ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

# Λύσεις $3^{ης}$ Σειράς Ασκήσεων για το μάθημα “Υπολογιστική Κρυπτογραφία”

*Δημήτριος Βασιλείου*
*03119830*
*$9^{ο}$ Εξάμηνο*

# ΑΣΚΗΣΗ 1

Για να είναι το πρόβλημα $RSA$ random-self reducible πρέπει αν γνωρίζουμε τη λύση για ένα οποιοδήποτε τυχαίο στιγμιότυπο του προβλήματος $c'$, να μπορούμε να το λύσουμε και για το $c$. Επιλέγουμε τυχαίο $\alpha \in Z_n$ και υπολογίζουμε $c' = a^e c \bmod n$. Γνωρίζουμε τη λύση για $c'$, άρα γνωρίζουμε $m'$ τέτοιο ώστε $m'^e = c' \bmod n$. Είναι $m'^e \equiv c \equiv a^e c \equiv a^e m^e \Rightarrow m'^e a^{-e} \equiv m^e \Rightarrow m \equiv a^{-1} m' (\bmod n)$. Συνεπώς, βρίσκουμε το $m$ και λύνουμε το πρόβλημα $RSA$ για το στιγμιότυπο $c$, συνεπώς είναι random-self reducible.

# ΑΣΚΗΣΗ 2

**Απόδειξη ότι $SDH \Leftrightarrow IDH$**

- Θα δείξουμε πρώτα ότι $IDH \leq SDH$, δηλαδή αν μπορούμε να λύσουμε το $SDH$, λύνουμε και το $IDH$. Μπορούμε να θεωρήσουμε ότι έχουμε ένα μαντείο $O$ που δεδομένων $g, g^x$ μας επιστρέφει το $g^{x^2}$. Ρωτάμε το μαντείο για το ζεύγος $g^r, g$ όπου $g^r$ τυχαίο και μας απαντάει με $O(g^r, g) = O\left(g^r, (g^r)^{r^{-1}}\right) = g^{rr^{-2}} = g^{r^{-1}}$. Άρα, υπολογίσαμε το $g^{r^{-1}}$ και λύσαμε το $IDH$.
- Θα δείξουμε τώρα ότι $SDH \leq IDH$, δηλαδή αν μπορούμε να λύσουμε το $IDH$, λύνουμε και το $SDH$. Θεωρούμε πάλι μαντείο $O$ που δεδομένων $g, g^x$ μας επιστρέφει $O(g, g^x) = g^{x^{-1}}$. Ρωτάμε το μαντείο για το ζεύγος $g^r, g$ όπου $g^r$ τυχαίο και μας απαντάει με $O(g^r, g) = O\left(g^r, (g^r)^{r^{-1}}\right) = (g^r)^{(r^{-1})^\wedge -1} = g^{r^2}$. Άρα υπολογίσαμε το $g^{r^2}$ και λύσαμε το $SDH$.
- Τελικά $SDH \Leftrightarrow IDH$ αφού το ένα ανάγεται στο άλλο.

**Απόδειξη ότι $SDH \Leftrightarrow CDH$ και $CDH \Leftrightarrow SDH$**

- Θα δείξουμε ότι $SDH \leq CDH$. Δηλαδή αν έχουμε μαντείο $O$ που για είσοδο $g, g^x, g^y$ επιστρέφει $g^{xy}$, υπάρχει αλγόριθμος $A$ που με είσοδο $g, g^x$ υπολογίζει το $g^{x^2}$. Θεωρούμε ότι ο $A$ παίρνει είσοδο $g, g^r$ όπου $g^r$ τυχαίο. Υπολογίζει τυχαίες τιμές $t_1, t_2 \in Z_q$ και ρωτάει το μαντείο για $O(g, g^{rt_1}, g^{rt_2}) = g^{r^2 t_1 t_2} = g^{r^2} g^{t_1} g^{t_2}$. Αφού $g, t_1, t_2$ γνωστά, ο $A$ υπολογίζει σε αμελητέο χρόνο το $g^{r^2}$ και το επιστρέφει, συνεπώς λύνει το $SDH$.
- Θα δείξουμε τώρα ότι $CDH \leq SDH$. Δηλαδή αν έχουμε μαντείο $O$ που για είσοδο $g, g^x$ επιστρέφει $g^{x^2}$, υπάρχει αλγόριθμος $A$ που με είσοδο $g, g^x, g^y$ υπολογίζει το $g^{xy}$. Ο $A$ επιλέγει τυχαία $s_1, s_2, t_1, t_2 \in Z_q$ και ρωτάει το μαντείο για $O(g, g^{xs_1}) = g^{(xs_1)^2}, O(g, g^{ys_1}) = g^{(ys_1)^2}$ και $O(g, g^{xs_1t_1 + ys_2t_2}) = g^{(xs_1t_1 + ys_2t_2)^2} = g^{(xs_1t_1)^2 + 2xys_1s_2t_1t_2 + (ys_2t_2)^2} = g^{(xs_1)^2} g^{t_1^2} g^{2s_1s_2t_1t_2} g^{xy} g^{(ys_2)^2} g^{t^2}$. Όλα είναι γνωστά εκτός του $g^{xy}$ το οποίο υπολογίζεται λύνοντας την εξίσωση, καθώς όλοι οι υπόλοιποι όροι όπως και η έξοδος του μαντείου, είναι γνωστοί. Άρα ο $A$ υπολογίζει το $g^{xy}$ συνεπώς λύνει το $CDH$.
- Τελικά $SDH \Leftrightarrow CDH$ και αφού $SDH \Leftrightarrow IDH$ και αφού $SDH \Leftrightarrow IDH$ τότε είναι $SDH \Leftrightarrow CDH \Leftrightarrow IDH$

# ΑΣΚΗΣΗ 3

Παραθέτουμε τον κώδικα για την επίλυση της ασκήσεως σε γλώσσα Python:

```python
import rsa

def rsa_encrypt(e, m, n):
    return pow(m, e, n)

def oracle(d, c, n):
    m = pow(c, d, n)
    location = 0
    if m % n > (n // 2):
        location = 1
    else:
```

```
        location = 0
    return location

def attack(e, d, c, n):
    low = 0
    high = n
    prev_high = high
    prev_low = low
    location = oracle(d, c, n)
    i = 1
    while low <= high:
        if location == 0:
            prev_high = high
            high = high - (n // (2 ** i))

        elif location == 1:
            prev_low = low
            low = low + (n // (2 ** i))

        location = oracle(d, c * rsa_encrypt(e, 2 ** i, n), n)
        i += 1
        if (prev_high == high and prev_low == low):
            break

    for m in range(low, high):
        if c == pow(m, e, n):
            print("Message generated from the attack: " + str(m))
            break

def program():
    keys = rsa.newkeys(128)
    n = keys[0].n
    e = keys[0].e
    d = keys[1].d
    print("n = " + str(n) + "\n" + "e = " + str(e) + "\n" + "d = " + str(d) + "\n")
    m = int(input("Give Message: "))
    print("Original Message: " + str(m))
    c = rsa_encrypt(e, m, n)
    attack(e, d, c, n)

program()
```

Ένα output της εκτέλεσης είναι το ακόλουθο:

```
n = 189164821258568782059014026194145564687
e = 65537
d = 39263455202104324883490805045167664273

Give Message: 20345
Original Message: 20345
Message generated from the attack: 20345
```

**Ανάλυση του κώδικα:**

- Η συνάρτηση `rsa_encrypt(e, m, n)` υλοποιεί την κρυπτογράφηση ενός μηνύματος εκτελώντας την πράξη $c = m^e \, mod \, n$.
- Η συνάρτηση `oracle(d, c, n)` προσομοιώνει το μαντείο και υλοποιεί τη συνάρτηση $loc$ αποκρυπτογραφώντας το `c`. Συγκεκριμένα επιστρέφει 1 αν
$$m \, mod \, n > \frac{n}{2}$$
Αλλιώς επιστρέφει 0.
- Η συνάρτηση `attack(e, d, c, n)` υλοποιεί την επίθεση. Ορίζουμε αρχικά ως high το n και ως low το 0. Μέχρι το low να φτάσει το high ελέγχουμε κάθε φορά, καλώντας την oracle, αν το location

είναι 0. Αν είναι 0, αναθέτουμε ως high την τιμή $high - (\frac{n}{2^i})$ δηλαδή μειώνουμε το άνω άκρο του διαστήματος, αλλιώς αναθέτουμε στο low την τιμή $low + (\frac{n}{2^i})$ δηλαδή αυξάνουμε το κάτω άκρο του διαστήματος. Μετά, καλούμε πάλι την oracle και υπολογίζουμε το νέο location. Εδώ ωστόσο κάνουμε την εξής παρατήρηση. Ο αλγόριθμος λέει ότι η oracle πρέπει κάθε φορά να δέχεται την κρυπτογράφηση του $m, 2m, 4m, 8m, \dots 2^i m, \dots$, συνεπώς εκμεταλλευόμαστε το γεγονός ότι $Encrypt(m2^i) = Encrypt(m)Encrypt(2^i)$. Για αυτό τον λόγο η νέα κλήση στη συνάρτηση location γίνεται ως `location = oracle(d, c * rsa_encrypt(e, 2 ** i, n), n)`. Σημειώνουμε επίσης ότι υπάρχει περίπτωση το low ποτέ να μην γίνει ίσο με high και να πέσουμε σε ατέρμονα βρόχο. Για αυτό το λόγο ορίζουμε τις μεταβλητές prev_high και prev_low οι οποίες ξεκινούν ίσες με high και low αντίστοιχα. Αν στο τέλος κάποιας επανάληψης, το prev_high ισούται με high και το prev_low ισούται με low σημαίνει πως δεν μπορούμε να περιορίσουμε άλλο το διάστημα πιθανών λύσεων, οπότε σταματάμε την εκτέλεση.

- Σαν τελικό βήμα, διατρέχουμε όλες τις τιμές που βρίσκονται στο διάστημα low, high και όποια τιμή ικανοποιεί τη σχέση $c = m^e \ mod \ n$ είναι το αρχικό κείμενο.
- Σημειώνουμε ότι για την παραγωγή των κλειδιών χρησιμοποιήσαμε τη βιβλιοθήκη rsa της Python.

## ΑΣΚΗΣΗ 4

**1).** Θα δείξουμε ότι $(c^e)^{t-1} \equiv m \ (mod \ n)$ οπότε γνωρίζοντας έναν RSA κύκλο βρίσκουμε το $m$. Πράγματι, είναι $(c^e)^{t-1} \equiv m \Leftrightarrow ((c^e)^{t-1})^e \equiv m^e \Leftrightarrow c^{e^t} = c$ που ισχύει.

**2).** Έστω $c = g^a \ mod \ n$ όπου $g$ ένας γεννήτορας του $Z_n$. Ένας RSA κύκλος θα είναι ο

$$c, c^e, c^{e^2}, \dots, c^{e^t} \text{ ή } g^a, g^{ae}, \dots, g^{ae^t}$$

Άρα για να υπάρχει δηλαδή για να είναι $c^{e^t} \equiv c \ (mod \ n)$, πρέπει $e^t \equiv 1 \ (mod \ n)(1)$. Γνωρίζουμε ότι $n = pq$ άρα οι μόνοι διαιρέτες του $n$ είναι οι $1, p, q, n$. Άρα $gcd(e, n) = 1$ οπότε από θεώρημα Euler θα έχουμε: $e^{\varphi(n)} \equiv 1 \ mod \ n$. Επομένως πάντα υπάρχει $t = \varphi(n)$ για να ικανοποιείται η $(1)$. Συνεπώς, υπάρχει RSA κύκλος για κάθε $c$.

**3).** Ένα άνω φράγμα είναι το $t = \varphi(n)$ διότι για το οποίο υπάρχει RSA κύκλος όπως δείξαμε παραπάνω.

**4).** Θα δείξουμε ότι αν το μήκος του RSA κύκλου δεν είναι πολυωνυμικά φραγμένο, τότε μπορούμε να ανακτήσουμε το $m$ και να αντιστρέψουμε τη συνάρτηση RSA.
Έστω ότι το μήκος του RSA κύκλου δεν είναι πολυωνυμικά φραγμένο. Τότε, μετά από πολυωνυμικό μήκος «υψώσεων» στην $e \ mod \ n$ θα φτάσουμε στο $(c^e)^{t-1} \equiv m \ (mod \ n)$ (όπως δείξαμε στο $(1)$). Άρα μπορούμε να αντιστρέψουμε τη συνάρτηση RSA «εύκολα», άτοπο. Άρα το μήκος RSA κύκλου δεν είναι πολυωνυμικό.

## ΑΣΚΗΣΗ 5

Σημειώνουμε ότι για την επίλυση της άσκησης βασιστήκαμε στην ακόλουθη μεταπτυχιακή εργασία:
[https://people.tamu.edu/~rojas//rex.pdf](https://people.tamu.edu/~rojas//rex.pdf)

Η διαίσθησή μας, λέει ότι αφού η εύρεση διακριτού λογαρίθμου είναι δύσκολο πρόβλημα, το ίδιο θα συμβαίνει και για τον προσδιορισμό οποιουδήποτε bit του διακριτού λογαρίθμου. Ωστόσο, για κάποια bits ο προσδιορισμός τους είναι στα αλήθεια «εύκολος». Θα ξεκινήσουμε εξετάζοντας το τελευταίο bit του διακριτού λογαρίθμου.

**Λήμμα 6.1.3:**
Έστω $g$ ένας γεννήτορας του $Z_p^*$ και έστω $\alpha$ ένα τετραγωνικό υπόλοιπο στο $Z_p^*$ με $a \equiv g^{2r} mod\, p$ για τον ακέραιο $r$ στο διάστημα $0 \le r < \frac{p-1}{2}$. Τότε $x = g^r\ mod\ p$ και $y = g^{r+\frac{p-1}{2}}\ mod\ p$ είναι οι δύο τετραγωνικές ρίζες του $a\ mod\ p$.
**Απόδειξη:**

$$x^2 \equiv g^{2r} \equiv a \equiv g^{2r} \cdot 1 \equiv g^{2r} \cdot g^{p-1} \equiv g^{2r+(p-1)} \equiv y^2 \ \ (mod\ p)$$

**Πόρισμα 6.1.4:**
Αν $x$ είναι η κύρια τετραγωνική ρίζα του $a$ τότε $\log_g x$ είναι το δεξί shift του $\log_g a$

Σημειώνουμε ότι υπάρχουν PPT αλγόριθμοι που υπολογίζουν τις τετραγωνικές ρίζες στο $Z_p^*$.

**Το τελευταίο bit (1)**
Είναι προφανές πως αν το το $h$ είναι τετραγωνικό υπόλοιπο τότε υπάρχει ακέραιος $r$ με $2r \in Z_{p-1}$ τέτοιος ώστε $h \equiv g^{2r}\ mod\ p$. Συνεπώς, είναι $\log_g h = 2r$ άρα αφού ο λογάριθμος είναι άρτιος, με βάση το σύμβολο $Legendre$, το τελευταίο του bit θα είναι 0. Επομένως, υπολογίζοντα μόνο το $(\frac{h}{p})$ μπορούμε με σιγουριά να αποφανθούμε για το τελευταίο bit.

**Μετατροπή τετραγωνικών μη υπολοίπων σε τετραγωνικά υπόλοιπα (2)**
Όπως είδαμε, αν $h \equiv g^r\ mod\ p$ είναι τετραγωνικό μη υπόλοιπο, τότε ο $r$ είναι περιττός και το 0-bit του $r$ είναι 1. Το $h$ μπορεί να μετατραπεί σε τετραγωνικό υπόλοιπο πολλαπλασιάζοντας το με $g^{-1}$, δηλαδή $h' = hg^{-1} = g^{r-1}(mod\ p)$ με $r-1$ άρτιο, αφού $r$ περιττός. Άρα, το $h'$ είναι τετραγωνικό υπόλοιπο αφού μπορεί να γραφεί ως $h' = g^{2x}\ mod\ p$ με $2x = r - 1$. Θα χρησιμοποιήσουμε αυτή τη διαδικασία στη συνέχεια.

**Το Bit Border (3)**
Δοθέντος ενός πρώτου $p$, μπορεί να γραφεί ως $p = 2^s k + 1$, όπου $k$ είναι περιττός και $s$ ακέραιος $> 0$. Θα ονομάσουμε ως **bit border** τον αριθμό $s$. Βλέπουμε ότι:

$$p - 1 = 2^s k \Rightarrow \frac{p-1}{2} = 2^{s-1} k$$

Συνεπώς,

$$2^{s-1} | \frac{p-1}{2}$$

Αυτό μας δείχνει ότι τα πρώτα $s-1$ bits της δυαδικής αναπαράστασης του $p - 1/2$ είναι όλα μηδέν. Έστω $h \equiv g^{2r}\ mod\ p$ ένα τετραγωνικό υπόλοιπο και έστω $x = g^r\ mod\ p$ και $y = g^{r+\frac{p-1}{2}}\ mod\ p$ οι δύο ρίζες του $h$ (βλέπε Λήμμα 6.1.3). Τότε, $\log_g x = r$ και $\log_g y = r + p - 1/2$. Είναι $|\log_g x - \log_g y| = p - 1/2 = 2^{s-1} k$, δηλαδή η διαφορά των δύο διακριτών λογαρίθμων είναι δυαδικός αριθμός που τελειώνει με $s-1$ μηδενικά, Αυτό μας δείχνει ότι τα $\log_g x$ και $\log_g y$ διαφέρουν πρώτη φορά στο s-οστό bit τους. Συνεπώς τα $\log_g x$ και $\log_g y$ μπορούν να γίνουν shift $s-1$ φορές μέχρι το LSB τους να διαφέρει. Αυτή η παρατήρηση είναι χρήσιμη στο να χρησιμοποιήσουμε την ίδια τακτική με το μηδενικό bit.

**Επεκτείνοντας την ιδέα για το 0-bit**
Με βάση το Πόρισμα 6.1.4 ο υπολογισμός της κύριας ρίζας του $h \equiv g^r\ mod\ p$ είναι ισοδύναμος με ολίσθηση του εκθέτη $r$ κατά 1 bit. Συνεπώς μπορούμε να σκεφτούμε τον ακόλουθο αλγόριθμο για τον υπολογισμό του εκθέτη $r$:
Κάθε φορά παίρνουμε το 0-bit του $r$ και το υπολογίζουμε όπως κάναμε στο μηδενικό bit (με σύμβολο Legendre). Μετά, μετατρέπουμε το $h$ σε τετραγωνικό υπόλοιπο με βάση την διαδικασία που περιγράψαμε προηγουμένως και τέλος θέτουμε ως $h$ τη ρίζα του $h$. Εφαρμόζοντας αυτόν τον αλγόριθμο, θεωρητικά θα

μπορούσαμε να υπολογίσουμε όλα τα bits του διακριτού λογαρίθμου. Ωστόσο, δεν ξέρουμε ποια είναι η κύρια ρίζα του $h \equiv g^r \bmod p$. Όμως, σίγουρα τα πρώτα $s-1$ bits των δύο ριζών είναι ίδια όπως δείξαμε προηγουμένως, επομένως δεν έχει σημασία ποια ρίζα θα χρησιμοποιήσουμε για τα πρώτα $s-1$ bits. Συνεπώς, μπορούμε να υπολογίσουμε τα $s-1$ πρώτα bits του διακριτού λογαρίθμου.

**Τα «δύσκολα» bits:**

Αποδεικνύεται ότι τα υπόλοιπα bits είναι «δύσκολο» να υπολογιστούν, καθώς δεν υπάρχει PPT που να οδηγεί στον υπολογισμό τους.

## ΑΣΚΗΣΗ 6

**1).** Ορίζουμε τη συνάρτηση αποκρυπτογράφησης ως $Dec(sk, m) = Dec(x, m) = ((g^{rx})^{-1}(g^r h^m)^x)^{\frac{1}{x}} = h^m$. Μετά υπολογίζεται ο διακριτός λογάριθμος και ανακτάται το μήνυμα $m$. Η ορθότητα της συνάρτησης είναι προφανής.

**2).** Θα ονομάσουμε για αυτό το ερώτημα το τροποποιημένο El Gamal ως $EGV$ $(El\ Gamal\ Variation)$.

**IND-CPA:**

Θα δείξουμε ότι αν το $DDHP$ είναι δύσκολο στην $G$, τότε το κρυπτοσύστημα $EGV$ διαθέτει $IND-CPA$.

Έστω ότι το $EGV$ δεν διαθέτει $IND-CPA$. Άρα υπάρχει αντίπαλος $A$ ο οποίος μπορεί να νικήσει το παιχνίδι $CPA$ με μη αμελητέα πιθανότητα. Κατασκευή $B$:

- Είσοδος: τριάδα στοιχείων
- Εσωτερικά: Προσομοίωση του $C_{IND-CPA}$ στο παιχνίδι $CPA$ και χρήση $A$ ως black box
- Αποτέλεσμα: Διαχωρισμός $DH$-τυχαίας τριάδας με μη αμελητέα πιθανότητα.

Το παιχνίδι δουλεύει ως εξής:

- Είσοδος $g^a, g^\beta, g^c$
- Στο $IND-CPA$ public key το $g^a$
- Ο $B$ προσομοιώνει το $IND-CPA$
- Όταν ο $A$ προκαλέσει με $m_0, m_1$:
  - Ο $C_{IND-CPA}$ δηλαδή ο $B$, διαλέγει $b$ τυχαία από το $\{0,1\}$
  - Ο $C_{IND-CPA}$ δηλαδή ο $B$, επιλέγει $r=\beta$ και στέλνει το $c = (pk^r, g^r h^m) = (g^{\alpha\beta}, g^\beta h^{m_b})$
- Ο A επιστρέφει $b'$
- Ο B επιστρέφει $b'$

Αναλύουμε τώρα το παιχνίδι: Για τριάδα $DH$ θα είναι $g^c = g^{\alpha\beta}$. Ο $A$ λαμβάνει έγκυρο κρυπτοκείμενο $EGV$. Επειδή $c = (pk^r, g^r h^{m_b}) = (g^{ar}, g^r h^{m_b}) \overset{r=\beta}{\Longrightarrow} c = (g^{\alpha\beta}, g^\beta h^{m_b})$. Ο $A$ υπολογίζει τα $h^{m_0}$, και $h^{m_1}$ και μπορεί με πιθανότητα $1$ να αποφασίσει ποιο μήνυμα κρυπτογραφήθηκε. Άρα με πιθανότητα $1$ καταλαβαίνει και ο $B$ ότι η τριάδα είναι $DH$. Αν η τριάδα είναι τυχαία, τότε ο $A$ μαντεύει τυχαία επειδή το κρυπτοκείμενο δεν είναι έγκυρο οπότε η πιθανότητα επιτυχίας του είναι $1/2$, ομοίως και του $B$. Συνεπώς το πλεονέκτημα του $B$ είναι $1 - 1/2 = 1/2$, δηλαδή μη αμελητέο. Συνεπώς, κατασκευάσαμε $B$ που με μη αμελητέα πιθανότητα λύνει το $DDHP$, άτοπο. Τελικά το $EGV$ διαθέτει ασφάλεια $IND-CPA$.

**IND-CCA:**

Θεωρούμε τον challenger $Chal$ και έναν αντίπαλο $A$ οι οποίοι παίζουν το παιχνίδι $IND-CCA$ ως εξής:

- Ο $A$ στέλνει μήνυμα $m'$ και ο $Chal$ απαντάει με $c' = (g^{xr'}, g^{r'} h^{m'})$.
- Ο $A$ στέλνει δύο μηνύματα $m_0, m_1$ και ο $Chal$ απαντάει με το challenge $c = (g^{xr}, g^r h^{m_b})$
- Ο $A$ ρωτάει για το $c'' = \left(g^{xr} g^{xr'}, a g^r g^{r'} h^{m_b + m'}\right) = \left(g^{x(r+r')},\ a g^{r+r'} h^{m_b+m'}\right)$, όπου το $\alpha \in G$ επιλέγεται από τον Ο $A$

- Ο $Chal$ απαντάει με $m'' = Dec(c'') = ((g^{x(r+r')})^{-1}(ag^{r+r'}h^{m_b+m'})^x)$^$(1/x) = ah^{m_b+m'}$. Διαιρώντας με το γνωστό $\alpha$, ο $A$ κατέχει το $h^{m_b+m'}$.
- Ο $A$ υπολογίζει τα $h^{m_0+m'}, h^{m_1+m'}$ και ανάλογα με το ποιο ισούται με $h^{m_b+m'}$, επιλέγει και επιστρέφει το $b'$. Είναι προφανές ότι με μη αμελητέα πιθανότητα ο $A$ επιλέγει σωστά, οπότε νικάει το παιχνίδι $IND-CCA$ και τελικά το $EGV$ δεν διαθέτει ασφάλεια $IND-CCA$.

**OW-CPA:**
Για να δείξουμε ότι το $EGV$ διαθέτει την ιδιότητα $OW-CPA$ θεωρούμε ότι έχουμε ένα μαντείο που σπάει το κρυπτοσύστημα δίχως γνώση του ιδιωτικού κλειδιού. Θα δείξουμε ότι με τέτοιο μαντείο μπορούμε να λύσουμε το $CDHP$.

- Είσοδος στο μαντείο: $g^{x_1}(pk),\ c = (g^{x_2}, a)$
- Έξοδος: $h^m$ τέτοιο ώστε $\alpha = g^{x_2/x_1}h^m \Rightarrow g^{x_2/x_1} = ah^{-m}$

Τα $a, h^{-m}$ είναι και τα δύο γνωστά συνεπώς υπολογίσαμε το $g^{x_2/x_1}$ άρα λύσαμε $DCDHP(Divisivle\ Computanional\ Diffie\ Hellman\ Problem)$ το οποίο είναι ισοδύναμο με το $CDHP$. Αφού λοιπόν με το μαντείο λύνουμε το $CDHP$, το $EGV$ διαθέτει την ιδιότητα $OW-CPA$.

## ΑΣΚΗΣΗ 7

**1).**

**Πληρότητα:**
Το πρωτόκολλο έχει πληρότητα, διότι ένας τίμιος Prover πείθει έναν τίμιο Verifier με πιθανότητα 1. Αυτό, επειδή:

- $g^{s_1}h^{s_2} = g^{t_1+em}h^{t_2+em}$
- $tc^e = g^{t_1}h^{t_2}g^{me}h^{re} = g^{t_1+em}h^{t_2+er}$

Δηλαδή ισχύει ότι $g^{s_1}h^{s_2} = tc^e$

**Ειδική Ορθότητα:**
Έστω δύο επιτυχείς εκτελέσεις του πρωτοκόλλου με διαφορετικό challenge, δηλαδή $(t, e, s_1, s_2)$ και $(t, e', s_1', s_2')$. Είναι

$$\begin{cases} g^{s_1}h^{s_2} = tc^e \Rightarrow t = g^{s_1}h^{s_2}c^{-e} \\ g^{s_1'}h^{s_2'} = tc^e \Rightarrow t = g^{s_1'}h^{s_2'}c^{-e'} \end{cases}$$

Άρα είναι

$$g^{s_1}h^{s_2}c^{-e} = g^{s_1'}h^{s_2'}c^{-e'} \Rightarrow g^{s_1}h^{s_2}g^{-em}h^{-er} = g^{s_1'}h^{s_2'}g^{-e'm}h^{e'r} \Rightarrow g^{-em+e'm}h^{-er+e'r} = \frac{g^{s_1'}h^{s_2'}}{g^{s_1}h^{s_2}} \Rightarrow$$

$$g^{m(e'-e)}h^{r(e'-e)} = g^{s_1'-s_1}h^{s_2'-s_2} \Rightarrow$$

$$\begin{cases} m(e'-e) = s_1' - s_1 \\ r(e'-e) = s_2' - s_2 \end{cases} \Rightarrow \begin{cases} m = \dfrac{s_1' - s_1}{e' - e} \\ r = \dfrac{s_2' - s_2}{e' - e} \end{cases}$$

Συνεπώς, ανακτάται ο witness $(m, r)$ άρα το πρωτόκολλο έχει ειδική ορθότητα.

**Μηδενική Γνώση:**
Για μηδενική γνώση θεωρούμε τίμιο Verifier. Έστω Simulator $S$ που δεν γνωρίζει το witness $(m, r)$, και τίμιος verifier $V$.

- Αρχικά, ο $S$ επιλέγει τυχαία $t_1, t_2 \in Z_q^*$ και στέλνει στον $V$ το $t = g^{t_1}h^{t_2}$
- Ο $V$ επιλέγει τυχαίο $e \in Z_q^*$ και το στέλνει στον $S$

- Αν ο $S$ μπορεί να απαντήσει (αμελητέα πιθανότητα), το πρωτόκολλο συνεχίζει κανονικά
- Αλλιώς, γίνεται rewind ο $V$
- Στη δεύτερη περίπτωση, ο $S$ δεσμεύεται στο $t = g^{t_1}h^{t_2}c^{-e}$
- Ο $V$ επιλέγει ίδιο $e \in Z_q^*$ (ίδιο random tape)
- Ο $S$ στέλνει $s_1 = t_1$ και $s_2 = t_2$
- Ο $V$ θα δεχτεί αφού $tc^e = g^{t_1}h^{t_2}c^{-e}c^e = g^{t_1}h^{t_2} = g^{s_1}h^{s_2}$

Η συζήτηση του $S$ ακολουθεί ίδια κατανομή με τη συζήτηση ενός Prover $P$, συνεπώς το πρωτόκολλο διαθέτει μηδενική γνώση για τίμιους verifiers

Τελικά, το $\Pi$ είναι Σ-πρωτόκολλο καθώς διαθέτει πληρότητα, ειδική ορθότητα και μηδενική γνώση.

**2).**
Θα δείξουμε ότι το $\Pi$ είναι witness indistinguishable. Για witness $(m,r)$ έχουμε τη συζήτηση $(t,e,s_1,s_2)$ με $t_1,t_2 \in Z_q^*$ και για witness $(m',r')$ έχουμε τη συζήτηση $(t',e',s_1',s_2')$ με $t_1',t_2' \in Z_q^*$. Παρατηρούμε πως υπάρχει μοναδικό ζεύγος $(t_1',t_2')$ με $t_1 \neq t_1', t_2 \neq t_2'$ το οποίο δίνει ίδια συζήτηση, δηλαδή τέτοιο ώστε $t = t', e = e', s_1 = s_1', s_2 = s_2'$. Συγκεκριμένα, αν $t_1' = t_1 + e(m-m'),\ t_2' = t_2 + e(r-r')$ θα είναι:

$$t' = g^{t_1'}h^{t_2'} = g^{t_1+em-em'}h^{t_2+er-er'} = g^{t_1}h^{t_2}g^{em-em'}h^{er-er'} = g^{t_1}h^{t_2}\frac{g^{em}h^{er}}{g^{em'}h^{er'}} = t\frac{c^e}{c^e} = t$$

Και

$$s_1' = t_1' + em' = t_1 + em - em' + em' = t_1 + em = s_1, \text{ όμοια είναι } s_2 = s_2'$$

Άρα, οποιαδήποτε συζήτηση δεν μπορεί να οδηγήσει σε διάκριση μεταξύ δύο witness, αυτών που πραγματικά χρησιμοποιήθηκαν. Έστω τώρα ότι ένας $V$ καταφέρνει να εξάγει witness $(m',r')$ μετά από πολυωνυμικό αριθμό αλληλεπιδράσεων με τον $P$. Τότε όμως θα σημαίνει ότι

$$c = g^m h^r = g^{m'}h^{r'} \Rightarrow g^{m-m'} = h^{r'-r} \Rightarrow \log_g h = \frac{m-m'}{r'-r}$$

Που σημαίνει ότι ο $V$ λύνει το $DLOG$ για δύο τυχαία στοιχεία του $G$, άτοπο.
Τελικά, το $\Pi$ είναι witness indistinguishable.

**3).** Ο Verifier πρέπει να ελέγξει ότι ισχύει η σχέση $g^{s_1}h^{s_2} = abc^e \Leftrightarrow g^{t_1+em}h^{t_2+er} = g^{t_1}h^{t_2}(g^m h^r)^e \Leftrightarrow g^{t_1+em}h^{t_2+er} = g^{t_1+em}h^{t_2+er}$.
Ελέγχουμε τώρα αν το $\Pi'$ είναι Σ-πρωτόκολλο, δηλαδή αν έχει τις ιδιότητες της πληρότητας, της ειδικής ορθότητας και της μηδενική γνώσης.

**Πληρότητα:**

$$\begin{cases} g^{s_1}h^{s_2} = g^{t_1+em}h^{t_2+er}, \\ abc^e = g^{t_1}h^{t_2}g^{me}h^{re} = g^{t_1+em}h^{t_2+er}, \end{cases}$$

Οι δύο όροι είναι ίσοι, άρα το πρωτόκολλο έχει πληρότητα.

**Ειδική Ορθότητα:**
Έστω δύο επιτυχείς εκτελέσεις του πρωτοκόλλου με διαφορετικό challenge, δηλαδή $(a,b,e,s_1,s_2)$ και $(a,b,e',s_1',s_2')$. Είναι

$$\begin{cases} g^{s_1}h^{s_2} = abc^e \Rightarrow ab = g^{s_1}h^{s_2}c^{-e} \\ g^{s_1'}h^{s_2'} = abc^{e'} \Rightarrow ab = g^{s_1'}h^{s_2'}c^{-e'} \end{cases}$$

Ακριβώς με την ίδια λογική στο ερώτημα **(α).** δείχνουμε ότι το $\Pi'$ έχει ειδική ορθότητα

**Μηδενική Γνώση:**
Για μηδενική γνώση θεωρούμε τίμιο Verifier. Έστω Simulator $S$ που δεν γνωρίζει το witness $(m,r)$, και τίμιος verifier $V$.

- Αρχικά, ο $S$ επιλέγει τυχαία $t_1, t_2 \in Z_q^*$ και στέλνει στον $V$ τα $\alpha = g^{t_1}$, $b = h^{t_2}$

- Ο $V$ επιλέγει τυχαίο $e \in Z_q^*$ και το στέλνει στον $S$
- Αν ο $S$ μπορεί να απαντήσει (αμελητέα πιθανότητα), το πρωτόκολλο συνεχίζει κανονικά
- Αλλιώς, γίνεται rewind ο $V$
- Ο $V$ επιλέγει ίδιο $e \in Z_q^*$ (ίδιο random tape)
- Στη δεύτερη περίπτωση, ο $S$ δεσμεύεται στα $\alpha = g^{t_1}$, $b = g^{t_2}c^{-e}$
- Ο $S$ στέλνει $s_1 = t_1$ και $s_2 = t_2$
- Ο $V$ θα δεχτεί αφού $abc^e = g^{t_1}h^{t_2}c^{-e}c^e = g^{t_1}h^{t_2} = g^{s_1}h^{s_2}$

Η συζήτηση του $S$ ακολουθεί ίδια κατανομή με τη συζήτηση ενός Prover $P$, συνεπώς το πρωτόκολλο διαθέτει μηδενική γνώση για τίμιους verifiers

Τελικά, το $\Pi'$ είναι Σ-πρωτόκολλο καθώς διαθέτει πληρότητα, ειδική ορθότητα και μηδενική γνώση.

## ΑΣΚΗΣΗ 8

Δημοσιοποιείται ο $g$ που είναι γεννήτορας της (υπο)ομάδας τάξης $q$ του $Z_p^*$.

Ο $P^*$ κάνει τα εξής:

- Τυχαία επιλογή $t$ από το $Z_q$
- Υπολογισμός $y = g^t \bmod p$
- Υπολογισμός $c = H(y)$ όπου $H$ hash function που δίνει τιμές στο $Z_q$
- Τυχαία επιλογή $s$ από το $Z_q$
- Δημοσιοποίηση του $(c, s)$
- Επαλήθευση από οποιονδήποτε: $c = H(g^s h^{-c})$

Ο malicious prover $P^*$ δημοσιοποιεί το $h = g^{\frac{s-t}{c}}$. Για το συγκεκριμένο $h$ έχουμε:

- $H(y) = H(g^t)$
- $H(g^s h^{-c}) = H\left(g^s \left(g^{\frac{s-t}{c}}\right)^{-c}\right) = H\left(g^s \Big/ g^{\frac{s-t}{c}c}\right) = H(g^t)$

Άρα $H(y) = H(g^s h^{-c})$ οπότε επαληθεύεται, χωρίς στα αλήθεια ο $P^*$ να γνωρίζει το διακριτό λογάριθμο, καθώς τον κατασκευάζει και τον δημοσιοποιεί εκ των υστέρων. Βασικά, ο $P^*$ δεν παίζει δίκαια καθώς πρώτα βρίσκει τα $s, t, c$ και μετά δημοσιοποιεί το $h$. Υπολογίζοντας $c = H(h||y)$ θα έπρεπε αναγκαστικά ο $P^*$ να έχει υπολογίσει πρώτα το $h$ οπότε δεν θα μπορούσε να «χακάρει» το πρωτόκολλο.

## ΑΣΚΗΣΗ 9

Υλοποιήσαμε το σχήμα υπογραφών Schnorr. Παραθέτουμε τον κώδικα σε γλώσσα Python:

```python
from primality import primality
import random
import hashlib

def generate_p_q_g():
    p = 4
    i = 42
    r = 1
    # find p, q
    while not primality.isprime(p):
        q = primality.nthprime(i)
        r += 1
        p = q * r + 1
        i += 1
```

```
    # find h
    while True:
        h = random.randint(2, p - 1)
        if pow(h, r, p) != 1:
            break
    # find g
    g = pow(h, r, p)
    return p, q, g

def generate_keys(p, q, g):
    sk = random.randint(1, q - 1)
    pk = pow(g, sk, p)
    return pk, sk

def sign(p, q, g, sk, pk, m):
    t = random.randint(1, q - 1)
    T = pow(g, t, p)
    val = bin(T)[2:] + bin(pk)[2:] + m

    sha256_hash = hashlib.sha256()
    sha256_hash.update(val.encode('utf-8'))
    hashed_value = sha256_hash.digest()
    c = int.from_bytes(hashed_value, byteorder='big') % q

    s = (t + c * sk) % q
    return T, s

def verify(p, q, g, pk, T, s, m):
    val = bin(T)[2:] + bin(pk)[2:] + m

    sha256_hash = hashlib.sha256()
    sha256_hash.update(val.encode('utf-8'))
    hashed_value = sha256_hash.digest()
    c = int.from_bytes(hashed_value, byteorder='big') % q

    first = pow(g, s, p)
    second = (T * pow(pk, c, p)) % p
    print("g^s = " + str(first))
    print("T * pk^c = " + str(second))
    if first == second:
        print("Message Verified!")
    else:
        print("Oops...Malicious user!")

def program():
    p, q, g = generate_p_q_g()
    pk, sk = generate_keys(p, q, g)
    print("p: " + str(p))
    print("q: " + str(q))
    print("g: " + str(g))
    print("pk: " + str(pk))
    m = input("Give message: ")
    binary_m = ''.join(format(ord(x), '08b') for x in m)
    T, s = sign(p, q, g, sk, pk, binary_m)
    print("Signature (T, s) = (" + str(T) + ", " + str(s) + ")")
    verify(p, q, g, pk, T, s, binary_m)

program()
```

Ένα output της εκτέλεσης είναι το ακόλουθο:

```
p: 383
q: 191
g: 67
```

```
pk: 372
Give message: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus
placerat rutrum purus sit amet aliquam. Proin eu tortor ante. Mauris molestie eros
gravida quam tincidunt, eget sodales nunc faucibus. Nam mattis sed orci et semper.
Proin bibendum ultricies ante, vitae imperdiet dui viverra tincidunt. Nulla quam
justo, rutrum non lacus at, porta blandit metus. Vivamus elementum ornare pulvinar.
Etiam aliquet justo vel tincidunt dignissim. Vestibulum semper aliquet ligula sed
elementum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per
inceptos himenaeos. Maecenas lobortis feugiat quam ac tempor. Sed non posuere
mauris, et
pretium ligula. Vestibulum eget ex suscipit, scelerisque mi at, tempus quam. Nulla
maximus eros sapien, eget condimentum velit dignissim a. Suspendisse efficitur
cursus mollis. Cras luctus felis nunc, sed ultrices elit dignissim nec.
Signature (T, s) = (165, 83)
g^s = 73
T * pk^c = 73
Message Verified!
```

**Ανάλυση του κώδικα:**

Για τις υπογραφές Schnorr γνωρίζουμε πως χρειαζόμαστε έναν πρώτο $p$ και μια υποομάδα τάξης πρώτου $q$ του $Z_p$. Χρειαζόμαστε επίσης έναν γεννήτορα του $g$ του $Z_q$. Για να βρούμε αυτές τις παραμέτρους εύκολα χρησιμοποιήσαμε τη λογική που ακολουθείται στο ακόλουθο link:
https://en.wikipedia.org/wiki/Schnorr_group

- Η συνάρτηση generate_p_q_g() υπολογίζει ακολουθώντας την λογική αυτή τα $p, q, g$. Συγκεκριμένα επιλέγουμε ως $q$ τον 42-οστό σε σειρά πρώτο. Ξεκινάμε το $r$ από 1 και όσο ο αριθμός $p = qr + 1$ δεν είναι πρώτος, αλλάζουμε τον $q$ και αυξάνουμε κατά 1 το $r$. Μόλις βρεθούν οι $p, q$ προχωράμε στην εύρεση του $g$.Επιλέγουμε τυχαίο $h$ στο $Z_p$ μέχρις ότου βρούμε ένα ώστε $h^r \bmod p \neq 1$. Το $h^r \bmod p$ είναι ο γεννήτορας $g$.
- Η συνάρτηση generate_keys(p, q, g) επιστρέφει το ιδιωτικό κλειδί και το δημόσιο κλειδί.
- Οι συναρτήσεις sign(p, q, g, sk, pk, m) και verify(p, q, g, pk, T, s, m) υπογράφουν το μήνυμα και ελέγχουν αν είναι έγκυρη η υπογραφή αντίστοιχα, ακολουθώντας το σχήμα υπογραφών Schnorr.
- Σημειώνουμε ότι χρησιμοποιήσαμε τη συνάρτηση sha256 για το hashing, χρησιμοποιώντας τη βιβλιοθήκη hashlib της Python.

## ΑΣΚΗΣΗ 10

Θεωρούμε ότι η συνάρτηση $H$ είναι δημόσια διαθέσιμη. Για να δείξουμε ότι το σχήμα δεν προστατεύει από επίθεση καθολικής πλαστογράφησης πρέπει να βρούμε τριάδα $(m, a, b)$ που να ικανοποιεί τις δύο σχέσεις.

- ➔ Πρέπει να ισχύει: $g^{H(m)} yb \equiv 1 \ (mod\ p) \Leftrightarrow g^{H(m)} \equiv (yb)^{-1} (mod\ p)$ (1)
- ➔ Επίσης πρέπει να ισχύει $yb \equiv g^a \ (mod\ p) \overset{(1)}{\Rightarrow} g^{-H(m)} \equiv g^a \ (mod\ p) \Rightarrow a = -H(m)$

Για να ικανοποιείται η πρώτη σχέση, πρέπει $yb \equiv g^{-H(m)} \bmod p \Rightarrow b \equiv y^{-1} g^{-H(m)} \bmod p$ όπου $y^{-1}$, $g^{-H(m)}$ γνωστά. Άρα, επιλέγοντας για τυχαίο μήνυμα το $a = -H(m)$ και το $b \equiv y^{-1} g^{-H(m)} \bmod p$ ο αντίπαλος παράγει έγκυρη υπογραφή για τυχαίο μήνυμα δίχως γνώση του ιδιωτικού κλειδιού.