מבוא למטבעות קריפטוגרפים – תרגיל בית 2

תאריך הגשה: יום שבת, 21 דצמבר, 23:59

מטרת התרגיל:

בתרגיל בית 1 בניתם מערכת תשלומים מאובטחת עם ישות מרכזית אחת - ה"בנק". למרות שהמערכת הייתה מבוססת על פעולות קריפטוגרפיות, הבנק עדיין יכל למנוע מטרנזקציות להיכנס לבלוקים, והוא לא וידא לחלוטין את הבלוקים שהוא יצר. בנוסף, הבנק יכל להדפיס כסף ככל העולה על רוחו.
בתרגיל זה, נחליף את הבנק בקודקודים שמריצים בלוקצ'יין עם פרוטוקול הרשת הארוכה ביותר. כמו קודם, אנחנו נניח לטרנזקציה יש input ו-output יחידים, והוא יכול להחזיק מטבע אחד בלבד.

סקירה כללית:

ה-API של הישות Node ("קודקוד") בתרגיל הוא שילוב של ה-APIs של הבנק ושל הארנק מתרגיל 1, עם הרחבה. הקודקודים יכולים לכרות בלוקים חדשים, ובנוסף גם לשלוח ולקבל כספים.
כל קודקוד מנהל mempool משלו, והוא מחזיק העתק של הבלוקצ'יין ושל אוסף ה-UTxO. לקודקוד יש גם כתובת (מפתח ציבורי) לקבלת כספים.
גם בתרגיל זה, נספק את ה-APIs שתצטרכו לממש. עבור אובייקטים מסוימים כמו בלוק או טרנזקציה תוכלו למשל להיעזר במימוש שלכם מתרגיל בית 1. במקרים אחרים, תצטרכו לבצע התאמות.
בדומה לתרגיל 1, נשתמש בספריות cryptography (בשביל חתימות), hashlib (בשביל SHA25), וב-secrets module (על מנת לייצר רצף רנדומלי של בייטים שנכלול בטרנזקציות של יצירת הכספים).
כעת, נסביר על התהליכים שהתווספו בתרגיל זה ל-APIs או שהשתנו מתרגיל בית 1.

תקשורת בין קודקודים: קודקודים יכול להתחבר לכל אחד (בעזרת קריאת API). בתהליך ההתחברות, הקודקודים יודיעו אחד לשני על ראש השרשרת שלהם. טרנזקציות שהיו קיימות ב-mempool עד כה לא ישותפו בין הקודקודים בתהליך ההתחברות.
קודקודים מחוברים יכולים להודיע אחד לשני על בלוקים חדשים שהם כרו, כמו גם על בלוקים שהם קיבלו (אחרי שקודקוד מסוים למד על ראש השרשרת מקודקוד אחר, כל הקודקודים השכנים שלו גם ילמדו על ראש השרשרת הזה – כלומר הקודקוד יצטרך לפעפע את מה שהוא למד).
הודעות כאלו יכולות להתרחש רק אם הבלוק תקין והוא אכן חלק מהשרשרת הארוכה ביותר.
בנוסף, קודקודים מודיעים לשכנים שלהם על טרנזקציות חדשות שהם למדו עליהם שנכנסו ל-mempool.
שימו לב, אם טרנזקציה מסוימת עלולה ליצור מצב של double-spend עם טרנזקציה אחרת שכבר קיימת ב-mempool אז היא לא תפועפע אל השכנים).

אימות בלוקים וסידור השרשרת (Chain Re-Orgs): קודקודים מודיעים אחד לשני על hash של בלוק שהם כרו או גילו. אם אותו hash הוא חלק מבלוק שאינו מוכר להם (לקודקוד שיידעו אותו על הבלוק החדש), אז אותו קודקוד יבקש את כל הבלוק. אם הבלוק שקיבלנו משורשר לבלוק קודם שהוא אינו מוכר, כלומר הבלוק הקודם בשרשרת, אז הקודקוד יבקש גם את הבלוק הזה. התהליך הזה חוזר חלילה עד שהקודקוד מגיע לבלוק שהוא מכיר (או ל-Genesis block).
לעיתים, בזמן תהליך זה, קודקודים עלולים לגלות שקודקוד אחר מחזיק שרשרת ארוכה יותר ממה שיש להם. במקרה זה, על הקודקוד לעדכן את העתק הבלוקציין שלו לשרשרת החדשה יותר, ולעדכן את כל מבני הנתונים הרלוונטיים (כמו למשל אוסף ה-UTxO ואוסף ה-mempool).
חשוב – קודקודים אחרים ברשת יכולים להיות רעים והוא ינסו לשנות בלוקים או טרנזקציות. לכן, חשוב מאוד לוודא שהבלוקים והטרנזקציות שמקבלים אכן תקינים.
בלוקים פגומים יכולים להיות גדולים יותר (עוברים את מגבלת הזיכרון של הבלוק, שהוא מוגדר כקבוע בתרגיל), להכיל טרנזקציות פגומות בתוכם, double spends, או סכום לא תקין של טרנזקצית יצירת הכסף (יותר או פחות ממטבע 1). **תהיו זהירים במיוחד כאשר אתם מקבלים בלוק פגום או לא תקף לשרשרת**.

כרייה: במקום שהקודקודים יפעלו לפי ה-proof-of-work כדי להחליט מתי הם כורים בלוק, נפעיל פונקציה חיצונית (למשל מתוך הסקריפט של הטסט) שבה נודיע לבלוק מסוים שזה התור שלו לכרות בלוק. ב-PoW רגיל, כל הכורים מנסים כל הזמן לייצר בלוקים, והכורה שמצא את hash מתחת לערך ה-target זוכה. אנחנו נעדיף שלא לגרום למעבדים שלכם לעבוד שעות נוספות, ולכן רק נעמיד פנים שהוא מריצים את התהליך הזה). לכן, אין צורך באמת לבלוק אם ה-hash של הבלוק הוא מתחת ל-target מסוים.

יצירת כספים: כל בלוק יכיל טרנזקצית יצירת כספים אחת בלבד. הטרנזקציה מקצה את הכסף אל הכורה. נזכיר שטרנזקציות של יצירת כספים אין להם input, ובמקום חתימה יש להם רצף רנדומלי של 64 בייטים.

שליחת כספים: מכיוון שקודקודים מתפקדים גם כארנקים וגם ככורים, כאשר קודקוד יוצר טרנזקציה הוא ישירות מכניס אותה אל ה-mempool שלו ומפעפע אותה אל השכנים שלו. קודקודים לא ינסו לבצע double-spend לטרנזקציה שהוא יצרו, לכן, לא ניצור טרנזקציות חדשות שמבזבזות outputs של טרנזקציות שכבר נמצאות ב-mempool. עם זאת, כאשר ננקה את ה-mempool של קודקוד אז ניתן לו את האפשרות לנסות לבזבז שוב את ה-outputs.
בנוסף לכך שקודקוד שם טרנזקציה חדשה ב-mempool שלו, ושהוא מודיע לשכנים שלו על טרנזקציה חדשה, השכנים גם כן יודיעו לשכנים שלהם על הטרנזקציה החדשה שהגיעה אליהם.
כלומר, בתורם הם יודיעו לשכנים שלהם וחוזר חלילה. שימו לב שתהליך זה קורה רק אם טרנזקציות אכן נכנסו ל-mempool של הקודקוד.

Tests:

בדומה לתרגיל 1, סיפקנו לכם טסטים בסיסיים של ה-API. הם יעזרו לכם להבין את ההתנהגות הרצויה של המערכת. הטסטים מכסים חלק מהאפשרויות, מומלץ לכתוב עוד טסטים כדי להרחיב את האפשרויות של המערכת.

מה לא נבדוק: יש כל מיני ניואנסים של איך הקודקוד צריך להתנהג. לדוגמה, לקודקודים שמשנים את השרשרת הארוכה ביותר שלהם עלול להיות טרנזקציות בבלוקים שהם קיימים בבלוקים של השרשרת החדשה. אנחנו לא נבדוק את ה-mempool אם אכן הכנסתם טרנזקציות כאלו.
בנוסף, אנחנו לא נבדוק את ההתנהגות של ה-mempool במקרה וקודקוד מנסה לשנות לשרשרת ארוכה ביותר חדשה שבסוף נתפסה כתקולה.
לבסוף, אנחנו לא נבדוק ניסיונות לגשת או לשנות שדות פנימיים של האובייקט Node או Block. הטסטים יכילו רק התנהגות נורמלית מצופה או "מתקפות" שתוקפים חיצוניים מנסים לבצע: לשלוח הודעות שנחתמו בצורה שגויה, בלוקים לא תקינים, ועוד.

הגשה:

יש להגיש את הקבצים בתוך zip שנקרא ex2.zip. קובץ ה-zip יכיל את קבצי ה-python של תת התיקייה ex2 **(אל תוסיפו את קבצי ה-tests)**. בנוסף, עליכם להוסיף קובץ README (לא README.txt). קובץ ה-README צריך להכיל את השמות שלכם, תעודות זהות והסבר קצר על התרגיל.
על מנת להריץ את הפתרון שלכם, אני אייבא אותו מ-ex2 (כמו שה-tests בתיקיית ה-tests עושים).
שימו לב שאתם מגדירים רק classes רלוונטיים (כמו בקבצי הטמפלייט) ושלא רץ קוד נוסף.

שימו לב שרק שותף אחד צריך להגיש את התרגיל!

בהצלחה!