563796f4a2079622073657373796c55 (HEX)
c = d2a2d6c636881880 (HEX)

```
iceberg_encrypt(key[128],m[64],N)
{
  // ROUND 0
  temp_key = key;
  sel_key = f(E(key),1);
  temp = SK(m,sel_key);
  // ROUNDS 1 - N/2
  for(i=0;i<(N/2);i++)
   {
      temp_key = BC(temp_key,0);
      sel_key = f(E(temp_key),1);
      temp = EK(gama(temp),sel_key);
   }
   // ROUNDS N/2+1 - N-1
   for(i=0;i<(N/2)-1;i++)
   {
      temp_key = BC(temp_key,1);
      sel_key = f(E(temp_key),1);
      temp = EK(gama(temp),sel_key);
   }
   // ROUND N
   temp_key = BC(temp_key,1);
   sel_key = f(E(temp_key),0);
   c = SK(gama(temp),sel_key);
   return c;
}

iceberg_decrypt(key[128],c[64],N)
{
  // ROUND 0
  temp_key = key;
  sel_key = f(E(key),0);
```

```
  temp = gama(SK(c,sel_key));
  // ROUNDS 1 - N/2
  for(i=0;i<(N/2);i++)
  {
    temp_key = BC(temp_key,0);
    sel_key = f(E(temp_key),0);
    temp = gama(EK(temp,sel_key));
  }
  // ROUNDS N/2+1 - N-1
  for(i=0;i<(N/2)-1;i++)
  {
    temp_key = BC(temp_key,1);
    sel_key = f(E(temp_key),0);
    temp = gama(EK(temp,sel_key));
  }
  // ROUND N
  temp_key = BC(temp_key,1);
  sel_key = f(E(temp_key),1);
  m = SK(temp,sel_key);
  return m;
}
```

S-boxes και P-boxes:

0	1	2	3
1	0	3	2

Table 1. p4.

0	1	2	3	4	5	6	7
0	1	4	5	2	3	6	7

Table 2. p8.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
d	7	3	2	9	a	c	1	f	4	5	e	6	0	b	8

Table 3. s_0.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
4	a	f	c	0	d	9	b	e	6	1	7	3	5	8	2

Table 4. s_1.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	e	d	3	b	5	6	8	7	9	a	4	c	2	1	f

Table 5. D.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	23	25	38	42	53	59	22	9	26	32	1	47	51	61
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
24	37	18	41	55	58	8	2	16	3	10	27	33	46	48	62
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
11	28	60	49	36	17	4	43	50	19	5	39	56	45	29	13
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
30	35	40	14	57	6	54	20	44	52	21	7	34	15	31	63

Table 6. p64.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
76	110	83	127	67	114	92	97	98	65	121	106	78	112	91	82
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
71	101	89	126	72	107	81	118	90	124	73	88	64	104	100	85
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
109	87	75	113	120	66	103	115	122	108	95	69	74	116	80	102
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
84	96	125	68	93	105	119	79	123	86	70	117	111	77	99	94
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
28	9	37	4	51	43	58	16	20	26	44	34	0	61	12	55
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
46	22	15	2	48	31	57	33	27	18	24	14	6	52	63	42
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
49	7	8	62	30	17	47	38	29	53	11	21	41	32	1	60
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
13	35	5	39	45	59	23	54	36	10	40	56	25	50	19	3

Table 7. p128.