rsa

Ο Αλγόριθμος RSA (rsa - 50 Μονάδες)
Κάθε φορά που συνδέεστε στο εργαστήριο με ssh, χρειάζεται να πληκτρολογήσετε τον κωδικό σας. Καθώς τον πληκτρολογείτε, τα bytes του κωδικού σας ταξιδεύουν στο δίκτυο ενσύρματα ή ασύρματα (θα μάθετε αργότερα πως) και καταλήγουν στον υπολογιστή του εργαστηρίου ώστε να πάρετε πρόσβαση. Με παρεμφερή διαδικασία μπαίνετε στο webmail σας, το instagram σας και ένα σωρό άλλες υπηρεσίες. Όμως... δεν μπορεί ο οποιοσδήποτε να δει αυτά τα bytes που στέλνετε και επομένως να δει τον κωδικό σας; Η απάντηση είναι ευτυχώς όχι και για αυτό ευχαριστούμε τον τομέα της κρυπτογραφίας. Σε αυτήν την άσκηση, θα ασχοληθούμε με έναν από τους πιο φημισμένους αλγορίθμους κρυπτογραφίας, τον αλγόριθμο RSA. Ο αλγόριθμος RSA κοινοποιήθηκε για πρώτη φορά το 1977. Πήρε το όνομά του από τα επώνυμα των εφευρετών του (Rivest-Shamir-Adleman) και η βασική ιδέα πίσω από τον αλγόριθμο είναι παρόμοια με όλα τα συστήματα κρυπτογραφίας που έχουμε σήμερα. Χρειαζόμαστε:
(1) μια συνάρτηση encrypt που να "κρύβει" το μήνυμά μας ώστε να μην μπορεί κάποιος άλλος να το διαβάσει καθώς το στέλνουμε και
(2) μια συνάρτηση decrypt η οποία να παίρνει το κρυπτογραφημένο μήνυμά μας και να το μετατρέπει (αποκρυπτογραφεί) στο αρχικό.
Πως όμως μπορούμε να "κρύψουμε" το μήνυμά μας; Η βασική ιδέα του RSA είναι η εξής: έστω ότι το μήνυμα που θέλουμε να στείλουμε είναι ένας ακέραιος m. Τότε για να κρύψουμε το μήνυμά μας αρκεί να το υψώσουμε σε μια μεγάλη δύναμη: m^x. Η αποκρυπτογράφηση μπορεί να γίνει εξίσου απλά, αρκεί να βρούμε έναν αριθμό y έτσι ώστε $(m^x)^y = m$. Βλέποντας αυτό το παράδειγμα, ίσως σκέφτεστε ότι x= 2 και y= 1
2 είναι μια πιθανή λύση στο πρόβλημά μας. Η σκέψη σας είναι σωστή, αλλά επειδή οποιοσδήποτε μπορεί να υπολογίσει την τετραγωνική ρίζα ενός αριθμού δεν μπορούμε να χρησιμοποιήσουμε αυτές τις πράξεις και αριθμούς για ασφαλή κρυπτογράφηση. Για να δούμε ποιους αριθμούς και πράξεις μπορούμε να χρησιμοποιήσουμε, χρειαζόμαστε πρώτα κάποιους ορισμούς:

Ορισμός 3. Ένας φυσικός αριθμός p μεγαλύτερος του 1 ονομάζεται πρώτος (prime) όταν έχει σαν μόνους διαιρέτες (το υπόλοιπο της διαίρεσης είναι 0) το 1 και το p. Για παράδειγμα, το 17 είναι πρώτος αριθμός, ενώ το 42 δεν είναι, αφού έχει σαν διαιρέτες το 2, το 3 και το 7, εκτός από τους 1 και 42. Μπορείτε να επιβεβαιώσετε ότι οι πρώτοι αριθμοί που είναι μικρότεροι από το 100 είναι οι εξής:

2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97

Μαθηματικοί στο παρελθόν έχουν ασχοληθεί με πολλά γνωστά προβλήματα όπως η κατανομή τους. Μέχρι σήμερα, δεν έχει βρεθεί μια συνάρτηση που να παράγει
τους πρώτους αριθμούς αποδοτικά.

Ορισμός 4. Δύο ακέραιοι a και b είναι πρώτοι μεταξύ τους ή σχετικά πρώτοι (coprime) όταν ο μέγιστος κοινός διαιρέτης τους είναι το 1, δηλαδή gcd(a,b) = 1. Για παράδειγμα, οι αριθμοί 8 και 9 είναι coprime εφόσον το gcd(8,9) = 1 παρόλο που κανείς από τους δυο τους δεν είναι ο ίδιος πρώτος.

Ορισμός 5. Η συνάρτηση φ: N→N του Euler (γνωστή και ως totient function) δέχεται έναν φυσικό αριθμό n και επιστρέφει το πλήθος των φυσικών αριθμών που είναι μικρότεροι του n και comprime με το n. Για παράδειγμα, φ(9) = 6, εφόσον υπάρχουν ακριβώς έξι coprime με το 9: 1, 2, 4, 5, 7 και 8. Η συνάρτηση φ έχει την πολλαπλασιαστική ιδιότητα, δηλαδή για κάθε δύο φυσικούς a, b με gcd(a,b) = 1 ισχύει ότι: φ(a·b) = φ(a)·φ(b). Επίσης, αν ο αριθμός aείναι πρώτος, τότε ισχύει πως φ(a) = a−1. Για παράδειγμα: φ(5) = 4, εφόσον οι αριθμοί 1, 2, 3, 4 είναι coprime ως προς το 5.

Έχοντας τους παραπάνω ορισμούς, μπορούμε επιτέλους να ορίσουμε τους περιορισμούς για να λειτουργήσει σωστά ο αλγόριθμος RSA:

1. Έστω οι ακέραιοι e, d, p, q(το μυστικό) και ο ακέραιος m(το μήνυμα)
2. Έστω ο ακέραιος N= p·q.
3. Περιορισμός: όλοι οι ακέραιοι πρέπει να είναι θετικοί.
4. Περιορισμός: το μήνυμα mπρέπει να είναι μικρότερο του N.
5. Περιορισμός: οι ακέραιοι ρκαι qείναι πρώτοι.
6. Περιορισμός: ο ακέραιος eείναι coprime με το φ(N).
7. Περιορισμός: οι ακέραιοι εκαι dείναι αντίστροφοι, δηλαδή: e·dmod φ(N) = 1.
   Με βάση τους παραπάνω περιορισμούς, μπορούμε πλέον να ορίσουμε την συνάρτηση κρυπτογράφησης encrypt ως: $encrypt(m) = m^e mod(N)$

Αντίστοιχα αν μας δώσουν έναν ακέραιο $c(= m^e mod(N))$ που είναι το κρυπτογραφημένο μήνυμα, μπορούμε να το αποκρυπτογραφήσουμε, χρησιμοποιώντας την συνάρτηση decrypt: $decrypt(c) = c^d mod(N)$

Το γιατί η παραπάνω πράξη μας δίνει το αρχικό μας μηνύμα είναι μεγάλη ιστορία — $c^d mod(N) = (m^e)^d mod(N) = m^{e \cdot d} mod(N) = m^{1+k\varphi(N)} mod(N) = m mod(N)$ — αλλά μπορείτε να βρείτε όλα τα βήματα της απόδειξης στην Wikipedia. Φτάσαμε επιτέλους στο ζητούμενο αυτής της άσκησης: να γράψετε ένα πρόγραμμα το οποίο να μπορεί να κρυπτογραφεί και να αποκρυπτογραφεί μηνύματα χρησιμοποιώντας τον παραπάνω αλγόριθμο RSA. Τα βήματα που θα υλοποιήσετε είναι αντίστοιχα με αυτά που τρέχουν κάθε φορά που κάνετε σύνδεση σε μια απομακρυσμένη υπηρεσία σήμερα στο διαδίκτυο! Οι τεχνικές προδιαγραφές ακολουθούν.
Τεχνικές Προδιαγραφές
• C Filepath: rsa.c
• Το πρόγραμμά θα πρέπει να παίρνει 5 ορίσματα από την γραμμή εντολών στην μορφή ./rsa op e d p q. Το πρώτο op επιτρέπεται να είναι "enc" (για encryption) και "dec" για decryption. Τα υπόλοιπα ορίσματα θα είναι ακέραιοι αριθμοί και

θα είναι στο εύρος [−1018,1018]. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.
• Το πρόγραμμα θα πρέπει να διαβάζει το μήνυμα m από την πρότυπη είσοδο (standard input) ως έναν δεκαδικό αριθμό επίσης στο εύρος [−1018,1018]. Γενικά, δεν θα σας ζητηθεί να χρησιμοποιήσετε ακεραίους με περισσότερα από 64 bits.
• Αν η είσοδος του χρήστη δεν πληροί τους περιορισμούς του αλγορίθμου RSA το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου 1 και αντίστοιχο μήνυμα λάθους όπως δείχνουμε παρακάτω στις ενδεικτικές εκτελέσεις.
• Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας πρέπει να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη `gcc -O3 -Wall -Wextra -Werror -pedantic -o rsa rsa.c`
• README Filepath: rsa/README.md
• Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

Παρακάτω παραθέτουμε αλληλεπιδράσεις με μια ενδεικτική λύση.

```
$ ./rsa pop 1 2 3 4
First argument must be 'enc' or 'dec'
$ echo $?
1
$ ./rsa enc 1 2 -3 4
Negative numbers are not allowed
$ echo $?
1
$ ./rsa enc 1 2 3 4
p and q must be prime
$ echo $?
1
$ ./rsa enc 3 6 17 19
e is not coprime with phi(N)
$ echo $?
1
$ ./rsa enc 5 6 17 19
e * d mod phi(N) is not 1
$ echo $?
1
$ echo 500 | ./rsa enc 5 173 17 19
Message is larger than N
$ echo $?
1
$ echo -42 | ./rsa enc 5 173 17 19
```

```
Negative numbers are not allowed
$ echo $?
1
$ echo 42 | ./rsa enc 5 173 17 19
264
$ echo 42 | ./rsa enc 17 26153 131 229
27187
$ echo 27187 | ./rsa dec 257 257 173 193
5343
$ echo 27187 | ./rsa dec 17 26153 131 229
42
$ echo 117 | ./rsa enc 17 26153 131 229 | ./rsa dec 17 26153 131 229
117
9

$ echo 43434343 | ./rsa enc 65537 2278459553 62971 38609 |
./rsa dec 65537 2278459553 62971 38609
43434343
$ echo 42 | ./rsa enc 65537 2278459553 62971 38609 > enc_msg
$ cat enc_msg
741088023
$ time ./rsa dec 65537 2278459553 62971 38609 < enc_msg
42
real 0m0.011s
user 0m0.004s
sys 0m0.008s
```

Καθώς οι εκθέτες του μυστικού στον οποίο υψώνουμε το μήνυμα γίνονται μεγαλύτεροι, είναι πιθανό πως το πρόγραμμά σας θα αρχίσει να παίρνει όλο και περισσότερο χρόνο για να τερματίσει. Αν παρατηρήσετε πως κάτι τέτοιο συμβαίνει και στον δικό σας αλγόριθμο, μπορείτε να δοκιμάσετε βελτιωμένους αλγόριθμους υπολογισμού της δύναμης ενός ακεραίου [20, 17].

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.