

Υπολογιστική Κρυπτογραφία

(ΣΗΜΜΥ, ΣΕΜΦΕ, ΑΛΜΑ, ΕΜΕ)

4η Σειρά Ασκήσεων

Προθεσμία παράδοσης: ημερομηνία εξέτασης

Άσκηση 1.

Σταθερό σημείο ενός κρυπτοσυστήματος ονομάζουμε ένα μήνυμα που το κρυπτοκείμενό του είναι το ίδιο το μήνυμα, δηλαδή $enc(m) = m$. Επομένως, στην περίπτωση του RSA, αν το δημόσιο κλειδί είναι το (N, e) , τότε για ένα σταθερό σημείο ισχύει $m^e \equiv m \pmod{N}$. Αποδείξτε ότι το πλήθος των σταθερών σημείων στο RSA είναι $[\gcd(e-1, p-1) + 1][\gcd(e-1, q-1) + 1]$.

Άσκηση 2.

Έστω το παρακάτω σχήμα υπογραφών όπου για τις παραμέτρους ισχύει ό,τι και στο σχήμα υπογραφών ElGamal. Κάθε χρήστης έχει ιδιωτικό κλειδί x και δημόσιο $y = g^x \pmod{p}$. Η υπογραφή λειτουργεί ως εξής:

i. Ο υπογράφων αρχικά επιλέγει $h \in \{0, \dots, p-2\}$ ώστε: $\mathcal{H}(m) + x + h \equiv 0 \pmod{p-1}$, όπου \mathcal{H} συνάρτηση σύνοψης ανθεκτική σε συγκρούσεις.

ii. Η υπογραφή είναι η τριάδα: $sign(x, m) = (m, (x + h) \pmod{p-1}, g^h \pmod{p})$.

iii. Για την επαλήθευση ότι μια τριάδα (m, a, b) είναι έγκυρη υπογραφή ελέγχεται εάν:

- $yb \equiv g^a \pmod{p}$ και
- $g^{\mathcal{H}(m)}yb \equiv 1 \pmod{p}$.

Να δείξετε ότι το σχήμα αυτό δεν προστατεύει από επίθεση καθολικής πλαστογράφησης.

Άσκηση 3. Να υλοποιήσετε σε γλώσσα προγραμματισμού της επιλογής σας την επίθεση αποκρυπτογράφησης ενός κρυπτοκειμένου c σε RSA που χρησιμοποιεί ένα oracle το οποίο μπορεί να αποφανθεί αν το μήνυμα που αντιστοιχεί στο κρυπτοκείμενο είναι στο 'πάνω' ή στο 'κάτω' μισό του \mathbb{Z}_n (δηλ. συνάρτηση loc - βλ. διάλεξη RSA - διαφάνειες 36–40).

Συγκεκριμένα πρέπει να υλοποιήσετε 2 προγράμματα, ή δύο τελείως χωριστά processes που να γίνονται fork από το ίδιο πρόγραμμα στην αρχή του:

(1) Το πρώτο θα 'προσομοιώνει' το oracle, αποκρυπτογραφώντας (κανονικά με το ιδιωτικό κλειδί) το c και υπολογίζοντας την loc .

(2) Το δεύτερο θα υλοποιεί την επίθεση ρωτώντας επαναληπτικά το oracle κατάλληλες ερωτήσεις για την loc .

Για την επικοινωνία των προγραμμάτων μπορείτε να χρησιμοποιήσετε οποιαδήποτε μορφή interprocess communication (RPC) γνωρίζετε, ή ακόμα και απλούστερη επικοινωνία μέσω ενός αρχείου ή pipe. Η παραγωγή των κλειδιών και η αρχική κρυπτογράφηση μπορεί να γίνει από δικό σας κώδικα ή χρησιμοποιώντας ένα έτοιμο εργαλείο όπως το Openssl.

Δείξτε την επίθεση με RSA **τουλάχιστον 2048 bits** (ή παραπάνω) και παραμετροποιήστε το/τα πρόγραμμα/τα σας ως προς το μήνυμα m (που θα είναι αρχικά γνωστό προφανώς στο πρόγραμμα 1, από το οποίο και θα υπολογίσει το c). Δοκιμάστε το με κάποιο συγκεκριμένο μήνυμα, αλλά θα ελεγχθεί η ορθότητά του και με άγνωστο σε εσάς μήνυμα. Συστήνεται η χρήση της βιβλιοθήκης GMP για τον χειρισμό των αριθμών.

Άσκηση 4. Να αποδείξετε ότι ένα σχήμα δέσμευσης δεν μπορεί να διαθέτει ταυτόχρονα τις ιδιότητες τέλειας δέσμευσης και τέλειας απόκρυψης.

Άσκηση 5. Υλοποιήστε το σχήμα υπογραφών Schnorr σε γλώσσα προγραμματισμού της επιλογής σας. Η υλοποίηση να βασιστεί στην υποομάδα τάξης q της ομάδας ακέραιων mod (safe) πρώτο $p = 2q + 1$ όπου q : πρώτος. Ο κώδικάς σας θα πρέπει να γεννάει τις παραμέτρους της ομάδας και της υποομάδας, να βρίσκει κάποιον κατάλληλο γεννήτορα εκείνη τη στιγμή, και να υπογράφει με τυχαίο $t \in_R \mathbb{Z}_q^*$ (όπως περιγράφει το πρωτόκολλο) το SHA-512 hash του (μεγάλου) αρχείου https://jasoncs.eu.org/large_file.bin. Παραθέστε τις παραμέτρους του συστήματός σας όπως δημιουργήθηκαν από τη διαδικασία και έπειτα σταθεροποιήστε τις στο πρόγραμμά σας (χωρίς όμως να σβήσετε τον τυχαιοποιημένο κώδικα δημιουργίας παραμέτρων). Επίσης, κατασκευάστε συνάρτηση που θα επαληθεύει την υπογραφή Schnorr (το πρόγραμμά σας δηλαδή θα πρέπει να υπογράφει και να επαληθεύει την υπογραφή του αρχείου). Η υπογραφή του προγράμματός σας θα επαληθευτεί και με εξωτερικό τρόπο, για αυτό παραθέστε με ευανάγνωστο τρόπο τις παραμέτρους του συστήματός σας. Κάθε πρώτος αριθμός πρέπει να είναι μεγέθους **τουλάχιστον 512 bits**. Εργασθείτε με το Fiat-Shamir challenge $c = H(y||m)$ για το σχήμα υπογραφών Schnorr.

Άσκηση 6.

Δίνεται το παρακάτω πρωτόκολλο μεταξύ ενός prover \mathcal{P} και ενός verifier \mathcal{V} το οποίο έχει στόχο την απόδειξη γνώσης του μηνύματος που αντιστοιχεί σε ένα δεδομένο κρυπτοκείμενο RSA με δημόσιο κλειδί (e, n) , δηλαδή $m \in \mathbb{Z}_n^*$ τέτοιο ώστε $y = m^e \bmod n$. Επιπλέον θεωρήστε ότι e πρώτος.

- Ο \mathcal{P} επιλέγει τυχαία ένα $t \in \mathbb{Z}_n^*$ και στέλνει στον \mathcal{V} το $h = t^e \bmod n$.
- Ο \mathcal{V} επιλέγει ένα τυχαίο $c, c \in \{0, \dots, e - 1\}$, και το στέλνει στον \mathcal{P} .
- Ο \mathcal{P} υπολογίζει το $r = tm^c \bmod n$ και το στέλνει στον \mathcal{V} .
- Ο \mathcal{V} αποδέχεται αν και μόνο αν $r^e \equiv hy^c \pmod{n}$.

Να αποδείξετε ότι το παραπάνω είναι Σ -πρωτόκολλο. Για την ιδιότητα HVZK η απόδειξη πρέπει να είναι στο επίπεδο ανάλυσης που ακολουθήθηκε στις διαφάνειες, αλλά να φαίνονται αναλυτικά τα transcripts του πρωτοκόλλου και η πιθανότητα εμφάνισής τους.

Άσκηση 7. Έστω το παρακάτω πρωτόκολλο μηδενικής γνώσης. Οι δημόσιες παράμετροι είναι $\langle p, m, g, h \rangle$ και ο prover γνωρίζει ένα x τέτοιο ώστε $g^x = h \pmod p$.

- Ο prover επιλέγει τυχαία ένα $t \in \mathbb{Z}_m^*$ και στέλνει στον verifier το $y = g^t \pmod p$.
- Ο verifier επιλέγει τυχαία $c \in \mathbb{Z}_m^*$ και το στέλνει στον prover.
- Ο prover υπολογίζει το $s = t + c + x$ και το στέλνει στον verifier.
- Ο verifier αποδέχεται αν και μόνο αν $g^s = yg^c h \pmod p$.

Εξετάστε αν το παραπάνω πρωτόκολλο είναι μηδενικής γνώσης για τίμιους επαληθευτές.

Άσκηση 8. Μία συνάρτηση σύνοψης $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ είναι ασφαλής για χρήση σε συστήματα proof of work (PoW) αν για κάθε είσοδο x είναι δύσκολο να βρεθεί λύση r ώστε να ισχύει $\mathcal{H}(x||r) \in Y$, όπου Y κάποιο σημαντικά μικρό υποσύνολο του $\{0, 1\}^n$.

1. Να αποδείξετε ότι μία συνάρτηση σύνοψης που έχει την ιδιότητα collision resistance δεν είναι απαραίτητα ασφαλής για PoW.

Υπόδειξη: Να κατασκευάσετε ένα αντιπαράδειγμα, δηλαδή μια συνάρτηση \mathcal{H}' που είναι collision resistant, αλλά όχι ασφαλής για PoW, επεκτείνοντας μια συνάρτηση \mathcal{H} που είναι collision resistant και ασφαλής για PoW.

2. Να δείξετε ότι η συνάρτηση $\mathcal{G}(z) = \mathcal{H}(z)||LSB(z)$, όπου $LSB(z)$ είναι το λιγότερο σημαντικό bit του z , είναι ασφαλής για PoW αλλά δεν έχει αντιστάση πρώτου ορίσματος.

Άσκηση 9.

1. Περιγράψτε ένα σενάριο στο οποίο δύο miners στο bitcoin δίκτυο ενώ ακολουθούν το πρωτόκολλο πιστά δημιουργούν δύο διαφορετικές αλυσίδες τις οποίες και ακολουθούν.
2. Να επιχειρηματολογήσετε ότι το παραπάνω σενάριο που περιγράψατε συμβαίνει με μικρή πιθανότητα.
3. Η Μίνα, μια κακόβουλη miner, σε κάθε block που βλέπει αλλάζει το coinbase transaction ώστε να πληρώνεται η ίδια πριν το κάνει relay στο δίκτυο. Γιατί η Μίνα δεν βγάζει επιπλέον κέρδη;

Άσκηση 10. (άσκηση αυξημένης βαρύτητας: 35% της σειράς)

Σε αυτή την άσκηση, θα γνωρίσουμε καλύτερα ορισμένες τεχνικές πτυχές του Bitcoin, και θα δοκιμάσουμε κάτι σαν "mining". Συγκεκριμένα, μπορείτε να διαβάσετε περισσότερα για τα transactions του Bitcoin [εδώ](#) και σχετικά με την ακριβή κωδικοποίηση και δημιουργία των διευθύνσεων [εδώ](#), καθώς και σε άλλες πηγές όπου μπορείτε να δείτε πώς ακριβώς προκύπτουν από private/public keys με την βοήθεια ενός elliptic curve.

(α) Εμείς εδώ θα πειραματιστούμε με την αλυσίδα "δοκιμών" του Bitcoin, δηλαδή το Bitcoin testnet και όχι το Bitcoin mainnet (προφανώς). Σε πρώτη φάση, κατασκευάστε ένα private key και την αντίστοιχη Bitcoin testnet διεύθυνσή του (διεύθυνση 1). Μπορείτε για αυτό να χρησιμοποιήσετε θεμελιώδεις συναρτήσεις, όπως RIPEMD-160, SHA-256 hash functions, και ECDSA signatures, αλλά **μην** χρησιμοποιήσετε έτοιμες βιβλιοθήκες για Bitcoin ή συναφή που να παρέχουν δημιουργία διευθύνσεων, έστω και τμηματικά. Ο σκοπός είναι να κάνετε τη δική σας διαδικασία δημιουργίας private key και (δημόσιας) διεύθυνσης, βάσει των προαναφερθέντων primitives.

Ψάξτε σε μηχανή αναζήτησης για "bitcoin testnet faucet" για να λάβετε δωρεάν κάποια νομίσματα (τα ονομάζουμε tBTC από το test Bitcoin) στην παραπάνω διεύθυνσή σας, για να μπορείτε να εκτελέσετε τις επόμενες λειτουργίες και να πειραματιστείτε.

Έπειτα, δημιουργήστε μια διεύθυνση (διεύθυνση 2) με την εξής ιδιότητα: αντί για το κλασσικό RIPEMD160 (SHA256(...)) θα έχετε το RIPEMD160(SHA256(Αριθμός_Μητρώου_σας)). **Παράδειγμα** (για να ελέγξετε τον κώδικά σας) αποτελεί η διεύθυνση

mkaWYS2DeChGv3u5tZMR59WSQkJEk61E3k

που θα ήταν η ζητούμενη για τον AM 03112345. (το μηδενικό συμπεριλαμβάνεται στην κωδικοποίηση!) Είναι εύκολο να λάβετε τα χρήματα από την παραπάνω (διεύθυνση 2) που δημιουργήσατε εσείς βάσει του AM σας, ή όχι; Αν ναι, περιγράψτε ακριβώς με ποια βήματα μπορείτε να ανακτήσετε τα χρήματα από εκείνη τη διεύθυνση. Αν όχι, γιατί;

Στη συνέχεια, κατασκευάστε (με όποιον τρόπο επιθυμείτε, όχι κατ' ανάγκη με κώδικα) ένα transaction με input(s) από την διεύθυνσή σας (διεύθυνση 1), με το οποίο θα αποστέλλετε στη διεύθυνση

n3Uk2aQLXogYEbzYJnKyk9JSCeUAKVyB7q

το ποσό των 0.01 tBTC, καθώς επίσης και στην (διεύθυνση 2) σας το ποσό των 0.01 tBTC, και εάν σας έχει περισσέψει κάτι σε ποσό πίσω στη (διεύθυνση 1) σας. Δηλαδή το transaction αυτό θα έχει 2 ή 3 outputs. Μην ξεχάσετε να αφήσετε ένα (πολύ μικρό, πχ 0.00001 tBTC) ποσό και για transaction fee, ειδικά το transaction σας μπορεί να μην γίνει confirm ποτέ! Ένα παράδειγμα τέτοιου transaction που είναι ήδη επάνω στο δίκτυο είναι αυτό με transaction id:

0071ed6b67e53750b6fb54a536a5f7fee0764485be1f0044c388ac754b7de2df

Συμπεριλάβετε τα βήματα που ακολουθήσατε, τον κώδικα που γράψατε, καθώς και όλες τις σχετικές διευθύνσεις (1, 2) και το παραπάνω transaction id, όπως εμφανίζονται επάνω στο δίκτυο και αποδεικνύουν την εκτέλεση των ανωτέρω (προσοχή, θα εκτελεστεί αυτόματος έλεγχος στα παραπάνω οπότε σιγουρευτείτε ότι έχετε καταλάβει καλά τα παραδείγματα που σας υποδεικνύουν πώς θα πρέπει να γίνουν).

(β) Τώρα θα κάνουμε κάτι σαν "mining" επάνω στο transaction που παραγάγατε. Συγκεκριμένα, σας ζητείται να βρείτε τέτοιο nonce (οποιοδήποτε μεγέθους σε πλήθος bits εσείς κρίνετε κατάλληλο) σε δεκαεξαδική μορφή, ώστε το concatenation του nonce και του transaction σας να έχουν ένα νέο, αρκούτσως μικρό "transaction id". Προφανώς, τα παραπάνω nonce/transaction αναφέρονται σε hex μορφή, που υποδηλώνει το περιεχόμενο των αντίστοιχων bytes, και **όχι** ASCII/UTF-8 κωδικοποίηση, πχ το SHA-256 hash του hex "transaction" 0x0100deadbeef θα ήταν αυτό:

e2b6d72e359802bd6ffeb28aaeaf67947be21bb06961defbc8d84638e9628740


και **όχι** αυτό:


9f659f3c812af3c4e18cee90896d31d8245d5b5c7e52c79923425fbfb024ce24

Μπορείτε να βρείτε την hex μορφή του transaction σας από κάποιον Bitcoin testnet explorer, αν δεν το έχετε ήδη. Γράψτε κώδικα που να προσδιορίζει κατάλληλο nonce με σκοπό τα πρώτα **32** bits του νέου "transaction id" να είναι μηδενικά. Προφανώς, θα πρέπει το $\text{SHA256}(\text{SHA256}(\text{tx_hex}))$ να δίνει το αρχικό σας transaction id που βλέπετε επάνω στο δίκτυο, ενώ το $\text{SHA256}(\text{SHA256}(\text{nonce} \parallel \text{tx_hex}))$ θα είναι το νέο "transaction id". **Λάβετε υπόψιν** σας ότι τα transactions είναι εσωτερικά κωδικοποιημένα με little-endian μορφή, ενώ το transaction id φαίνεται στον χρήστη ως big-endian, με λίγα λόγια θα χρειαστεί να αντιστρέψετε το "byte order" της εξόδου από little σε big έπειτα από την διπλή SHA256. Σιγουρευτείτε ότι το ακόλουθο παράδειγμα σας δουλεύει σωστά: το transaction με id

0071ed6b67e53750b6fb54a536a5f7fee0764485be1f0044c388ac754b7de2df

έχει hex content εδώ: https://jasoncs.eu.org/hex_tx.txt οπότε θα πρέπει να σας βγαίνει το εν λόγω transaction id με "κενό" nonce.

 (γ) Εκτελέστε το παραπάνω ερώτημα (β), όπου όμως τα πρώτα **42** bits θα πρέπει να είναι μηδενικά. Ενδέχεται να χρειαστεί να μεταβιβάσετε κάποιο τμήμα του κώδικά σας σε GPU, για να καταφέρετε να το υπολογίσετε χωρίς να περάσει αρκετός καιρός...

 : bonus ερώτημα.

Σύντομες οδηγίες: (α) προσπαθήστε μόνοι σας, (β) συζητήστε με συμφοιτητές σας, (γ) αναζητήστε ιδέες στο διαδίκτυο, με αυτή τη σειρά και αφού αφιερώσετε αρκετό χρόνο σε κάθε στάδιο! Σε κάθε περίπτωση οι απαντήσεις πρέπει να είναι *αυστηρά ατομικές*.