תמיר אלביליה

כיתה: יב 6

מדריך: גל בראון

ת"ז: 325832681

DREAMVEIL

A general purpose blockchain and cryptocurrency.
12th grade cyber project.

תוכן עניינים

פתיחה

רקע:

באינטרנט הנוכחי קיימים מספר רב של שירותים שונים ומגוונים, אך למרות זאת, במשותף לכל השירותים הללו הוא שיש לסמוך על שרת מרכזי על מנת לעשות בהם שימוש. אך מדוע לנו לסמוך על שרת מעבר למסך שאכן ינהג בהגינות כלפי כל משתמשיו? באמצעות הקריפטוגרפיה ניתן לעקוף את מתן הסמכות והצורך בשרת מרכזי למתן שירות ובכך למגר את עריצות השרת המרכזי.

הרעיון:

מטרת הפרויקט היא יצרת בלוקצ'יין – מסד נתונים המאוחסן ללא צורך בשרת מרכזי (Decentralized) כך שלא יהיה כוח רב לאף שרת יחיד שבו כל המחשבים המשתתפים במערכת מחוברים ברשת P2P מפוזרת. המערכת תוכל לתפקד כראוי גם כאשר חלק מן השרתים מנסים לתקוף את המערכת. בפרויקט זה מסד הנתונים יתמוך בהעברה של כסף דיגיטלי (Cryptocurrency) והפצה של הודעות ברבים.

שימוש:

המערכת שימושית עבור כל מי שמעוניין להעביר כסף דיגיטלי ללא הרצון לסמוך על שרת מרכזי כזה או אחר או לשדר מידע ברבים ללא צנזור.

הגדרת הבעיה ופתרונה

כעת באינטרנט שירותים כמו העברת כספים והפצת הודעות מחייבות מתן סמכות בשרת מרכזי. כדי שנוכל להסיר את מתן הסמכות מהמערכת ראשית יש לעבור לרשת עמית לעמית שבה כל מחשב במערכת מסוגל להתחבר לכל מחשב אחר במערכת ושאינה כוללת שרת מרכזי. כל המחשבים המחוברים במערכת יקליטו את רצף האירועים והעסקאות שקרו במערכת. אך מכך נובעת בעיה חדשה, כיצד ניתן להגיע לקונצנזוס ולהחליט מהו רצף האירועים האמיתי שקרה. על מנת לפתור בעיה זו על מסד הנתונים להיות חסין מפני שינוי ספונטני או זיוף וזהום של כל אחד מהעמיתים במערכת. נעשה זאת באמצעות הגדרת שרשרת בלוקים (בלוקצ'יין). שרשרת הבלוקים תהיה מורכבת מאוסף בלוקים שבו כל בלוק מקושר לבלוק הקודם לו באמצעות hash. כל בלוק כולל בתוכו את הhash שלו עצמו ואת הhash של הבלוק הקודם לו. מכיוון שהhash הוא בהגדרה ייחודי עבור כל בלוק, מתקבלת התופעה שעל מנת לשנות בלוק יחיד בשרשרת הבלוקים, יש לשנות את כל הבלוקים אחריו. בכדי להפוך תופעה זו לכלי הגנה מפני שינויים בשרשרת נשתמש בproof-of-work (הוכחת עבודה) ונגדיר שבשביל שhash של בלוק יהיה תקין, על הhash להיחשב כתקין, עליו לקיים $nMSb(hash) = \{0\}^n$; יש לשים לב שמשום שהפלטים פונקציית הhash היא פסודו-אקראית, ההסתברות לקבל hash שעובר proof-of-work בקושי n היא $\left(\frac{1}{2}\right)^n$. לכן כאשר מספר האפסים הנדרשים בתחילת כל hash גדול, שינוי של כל בלוק בשרשרת נעשה בלתי אפשרי עבור כל תוקף שאין לו לפחות כסף מחצי מכוח המחשוב במערכת, זאת משום שתוקף שכזה יצטרך לכרות מחדש את כל הבלוקים שאחרי הבלוק שברצונו לשנות ובו זמנית לעקוף את שאר חברי המערכת שעובדים תמידית להרחיב אותה. כאשר אין לתוקף שכזה מספיק כוח מחשוב, הוא לא יוכל לעקוף את עבודתם של שאר מחשבי הרשת. עקב כך ניתן לקבוע את השרשרת הנכונה כשרשרת המאסיבית ביותר אשר להכינה היה נדרש את כוח המחשוב הרב ביותר. מתקבל קונצנזוס משום שכל שאר המשתתפים במערכת ישתפו אחד עם השני על העדכונים ששמעו ממשתפים אחרים וישתמשו בשרשרת המאסיבית ביותר כשאר כל המשתתפים. כל אחד מהבלוקים בשרשרת מכילים העברות (transactions). כל העברה מסמלת פעולה שבה משתמש מעביר מהכסף האלקטרוני הרשום על חשבונו למספר חשבונות אחרים. על מנת לוודא שמחבר ההעברה הוא אכן בעל החשבון נשתמש בפתרון הכולל חתימות דיגיטליות ובכדי למנוע בזבוז כפול של כספים שכבר בוזבזו, נשתמש במבנה נתונים שמאפשר אכסון וארגון של כל ההעברות ההכשרות שתוצאותיהן עוד לא בוזבזו.

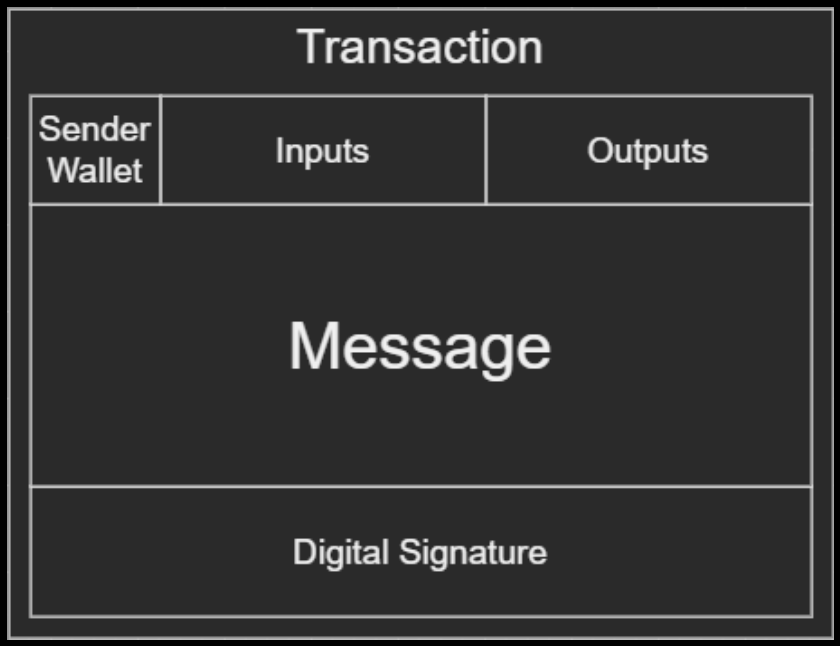
מבנה הבלוקצ'יין

העברה – Transaction

בבלוקצ'יין העברה היא המידע הבסיסי שמסד הנתונים מכיל, כל העברה היא הצהרה על מידע שיש לשמר בבלוקצ'יין. העברה מכילה הכרזה על העברת מטבעות דיגיטליים מארנק אחד לארנק אחר. ההעברות ישודרו בפומבי אל משתמשי הרשת על מנת שיועברו על להעברות מספר תנאים שיש לעמוד בהם על מנת לממש רכיב זה בבלוקצ'יין.

1. כל העברה חייבת להיות מיוצרת על ידי המקור המצוין בה בהכרח.
2. יש למנוע מכלל הלקוחות לשנות את התוכן של העברה לאחר שפורסמה ברשת
3. נדרשת היכולת להבדיל בין העברה אחת להעברה אחרת ללא תלות בתוכן ההעברה.

הינה תרשים של Transaction המתארת את מאפייני ההעברה:



- Sender wallet – הארנק של שולח ויוצר העברה והארנק של הגוף בהעברה הארנק יהווה מפתח ציבורי במערכת הצפנה א-סימטרית RSA.
- Inputs – מילון המכיל את כל מקורות ההעברה וערכם.
- Outputs – מילון המכיל את כל ארנקי מקבלי ההעברה וחלקם ממנה.
- Message – ההודעה שמכילה ההעברה, אורך מקסימלי של 222 תווים.
- Nonce – טקסט אקראי שמטרתו להשפיע על החתימה הדיגיטלית של ההעברה. מכיוון שהNonce מיוצר אקראית עבור כל העברה, החתימה הדיגיטלית שלה

תהיה ייחודית רק לה. הNonce מונע מאותן העברות לקבל שיכפול ושימוש חוזר לא כדין.

- Sender’s Digital Signature – החתימה הדיגיטלית של מחבר ההעברה, החתימה כוללת את כל העברה ומושפעת מערך הNonce. מטרתה להבטיח שימור של הערכים בהעברה ולהבטיח ששולח ההעברה הוא גם המחבר שלה.

עקרונות, אתגרים במימוש ופתרונות

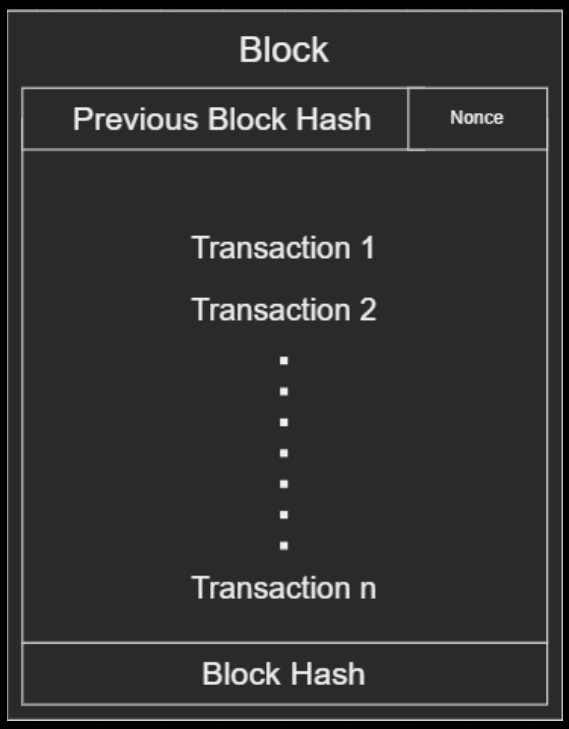
- כיצד ניתן למנוע מגוף זר לבצע העברה עבור לקוח אחר? על מנת לפתור בעיה זו נפנה למושג בקריפטוגרפיה הנקרא חתימה דיגיטלית.
כל לקוח ייצר צמד מפתחות אחד פרטי ואחד ציבורי, כאשר לקוח מעוניין לחבר העברה, הוא יצמיד אליה את המפתח הציבורי שלו (הוא גלוי לכולם). בסוף ההעברה ייצר המחבר חתימה דיגיטלית המושפעת מגוף ההעברה ויצרף אותו אל סוף ההעברה. בסופו של דבר תתקבל מערכת שבה ניתן לוודא שהעברה מסוימת אכן חוברה על ידי השולח. כל לקוח כעת יוכל לאמת זאת באמצעות המפתח הציבורי המצורף בהודעה, והחתימה הדיגיטלית בסוף ההודעה.
- החתימה הדיגיטלית גם פותרת בעיה נוספת, היא מונעת מכול זר לשנות את התוכן הקיים בה, זאת משום שכל שינוי של התוכן אמור לשנות את ערך החתימה הדיגיטלית ולא ניתן לשנות את ערך החתימה משום שהמפתח הפרטי נדרש על מנת לייצר אותה, אך אין לאף זר מלבד השולח את אותו מפתח הפרטי. כל שינוי של התוכן יבטל את ההתאמה של המפתח הציבורי שצירף השולח לחתימה הדיגיטלית של ההעברה.
- כיצד נוכל לדעת אם העברה כלשהי היא חוקית ויש למחבר מספיק יתרה לבצע אותה? בכדי שההעברה תיחשב חוקית עליה לקיים את חוק שימור הכסף בinputs וoutputs שלה, כלומר sum(inputs) – sum(outputs) = 0.
- כיצד נדע איך כל אחד מהמקורות אכן על השרשרת ואינו בוזבז? שרשרת הבלוקים היא גדולה מאוד ולעבור על כולה עבור כל מקור זה אינו פתרון ממשי. על מנת לאכסן את מצב ההעברות שאינן בוזבזו נשתמש במבנה נתונים יעיל AVL שבו נאכסן את מצב כל הoutputs של כל העברה שהוכנסה לשרשרת באופן ממוין לפי החתימה הדיגיטלית של ההעברה אליהם הם משתייכים. בעץ AVL הסיבוכיות להוסיף או למצוא איבר במבנה הנתונים היא log(n) כאשר n זה מספר ההעברות בשרשרת. עקב כך נגדיר שמקור הוא חוקי רק כאשר מופיע בעץ ומחבר ההעברה הוא בעל המקור.
- להעברה מותר לציין את המקור המיוחד "BLOCK" המסמל שהמקור הוא הפרס המתקבל מהבלוק עליה היא נמצאת.
- להעברה מותר לציין את הפלט המיוחד "MINER" המסמלת שיש להוסיף את המטבעות הרשומים בפלט זה לפרס הבלוק שבו נמצאת ההעברה. בעצם פלט זה

משמש כמס לכורים על מנת לשכנע אותם לכלול את ההעברה הזו בבלוק שלהם לפני העברות אחרות.

בלוק – (Block)

בלוק מכיל אוסף של העברות ומטרתו לשמר את סדר האירועים שבו העברות התקבלו. כל בלוק מכיל hash שלו עצמו וhash של הבלוק הקודם לו. על מנת שבלוק יהיה תקין אוסף ההעברות שלו חייב להיות תקין. על מנת שבלוק יתקבל על ידי משתתפי הרשת על hash הבלוק לעבור proof-of-work.

הינה תרשים של Block המתאר את מאפייני הבלוק:



- Previous Block Hash – hashה של הבלוק הקודם
- Nonce – ערך טקסט אקראי שמטרתו לשנות ולהשפיע על hash הבלוק, נקרא גם פתרון הבלוק
- transactions – רשימה של כל ההעברות שהבלוק מכיל.
- Block Hash – hashה של הבלוק הנוכחי.

מאפיינים ועקרונות

- מכיוון שהhash של הבלוק מכיל את הhash של הבלוק הקודם לו כאשר משנים בלוק במקום הn, פעולה זו מבטלת את כל הבלוקים אחריו משום שה Previous

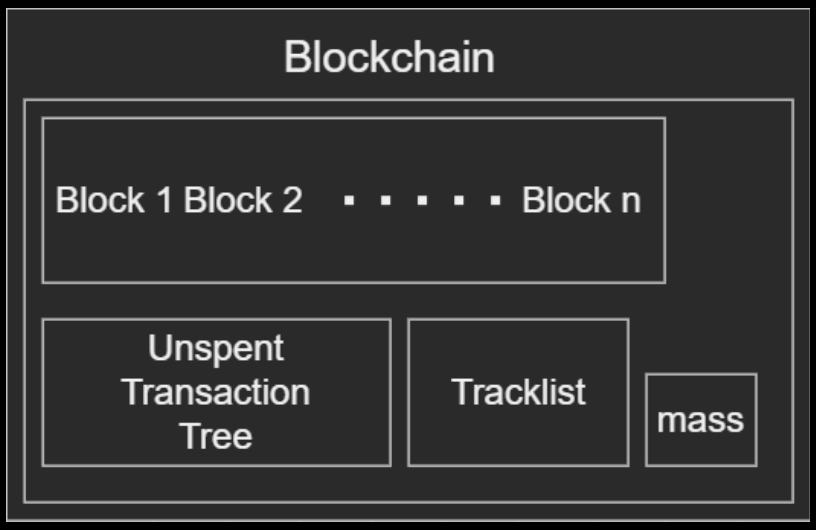
Block Hash שלהם אמור להשתנות בהתאמה ולכן גם hash שלהם. כאשר על הhash.

- כאשר מייצרים בלוק אין זה סביר שהhash שיתקבל יתאים לproof-of-work הרצוי על ידי הרשת. לכן על מנת לשנות ולהשפיע על הhash, נשנה את ערך זה עד אשר יתקבל hash מתאים. תהליך זה נקרא כרייה והוא לוקח עוצמת מחשוב וזמן רבים ביחס לקושי הproof-of-work הרצוי.
- על מנת לעודד כורים להשקיע זמן וכוח מחשוב בכריית בלוקים, כאשר בלוק נכרה, לכורה הבלוק היכולת להוסיף העברה מיוחדת שמקורה אינה העברה קיימת ובכך מקבל הכורה פרס על עבודתו. גודל הפרס נקבע לפי מקום הבלוק בשרשרת. הגודל התחילי של הפרס שווה ל727 וכל 52560 בלוקים נחתך בחצי. מכיוון שמקור כל המטבעות בשרשרת מכריית בלוקים, ניתן לחשב את מספר המטבעות הכולל $2 * 727 * 52560 = 76422240$

שרשרת בלוקים (בלוקצ'יין) – Blockchain

שרשרת הבלוקים היא מבנה הנתונים העליון והיא מכילה אוסף של בלוקים המסודרים לפי הhashes שלהם כך שעבור n block הPrevious Block Hash שלו שווה לHash של הבלוק הקודם לו בשרשרת והPrevious block hash של הבלוק העוקב לו שווה לBlock Hash שלו. לשרשרת הבלוקים גם עץ AVL שבו היא שומרת על איזה העברות כבר בוזבזו.

הינה תרשים של Blockchain המתארת את מאפייני שרשרת הבלוקים:



- chain – רשימה של הבלוקים שהשרשרת מכילה
- Unspent Transaction Tree – עץ AVL העוקב אחר אלו מקורות חוקיים עוד לא בוזבזו.
- mass – סכום הproof-of-work של כל הבלוקים בשרשרת.

- Tracklist – מילון שמפתחותיו הם כל כתובות הארנקים שיש לעקוב אחר פעולותיהם (ההעברות הקשורות אליהם) וערכיו הם הבלוק שבה בוצעה הפעולה והחתימה הדיגיטלית של ההעברה הרלוונטית.

עקרונות

- בלוקים ששורשרו לשרשרת אינם ניתנים לשינוי. לאחר שבלוק שורשר לשרשרת, הבלוק מקובע בה.
- הבלוק הראשון בשרשרת (מכונה גם בלוק האפס או בלוק בראשית " Genesis Block") הוא הבלוק היחידי שלערך הprevious_block_hash שלו מותר וחייב להיות סטרינג ריק.

chain_block

הפעולה מקבלת Block כפרמטר ומטרתה לנסות לשרשר את הבלוק הנתון אל שרשרת הבלוקים. במידה והשרשור צלח הפעולה מחזירה True אחרת הפעולה מחזירה False.

ראשית הפעולה בודקת שהבלוק עצמו תקין.

לאחר מכן הפעולה בודקת אם גודל השרשרת גדול מאפס, אם כן על ה Previous Block Hash של הבלוק להיות שווה לHash של הבלוק העליון הנוכחי בשרשרת. אם גודל השרשרת שווה לאפס על הPrevious Block Hash להיות שווה לסטרינג ריק.

לאחר מכן הפעולה קוראת Blockchain.verify_block שתפקידה לחפש כל אחת מן המקורות הנמצאים בבלוק ולוודאות שהמקורות אכן נמצאות בעץ unspent_transactions_tree, כלומר שהן עדיין לא בוזבזו. הפעולה גם מוודא שהפרס הנמצא בבלוק אכן זהה לפרס הבלוק המחושב לפי אורך השרשרת ולפי סכום המיסים בכל העברות הבלוק.

אם כל הצעדים הללו עברו בהצלחה הבלוק משורשר לשרשרת הבלוקים.

הפעולה כעת מוסיפה כל אחת מההעברות בבלוק לעץ ההעברות הלא מבוזבזות ביחד עם הפלטים שלה ומסירה את כל המקורות מהעץ (הם בוזבזו).

לבסוף עבור כל ארנק בtracklist הפעולה מוסיפה את כל המקורות או הפלטים בבלוק ששורשר הקשורים לארנק בפורמט (new_block_index, transaction.signature) לערך הארנק בtracklist.

הפעולה מחזירה True כאשר הבלוק שורשר בהצלחה.

```
def chain_block(self, block:Block):
"""Chains a block to the blockchain. This function succeeds only if a block is valid.
:returns: Did block chain (boolean)"""
```

```
canidate_block = Block.loads(block.dumps())
if canidate_block is None:
return False

if len(self.chain) > 0:
# Block does not continue our chain
if canidate_block.previous_block_hash != self.chain[-1].block_hash:
return False
else:
if canidate_block.previous_block_hash != "":
return False

if not self.verify_block(canidate_block, len(self.chain)):
return False

# Add the newly accepted block into the blockchain
new_block_index = len(self.chain)
self.chain.append(canidate_block)

# Update the mass of the chain
self.mass += Block.calculate_block_hash_difficulty(canidate_block.block_hash)

# For each now accepted transaction in the newly chained block
for transaction in self.chain[-1].transactions:
# Mark the new transaction as unspent
self.unspent_transactions_tree.insert(data_structures.binary_tree_node(transact
ion.signature, transaction.outputs.copy()))

# For each input the new transaction referenced
for heavenly_principle_struck_transaction in transaction.inputs: # All is lost to
time (and use) (?)
# Find the node that stores the status of the input-referenced transaction
intree_node =
self.unspent_transactions_tree.find(heavenly_principle_struck_transaction)

if intree_node is not None:
# We remove the transaction's output as it was spent
intree_node.value.pop(transaction.sender)
```

```
# Track the transaction for each of the tracklist addresss
for tracklist_address in self.tracklist.keys():
if tracklist_address in transaction.inputs or tracklist_address in
transaction.outputs:
self.tracklist[tracklist_address].append((new_block_index,
transaction.signature))
return True
```

```
def verify_block(self, block:Block, block_height:int):
"""
This function verifies that the sender of each transaction in the block has the
resources to carry it out.
Transactions do not recognize other transactions in the same block to prevent
order frauding
"""

assert block_height >= 0
block_fees = to_decimal(0)
miner_reward_transaction = None
# For each transaction in the block
for transaction in block.transactions:
# Go over all of the inputs referenced in the block.
for input_source, input_amount in transaction.inputs.items():
if input_source != "BLOCK":
transaction_node = self.unspent_transactions_tree.find(input_source)
if transaction_node is None:
print("Block rejected in verify_block (referenced transaction does not
exist)")
return False
if transaction.sender not in transaction_node.value.keys():
print("Block rejected in verify_block (output was already spent or is
invalid)")
return False
if transaction_node.value[transaction.sender] != input_amount:
print("Block rejected in verify_block (output amount is not the same as
specified in the transaction)")
return False
```

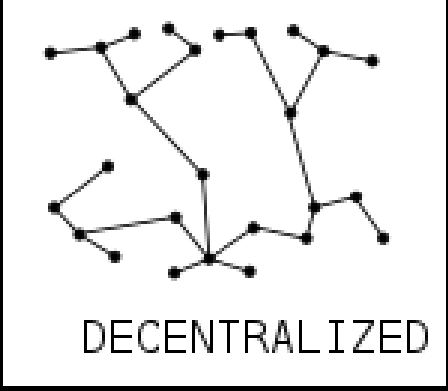
```
else:
miner_reward_transaction = transaction
block_fees += transaction.get_miner_fee()

proposed_block_reward = to_decimal(0)
for output in miner_reward_transaction.outputs.values():
proposed_block_reward += to_decimal(output)
if proposed_block_reward != Blockchain.calculate_block_reward(block_height) +
block_fees:
print("Block rejected in verify_block (Miner transaction does not evaluate to the
correct amount)")
return False
return True
@staticmethod
def calculate_block_reward(height):
"""
Calculate the reward of the block using a predefined geometric series

We divide block reward in two every 52560 blocks (half a year if 5m per block)
a0 = 727 * 52560 = 38211120
sum of geometric series = 2 * a0 = 76422240
Total currency amount 76422240

:param Height: Height of the block
:returns: decimal.Decimal block_reward
"""
q = 0.5
n = height // Blockchain.BLOCK_REWARD_SEASON
block_reward = Blockchain.BLOCK_INITIAL_REWARD * q**n
return to_decimal(block_reward)

def calculate_transaction_value(self, transaction, address):
"""
Returns the amount of funds a transaction has for a given wallet address.
:returns: None if the transaction does not exist or is spent;
decimal.Decimal Transaction value.
"""
transaction_node = self.unspent_transactions_tree.find(transaction.signature)
if transaction_node is not None:
```

```
if address in transaction_node.value:
return transaction_node.value[address]
return None
```

מבנה התקשורת

התקשורת בפרויקט נעשית באמצעות רשת עמית לעמית (P2P) שבה כל עמית (peer) יוצר חיבורים ומקבל חיבורים במקביל. מטרת הרשת היא להיות מפוזרת ככל שאפשר ולכן אל לנו להשתמש בשרת מרכזי ורצוי שהרשת תהיה מפוזרת ככל שאפשר.

להלן ציור מופשט של מודל של רשת מפוזרת:



שרת - Server

השרת מטרתו לנהל את כל החיבורים עם שאר העמיתים שאליהם עמית כלשהו מחובר. כל עמית פותח במחשבו שרת שביכולתו ליצור ולקבל חיבורים במקביל. השרת מתחיל להאזין לחיבורים לפי כתובת הIP והפורט שצוינו בקובץ הconfig של הפרויקט (פורט ברירת המחדל הוא 22222). לשרת כמות מוגבלת של חיבורים שלהם מוכן להסכים, פרמטר זה מוגדר גם כן על ידי קובץ הconfig וברירת המחדל שלו היא 150. כאשר מתבצע חיבור עם עמית אחר השרת מוסיף אותו למילון העמיתים שאליהם מחובר ביחד עם אובייקט הConnection שיפתח עבור אותו עמית. בעת פתיחת הConnection שני העמיתים עוברים תהליך של תסדיר.

Server - __init__

פעולה זו מאתחלת את אובייקט השרת, ומריצה את התרדים של הפעולות seeker וaccepter

```
def __init__(self, address:str, port:int=22222, max_peer_amount:int=150):
if dreamnail.Server.singleton is not None:
```

```
raise Exception("Singleton class limited to one instance")
dreamnail.Server.singleton = self

self.address = address
self.port = port
self.max_peer_amount = max_peer_amount
self.user_key = dreamnail.singleton.user_data["key"]
self.version = dreamnail.singleton.VERSION
self.blockchain = dreamnail.singleton.blockchain
self.peer_pool = dreamnail.singleton.peer_pool
self.transaction_pool = dreamnail.singleton.transaction_pool

self.difficulty_target = int(2**4) # Static difficulty target
self.peers = {}
self.miner_open = False
self.socket = None
self.chain_lock = threading.Lock()

self.closed = False
self.miner_thread = None
self.seeker_thread = threading.Thread(target=self.seeker)
self.accepter_thread = threading.Thread(target=self.accepter)

dreamnail.singleton.log("Starting server and assigning seeker and accepter
threads")
dreamnail.singleton.log("-------------------------------------------------------")
self.accepter_thread.start()
self.seeker_thread.start()

dreamnail.singleton.log("Server is now running...")
```

roll_peer

מטרת פעולה זו היא להגריל בין העמיתים הקיימים בבריכת העמיתים של האפליקציה.

לכל עמית משויך סטטוס הניתן לו לפי החיבור האחרון איתו. הסטטוסים הקיימים הם: "CONVERSED" שאומר שהחיבור האחרון נעשה בהצלחה, "OFFLINE" שאומר שהחיבור האחרון נכשל ו"UNKNOWN" שאומר שעוד לא בוצע ניסיון חיבור לאותו עמית.

הפעולה ראשית תגריל אקראית עמית בין העמיתים שהסטטוס שלהם הוא אינו "OFFLINE", במידה ואין עמיתים שכאלו, הפעולה תגריל אקראית בין העמיתים שהסטטוס שלהם הוא "OFFLINE". במידה ואין כלל עמיתים להגריל מהם, הפעולה לא תגריל עמית ותחזיר None.

```
def roll_peer(self):
peer_options = []
offline_peer_options = []
for peer, status in self.peer_pool.items():
if status != dreamnail.Server.PEER_STATUS_OFFLINE and peer not in
self.peers.keys():
peer_options.append(peer)
elif status == dreamnail.Server.PEER_STATUS_OFFLINE and peer not in
self.peers.keys():
offline_peer_options.append(peer)
if len(peer_options) > 0:
output = random.choice(peer_options)
dreamnail.singleton.log(f"### Rolled {output} from peer options")
elif len(offline_peer_options) > 0:
output = random.choice(offline_peer_options)
else:
output = None
return output
```

seeker

תפקיד פעולה זו להתחבר לעמיתים השונים הנמצאים בבריכת העמיתים של האפליקציה. הפעולה תגריל עמית להתחבר אליו באמצעות הפונקציה Server.roll_peer. הפעולה תנסה להתחבר לעמית באמצעות הפעולה Server.connect. אם החיבור נענה או כשל, הפעולה תעדכן את סטטוס העמית בהתאם. פעולה זו תמשיך בריצתה עד אשר הדגל Server.closed יהיה שווה לTrue.

```
def seeker(self):
time.sleep(5)
dreamnail.singleton.log(f"Server is now seeking new connections")
try:
while not self.closed:
# Once connection amount is too low, seek connections if possible.
while len(self.peers) < math.floor(self.max_peer_amount*(2/3)) and not self.closed:
```

```
new_peer = self.roll_peer()
if new_peer is not None:
connection_result = self.connect(new_peer)
if connection_result is None:
# TODO: Define peer status system
if self.peer_pool[new_peer] != dreamnail.Server.PEER_STATUS_OFFLINE:
self.peer_pool[new_peer] = dreamnail.Server.PEER_STATUS_OFFLINE
dreamnail.singleton.log(f"### Marked {new_peer} as OFFLINE")
else:
self.peer_pool[new_peer] = dreamnail.Server.PEER_STATUS_CONVERSED
finally:
dreamnail.singleton.log("### Server connection seeker is shutdown.")
```

accepter

תפקיד פעולה זו להאזין לחיבורים מעמיתים אחרים. במידה והתקבל חיבור הפעולה תאתחל אובייקט Connection עם הכתובת והסוקט של החיבור שהתקבל. פעולה זו תמשיך בריצתה עד אשר הדגל Server.closed יהיה שווה לTrue.

```
def accepter(self):
try:
self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
self.socket.bind((self.address, self.port))
except OSError as err:
dreamnail.singleton.close_server()
return
self.socket.listen(self.max_peer_amount)
dreamnail.singleton.log(f"Server is now accepting incoming connections and is binded to
{(self.address, self.port)}")

while not self.closed:
# Do not accept new connections once peer count exceeds maximum
allowed
try:
while len(self.peers) < self.max_peer_amount and not self.closed:
peer_socket, peer_address = self.socket.accept()
dreamnail.Connection(peer_socket, peer_address[0])
```

```
dreamnail.singleton.log(f"### {peer_address[0]} connected to node")
except OSError:
pass
finally:
dreamnail.singleton.log("### Server connection accepter is shutdown.")
```

close

מטרת הפעולה לסגור את השרת. הפעולה תקרא Connection.close עבור כל החיבורים הקיימים בשרת ותגדיר את הדגל Server.closed להיות True.

```
def close(self):
"""Terminated the server and all of its ongoing connections"""
dreamnail.singleton.log("### SHUTTING DOWN SERVER")
self.socket.close()
for conn in list(self.peers.values()):
conn.close()

self.closed = True
dreamnail.Server.singleton = None
```

try_chain_block

תפקיד הפעולה להוות עטיפה לשרשור בלוקים עבור האפליקציה. הפעולה מקבלת Block שאותו תנסה לשרשר לשרשרת הבלוקים הנוכחית של השרת. אם הפעולה צלחה בשרשור; הפעולה תעדכן את הGUI של חוקר השרשרת, תסיר את כל ההעברות בבלוק ששורשר מבריכת ההעברות, תעדכן את היתרה של המשתמש הנוכחי ותשלח את הבלוק לכל העמיתים שמחוברים לשרת מלבד העמיתים שצוינו בפרמטר exclusions.

```python
def try_chain_block(self, block, exclusions:list=None):
    """Attempts to chain a given block to the current blockchain.
    On success sends the block to all connected peers.
    Will not share with peer addresses given in :exclusions:"""
    if exclusions is None:
        exclusions = []
    self.chain_lock.acquire()
    try:
        if self.blockchain.chain_block(block):
            dreamnail.singleton.log(f"### SUCCESFULY CHAINED BLOCK {block.block_hash}")
            if dreamnail.singleton.ui.tabWidget.currentIndex() == 3:
                dreamnail.singleton.updateBlockchainExplorerTab()

            for transaction in block.transactions:
                if "BLOCK" not in transaction.inputs:
                    self.remove_from_transaction_pool(transaction)

                user_address = dreamveil.key_to_address(dreamnail.singleton.user_data["key"])
                new_balance = decimal.Decimal(0)
                input_transactions = {}
                for relevant_transaction_block_index, transaction_signature in self.blockchain.tracklist[user_address][::-1]:
                    for transaction in self.blockchain.chain[relevant_transaction_block_index].transactions:
                        if transaction.signature == transaction_signature:
                            transaction_value = self.blockchain.calculate_transaction_value(transaction, user_address)
                            if transaction_value is not None:
                                new_balance += dreamveil.to_decimal(transaction_value)
                                input_transactions[transaction.signature] = transaction_value
                dreamnail.singleton.user_data["balance"] = new_balance

            current_peer_addresses = list(self.peers.keys())
            for peer_addr in current_peer_addresses:
                if peer_addr not in exclusions:
                    action_thread = threading.Thread(target=self.peers[peer_addr].SENDBK, args=(block,))
                    action_thread.start()
            return True
        else:
            return False
    except KeyError:
        return False
    finally:
        self.chain_lock.release()
```

חיבור – Connection

אובייקט החיבור מטרתו לנהל כל חיבור בין עמית יחיד לעמית יחיד בלבד ואת התקשורת ביניהם. בתחילת החיבור נפתח Connection run thread שבו מאזין החיבור להודעות מהעמית המחובר. לאחר מכן מתקיים תהליך של תסדיר setup שבו שני העמיתים המחוברים מחליפים ביניהם מידע לגבי עצמם. לאחר התסדיר החיבור יכול לשמש לצרכיהם של כל אחד מהעמיתים אשר יכולים לשלוח פקודות לעמית המחובר בהתאם לצרכיהם ומצב השרשרת שלהם.

Connection - __init__

מטרת הפעולה לאתחל את אובייקט הConnection. אם העמית שאליו האובייקט משויך כבר נמצא בServer.peers, כלומר הוא כבר מחובר לשרת תחת אובייקט אחר, הפעולה תסגור את האובייקט הנוכחי.

```
def __init__(self, socket, address):
dreamnail.Connection.connection_lock.acquire()
try:
self.command_lock = threading.Lock()
self.command_lock.acquire()
self.last_message = None
self.socket = socket
self.address = address
self.closed = False
self.peer_chain_mass = None
self.completed_setup = False
try:
self.first_to_move = dreamnail.Server.singleton.address > address
except (AttributeError):
self.close()

if address not in dreamnail.Server.singleton.peers:
dreamnail.Server.singleton.peers[self.address] = self
dreamnail.singleton.add_peer(address)
else:
self.close(remove_peer=False)
return
finally:
dreamnail.Connection.connection_lock.release()
self.run_thread = threading.Thread(target=self.run)
self.run_thread.start()
self.setup()
```

Connection – recv

תפקיד הפעולה לקבל הודעה יחידה מן סוקט החיבור. פורמט הודעה הוא:

Size || Message כאשר Size הוא אורך ההודעה ואורכו שלו קבוע וגודלו 7 תווים. אורך הודעת פקודה הוא קבוע וגודלו 6 תווים. אורך הודעה רגילה הוא כאורך בלוק מקסימלי

2MB (2097152 בייטים). במקרה שאורך ההודעה אינו שבעה ספרות יש לעשות פאד לsize לגודל 7 תווים עם אפסים.

הפעולה קוראת כsize + message בייטים מההודעה ולפי האורך שצוין בsize מחלצת הפעולה את message ומחזירה אותו.

במידה והפעולה נכשלת או מתקבלת הודעה שאינה תקינה היא סוגרת את החיבור.

```
def recv(self):
try:
message =
self.socket.recv(dreamnail.Connection.MAX_MESSAGE_SIZE).decode()
try:
assert len(message) >= dreamnail.Connection.HEADER_LEN
message_len = message[:dreamnail.Connection.HEADER_LEN]
assert len(message_len) == dreamnail.Connection.HEADER_LEN
message_len = int(message_len)
assert message_len >= 0
message_contents =
message[dreamnail.Connection.HEADER_LEN:dreamnail.Connection.HEADER_LEN +
message_len]
assert len(message_contents) == message_len
except (ValueError, AssertionError):
dreamnail.singleton.log(f"Recieved invalid message from ({self.address})")
self.close()
return
dreamnail.singleton.log(f"### Recieved message from ({self.address}):
{message_contents}")
return message_contents
except (ConnectionResetError, ConnectionAbortedError, OSError):
if not self.closed:
self.close()
```

Connection – send

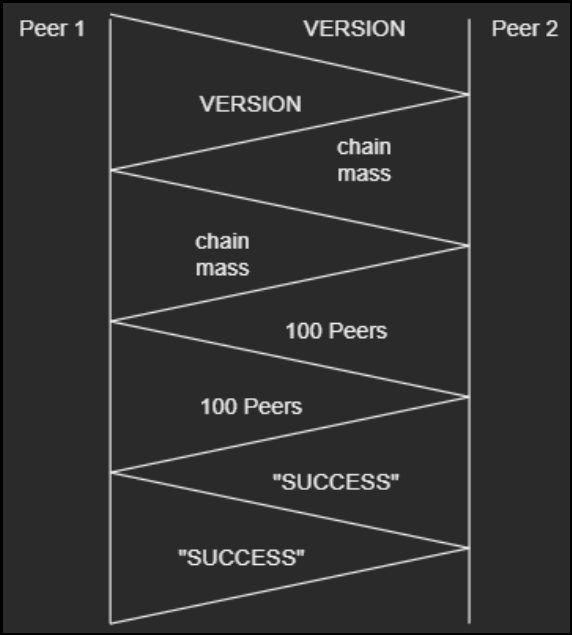
תפקיד הפעולה לשלוח הודעה לעמית המחובר. הפעולה מקבלת str message שהיא ההודעה שיש לשלוח. לפני שליחת ההודעה, הפעולה תוסיף את אורכה לתחילת ההודעה ותעשה לאורך פאד לפי הצורך. הפעולה אינה מחזירה דבר.

במידה והפעולה נכשלת היא סוגרת את החיבור.

```
def send(self, message:str):
try:
assert len(message) <= dreamnail.Connection.MAX_MESSAGE_SIZE

dreamnail.singleton.log(f"### Sending message to ({self.address}):
{message}")
if not self.closed:
message = str(len(message)).zfill(dreamnail.Connection.HEADER_LEN) +
message
self.socket.send(message.encode())
except Exception as err:
dreamnail.singleton.log(f"Failed to send message to {self.address} due to
error: {type(err)}: {err.args}")
self.close()
raise
```

תסדיר - setup

התסדיר מתקיים בעת פתיחת חיבור. והוא תהליך של העברת מידע על כל אחד מהעמיתים. יש לשים לב שמכיוון שהתוכנה היא סימטרית, על מנת לשמור על סדר דיבור הגיוני, נגדיר שהעמית הראשון לדבר הוא האחד בעל כתובות הIP הגדולה יותר. להלן תרשים של שיחת התסדיר:



ראשית מחליפים שני העמיתים את גרסת האפליקציה שלהם. העמיתים ממשיכים בשיחה רק אם הגרסה שלהם זהה, אחרת החיבור נסגר. הגרסה מוגדרת בקובץ הconfig של הפרויקט.

לאחר מכן הם מחליפים את מסת השרשראות שלהם, כל עמית שומר את מסת השרשרת של העמית שאליו הוא מחובר באובייקט הConnection.

לאחר מכן כל אחד מהם שולח עד 100 עמיתים בצורה של כתובות IP מברכת העמיתים המוכרים שלו אשר נבחרו אקראית. כל עמית יוסיף לברכת העמיתים המוכרים שלו את העמיתים ששותפו אתו כעת. שיתוף זה מרחיב את כמות המחשבים שעמית מסוים מכיר ברשת ובכך תורם לפיזור שלה.

לסיום כל עמית שולח "SUCCESS" להראות שהתהליך הסתיים בהצלחה.

```
def setup(self):
try:
# Check that node versions match
peer_version = None
if self.first_to_move:
self.send(dreamnail.Server.singleton.version)
peer_version = self.read_last_message()
else:
peer_version = self.read_last_message()
self.send(dreamnail.Server.singleton.version)
assert peer_version == dreamnail.Server.singleton.version

# Exchange chain masses
if self.first_to_move:
self.send(f"{dreamnail.Server.singleton.blockchain.mass}")
peer_chain_mass = self.read_last_message()
else:
peer_chain_mass = self.read_last_message()
self.send(f"{dreamnail.Server.singleton.blockchain.mass}")
peer_chain_mass = int(peer_chain_mass)
assert peer_chain_mass >= 0
self.peer_chain_mass = peer_chain_mass

# Send and recieve 100 random peers to further establish the connection of
nodes into the network
```

```
peers_to_share =
random.sample(list(dreamnail.Server.singleton.peer_pool.keys()), min(100,
len(dreamnail.Server.singleton.peer_pool)))
if self.first_to_move:
self.send(json.dumps(peers_to_share))
newly_given_peers = self.read_last_message()
else:
newly_given_peers = self.read_last_message()
self.send(json.dumps(peers_to_share))
newly_given_peers = json.loads(newly_given_peers)
assert len(newly_given_peers) <= 100 and type(newly_given_peers) == list
for peer in newly_given_peers:
assert ipaddress.ip_address(peer)
if peer not in dreamnail.Server.singleton.peer_pool and peer !=
dreamnail.Server.singleton.address:
dreamnail.Server.singleton.peer_pool[peer] =
dreamnail.Server.PEER_STATUS_UNKNOWN

if self.first_to_move:
self.completed_setup = True
self.send("SUCCESS")
self.read_last_message()
time.sleep(0.05)
else:
self.completed_setup = True
self.read_last_message()
self.send("SUCCESS")
time.sleep(0.05)

if self.peer_chain_mass >= dreamnail.Server.singleton.blockchain.mass +
dreamnail.Server.singleton.difficulty_target * dreamnail.Server.TRUST_HEIGHT:
dreamnail.singleton.log(f"### Noticed that we use a significantly larger
chain than {self.address} (dM-chain = {dreamnail.Server.singleton.blockchain.mass -
self.peer_chain_mass} Starting to sync with it")
chnsyn_thread = threading.Thread(target=self.CHNSYN)
chnsyn_thread.start()

dreamnail.singleton.log(f"### Connection with {self.address} completed setup
(version: {peer_version})")
```

```
self.command_lock.release()
except (AssertionError, TimeoutError, ValueError) as err:
dreamnail.singleton.log(f"!!! Failed to initialize connection in setup with
{self.address} (ver: {peer_version}) due to {type(err)}: {err.args}")
# Terminate the connection
self.close()
```

Connection – run

כל עוד החיבור פתוח באובייקט החיבור רץ תרד השייך לחיבור על פעולת הRun של החיבור. הפעולה מאזינה להודעות מן העמית באמצעות פעולת הrecv, כאשר מתקבלת הודעה הפעולה תריץ את Connection.execute_command אשר תבצע את הפקודה שהלקוח שלח במידה וההודעה אכן פקודה. הפעולה תשמור את ההודעה במאפיין באובייקט החיבור הנקרא last_message כדי שפעולות אחרות באובייקט יוכלו לגשת אליו בקלות. במידה והתקבל "TERMINATE", הפעולה סוגרת את החיבור. הפעולה לא תריץ את execute_command אילולא החיבור צלח את התסדיר.

```
def run(self):
while not self.closed:
try:
command_message = self.recv()
if command_message == "TERMINATE":
self.close()
break
self.last_message = command_message
if self.completed_setup:
cmd_thread = threading.Thread(target=self.execute_command,
args=(command_message,))
cmd_thread.start()
except Exception as err:
dreamnail.singleton.log(f"!!! Connection at {self.address} failed and forced
to close due to {err}.")
self.close()
```

Connection – close

מטרת הפעולה לסגור את החיבור הנוכחי. הפעולה תגדיר את הדגל Connection.closed לTrue ותסיר את העמית שאליו האובייקט משויך מServer.peers המילון אשר שומר את העמיתים המחוברים ואת אובייקטי הConnection שלהם.

```
def close(self, remove_peer=True):
if self.closed:
return
try:
self.closed = True
dreamnail.singleton.log(f"### Terminated connection with {self.address}")
self.send("TERMINATE")
if self.address in dreamnail.Server.singleton.peers and remove_peer:
del dreamnail.Server.singleton.peers[self.address]
dreamnail.singleton.remove_peer(self.address)
finally:
self.socket.close()
```

Connection – read_last_message

פעולה זו מנסה לקרוא את הערך Connection.last_message אשר תחילה מאותחל לNone. הפעולה תקרא את הערך רק כאשר ערכו אינו None (הפעולה run קיבלה הודעה ושינתה אותו). כאשר last_message אינו None הפעולה קוראת את הערך, מאפסת אותו חזרה לNone ומחזירה את ערכו לפני האיפוס.

כל עוד last_message שווה לNone הפעולה תחכה עד שערכו אינו None או שעברו 15 שניות. אם עברו 15 שניות ושום ערך לא נקרא הפעולה תעלה שגיאה (אך לא תסגור את החיבור)

```
def read_last_message(self, timeout=15.0):
start = timeit.default_timer()
while self.last_message is None:
if self.closed:
return
if timeit.default_timer() - start >= timeout:
raise TimeoutError("Did not recieve answer from peer")
output = self.last_message
self.last_message = None
```

```
return output
```

Connection – execute_command

הפעולה מקבלת הודעה str message ובמידה וההודעה היא פקודה ידועה, הפעולה תפתח בשיחת פקודה עם העמית כדבר הפקודה. במידה והפקודה צלחה הפעולה תחזיר True אחרת אם הפקודה נכשלה תחזיר False.

פקודות Connection

לכל פקודה אפשרית יש קוד משלה שהוא שמה של הפקודה מקוצר לשישה תווים. הפקודות החוקיות בחיבור הן SENDTX שליחת העברה, SENDBK שליחת בלוק ולבסוף CHNSYN סנכרון שרשראות בלוקים.

לכל פקודה אפשרית גם צמד שני פעולות המבצעות אותה בין העמיתים, פעולה של פוקד ונפקד. פעולות הנפקד מקומם בתוך execute_command. ופעולות הפוקד הם פעולות של Connection.

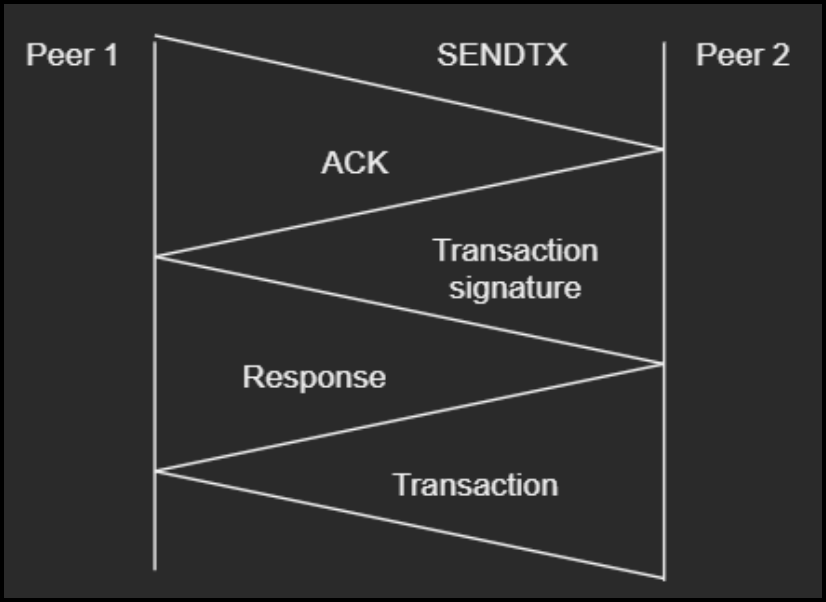
כל פעולת פקודה פוקדת תמיד ראשית שולחת את קוד הפקודה אל העמית לפני תחילת ביצועה.

```
def connection_command(command_func):
def wrapper(self, *args, **kwargs):
self.command_lock.acquire()
try:
dreamnail.singleton.log(f"### Locked {command_func.__name__} in
{self.address}")
self.send(command_func.__name__)
output = command_func(self, *args, **kwargs)
return output
except Exception as err:
dreamnail.singleton.log(f"!!! Failed commanding {self.address} due to error
{err}. {command_func}")
finally:
time.sleep(0.05)
dreamnail.singleton.log(f"{command_func} finished {self.address}")
self.command_lock.release()
return wrapper
```

SENDTX

מטרת הפקודה היא לשתף בהעברה שהגיע אל הפוקד, הפוקד מעוניין לשלוח את ההעברה לנפקד

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח את קוד הפקודה לנפקד והנפקד מחזיר תשובה שקיבל את ההודעה. לאחר מכן הפוקד שולח את החתימה הדיגיטלית של ההעברה והנפקד מחזיר כתשובה אם הוא רוצה לקבל את ההעברה. אם כן הפוקד שולח את ההעברה אחרת לא נשלח דבר והפקודה הסתיימה.

הנפקד מעוניין בהעברה כל עוד הוא אינו מכיר אותה בעצמו או היא איננה על שרשרת הבלוקים שלו.

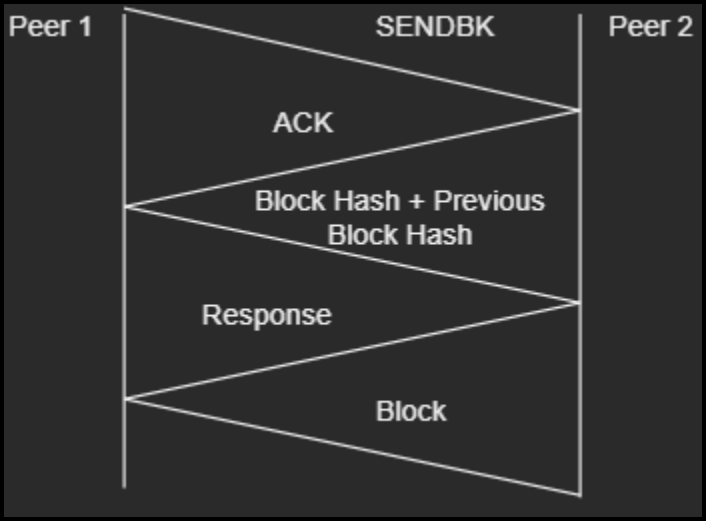
לאחר שההעברה התקבלה על ידי הפוקד, הפוקד יאמת שהיא תקינה ושהחתימה הדיגיטלית שלה שווה לזו שציין הפוקד לפני שליחתה. אם ההעברה אומתה הנפקד יוסיף אותה אל בריכת ההעברות שלו וישתף אותה (יריץ SENDTX) עם כל שאר עמיתיו שהם אינם הפוקד. אחרת אם החתימה הדיגיטלית אינה זהה, הפעולה תסגור את החיבור.

```
@connection_command
def SENDTX(self, transaction:dreamveil.Transaction):
assert self.read_last_message() == "ACK"
self.send(transaction.signature)
ans = self.read_last_message()
if ans == "True":
self.send(transaction.dumps())
```

SENDBK

מטרת הפעולה היא לשתף בלוק ששורשר בצד הפוקד אל הנפקד. הפוקד מעוניין לשלוח את הבלוק לנפקד. עמית מעוניין לשלוח בלוק לעמית אחר רק כאשר שרשר בלוק חדש.

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח את קוד הפקודה לנפקד והנפקד מחזיר תשובה שקיבל את ההודעה. לאחר מכן הפוקד שולח את הhash של הבלוק וגם את הhash של הבלוק הקודם לבלוק שאותו מעוניין לשלוח והנפקד מחזיר כתשובה אם הוא רוצה לקבל את הבלוק. אם כן הפוקד שולח את הבלוק אחרת לא נשלח דבר והפקודה הסתיימה.

הנפקד מעוניין בבלוק רק עם הprevious_block_hash שנשלח שווה לhash של הבלוק העליון בשרשרת הנוכחית של הנפקד (הבלוקים מסוגלים להשתרשר).

לאחר שהבלוק התקבל על ידי הפוקד, הנפקד ינסה לשרשר אותו לשרשרת שלו. (באמצעות הפונקציה Server.try_chain_block)

כאשר מתקבל הBlock hash מן הפוקד, הנפקד מעדכן את peer_chain_mass באובייקט החיבור שלו ומוסיף לו את גודל הProof of work של הhash שקיבל.

```
@connection_command
def SENDBK(self, block:dreamveil.Block):
assert self.read_last_message() == "ACK"
self.send(block.get_header())
ans = self.read_last_message()
if ans == "True":
self.send(block.dumps())
```

CHNSYN

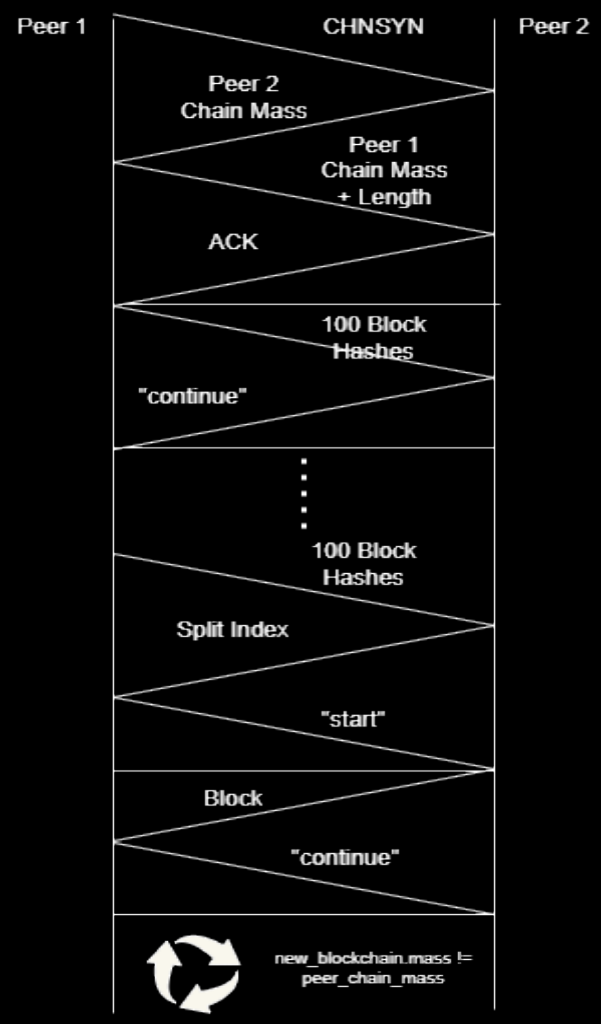
מטרת הפעולה היא לסנכרן את השרשרת של הפוקד עם השרשרת הכבדה יותר של הנפקד. עמית מעוניין להסתנכרן עם שרשרת של עמית אחר רק כאשר שם לב שהשרשרת של עמית שאליו מחובר כבדה משמעותית מהשרשרת שלו. שרשרת כבדה משמעותית מהשרשרת הנוכחית כאשר מתקיים:

peer_chain_mass >= my_chain_mass * difficulty_target * TRUST_HEIGHT

כאשר difficulty_target זה קושי הProof of work המינימלי שנדרש על מנת לשרשר בלוק

ו TRUST_HEIGHT זה קבוע שגודלו 6 שמטרתו למנוע משרשראות להסתנכרן מוקדם מדי כשעדיין לא ברור שהן השרשרת הפופולרית ביותר האמיתית (אני ארחיב על זה בהמשך)

להלן תרשים המתאר את התקשורת בין הפוקד (Peer 1) לבין הנפקד (Peer 2)



ראשית הפוקד שולח לנפקד את קוד הפקודה. לאחר מכן הנפקד שולח את מסת השרשרת שלו. הפוקד שולח בחזרה את מסת השרשרת של עצמו ומספר הבלוקים בה והנפקד משיב עם "ACK".

לאחר מכן הפוקד שולח את מאה הBlock hashes העליונים בשרשרת שלו (או פחות אם אין מספיק). במידה והנפקד מוצא הצטלבות בין אחד מהBlock hashes שקיבל לבין בלוק בשרשרת שלו ידוע כי באינדקס אחד אחרי, זוהי נקודת הפיצול בין השרשראות (Split Index). אם פיצול נמצא הוא מוחזר על ידי הנפקד, אחרת הנפקד עונה "continue" והפוקד שולח את מאה הבלוקים הבאים עד אשר הSplit Index נמצא או שנגמרו הבלוקים בשרשרת הפוקד או הנפקד.

אם נמצא Split Index זה אומר שהשרשראות של העמיתים נבעו מבסיס משותף ואין צורך לשלוח את החלק המשותף.

אם לא נמצא הנפקד מחזיר את הערך 0 שאומר שאין נקודת פיצול משום שהשרשראות שונות.

לאחר מכן הפוקד יוצר אובייקט שרשרת חדשה שעתידה להחליף את השרשרת הנוכחי והפוקד משרשר אליה את כל הבלוקים שכבר ידועים לו לפי Split Index.

לאחר מכן הפוקד שולח "start" המעיד על מוכנותו להתחיל לקבל הזרמה של בלוקים שישרשר אל השרשרת החדשה.

הנפקד מתחיל בלשלוח בלוקים מהשרשרת שלו. הוא שולח בלוק אחד עבור כל הודעה והוא מתחיל בשליחת הבלוק בSplit Index ועולה מעלה. בתשובה לכל בלוק הפוקד שולח "continue" בשביל להראות שהוא מוכן לקבל את הבלוק הבא. כאשר הנפקד מקבל בלוק הוא משרשר אותו לשרשרת החדשה. אם השרשור נכשל הפוקד סוגר את החיבור. התהליך חוזר חלילה עד אשר מסת השרשרת החדשה שווה למסה שהובטחה לו בתחילת הפקודה על ידי הנפקד. במידה והשרשרת החדשה איננה בגודל שהובטח (הנפקד פסק לספק בלוקים באמצע התהליך) השרשרת החדשה נמחקת והחיבור נסגר.

בסוף הפקודה כאשר השרשרת החדשה התקבלה במלואה, אנו מעבירים אליה את הwallet tracklist של השרשרת הישנה ואז אנו מחליפים את השרשרת הישנה בחדשה.

```
@connection_command
def CHNSYN(self):
"""Syncs ourselves with the peer's larger chain"""
peer_chain_mass = self.read_last_message()
peer_chain_mass = int(peer_chain_mass)
assert peer_chain_mass > 0
self.peer_chain_mass = peer_chain_mass
my_chain_mass = dreamnail.Server.singleton.blockchain.mass
```

```
my_chain_len = len(dreamnail.Server.singleton.blockchain.chain)

if peer_chain_mass >= my_chain_mass +
dreamnail.Server.singleton.difficulty_target * dreamnail.Server.TRUST_HEIGHT:
# Locate the split where the current blockchain is different from the proposed
blockchain by the peer.
self.send(f"{my_chain_mass} {my_chain_len}")
assert self.read_last_message() == "ACK"
hash_batches_sent = 0
split_index = "continue"
while split_index == "continue":
# We send a block hash batch to the peer (max length 100)
# The peer will match the hashes against his own chain to find where they
split
# Repeats this proccess until the split is found.
# split_index: index on the chain where the blocks are different but on
split_index-1 are the same for both chains.
hash_batch = [block.block_hash for block in
dreamnail.Server.singleton.blockchain.chain[::-
1][100*hash_batches_sent:100*(hash_batches_sent+1)]]
hash_batch = hash_batch[::-1]
self.send(" ".join(hash_batch))
split_index = self.read_last_message()
split_index = int(split_index)
assert split_index >= 0 and split_index <= my_chain_len
form_new_chain = split_index < my_chain_len and my_chain_len != 0

# Create the new blockchain object and fill in the known blocks
if form_new_chain:
new_blockchain = dreamveil.Blockchain()
for i in range(split_index):
new_blockchain.chain_block(dreamnail.Server.singleton.blockchain.chai
n[i])
else:
new_blockchain = dreamnail.Server.singleton.blockchain

self.send("start")
# Download all the blocks mentioned in the inventory list from the peer
try:
```

```
while new_blockchain.mass != peer_chain_mass:
new_bk = dreamveil.Block.loads(self.read_last_message())
chain_result = new_blockchain.chain_block(new_bk)

if chain_result:
self.send("continue")
else:
dreamnail.singleton.log(f"!!! Block recieved in CHNSYN from
({self.address}) failed to chain. Using new blockchain: {form_new_chain}.")
if form_new_chain and not new_blockchain is
dreamnail.Server.singleton.blockchain:
del new_blockchain
self.close()
return

# We swap the blockchain objects to the new larger one.
new_blockchain.tracklist =
dreamnail.Server.singleton.blockchain.tracklist.copy()
dreamnail.singleton.blockchain = new_blockchain
self.blockchain = dreamnail.singleton.blockchain

# Remove all of the outdated pool transactions
for block in new_blockchain.chain:
for transaction in block.transactions:
dreamnail.Server.singleton.remove_from_transaction_pool(transaction
.signature)

dreamnail.singleton.log(f"### With ({self.address}) finished syncing new
chain with mass {dreamnail.Server.singleton.blockchain.mass} and length
{len(dreamnail.Server.singleton.blockchain.chain)} (old: {my_chain_mass})")
except Exception as err:
dreamnail.singleton.log(f"!!! Error {err} while getting blocks from peer in
CHNSYN ({self.address}). specified: {peer_chain_mass.mass} given:
{new_blockchain.mass}")
if form_new_chain and not new_blockchain is
dreamnail.Server.singleton.blockchain:
del new_blockchain
return False
else:
```

```
# We are not interested in the chain of the peer.
self.send("False")
```

טעינה ופריקה של אובייקטי Dreamveil – Transaction, Block, Blockchain

ידוע כי בפקודות שהוגדרו מעלה ואחרות בפרויקט קיים הצורך בשמירת קבצי הפרויקט והצורך לשלוח לקבל אובייקטי Dreamveil. על מנת להמיר אובייקט Transaction, Block או Blockchain לטקסט יש להשתמש במתודה dumps של כל אחד מהאובייקטים (לדוגמא: transaction.dumps()). הפעולה תמיר את האובייקט לstr שבטוח לשלוח ולשמור לקובץ.

על מנת לטעון אובייקט מתוך הטקסט הפרוק, יש לקרוא למתודה הסטטית loads עבור סוג האובייקט הרצוי (דוגמא: Transaction.loads(transaction.json)).

יש לשים לב כי loads גם בודק את אמיתות האובייקטים וערכיהם בעת הטעינה ובמידה והאובייקט איננו תקין הפעולה תחזיר None ולא אובייקט מהסוג הצפוי.

בחרתי בשיטה זו לאימות אובייקטים זרים מכיוון שתמיד שמקבלים אובייקט זר בהכרח נצטרך לטעון אותו, שיטה זו מסירה מן הספק שיתקבל אובייקט שאיננו תקין כתקין.

בנוסף חשוב לציין כי כאשר מייצרים את האובייקט ישירות (באמצעות הinit הרלוונטי), התוכנה לא תבדוק את נכונות הפרמטרים. זהו מכוון על מנת לאפשר בנייה אישית של אובייקטים בזמן אמת, אך יש לשים לב שהאובייקט הבנוי אכן תקין לפי שימושו!

```
def dumps(self):
information = [self.sender, self.inputs, self.outputs, self.message, self.nonce,
self.signature]
output = json.dumps(information)
if type(Transaction.loads(output)) != type(self):
print("WARNING dumped transaction is invalid!")
return output
@staticmethod
def loads(json_str:str):
try:
assert len(json_str) < Transaction.MAX_TRANSACTION_SIZE
information = json.loads(json_str)
assert type(information) == list
assert len(information) == 6
```

```
assert type(information[0]) == str and type(address_to_key(information[0])) == RSA.RsaKey
assert type(information[1]) == dict
assert type(information[2]) == dict
assert type(information[3]) == str and len(information[3]) <= 222
assert type(information[4]) == str and len(information[4]) == 64
assert type(information[5]) == str and len(information[5]) == 512

transaction_object = Transaction(*information)
assert transaction_object.verify_io()
assert transaction_object.verify_signature()
return transaction_object
except Exception as err:
# print(f"Transaction rejected due to {type(err)}: {err.args}")
return None
```

מנעולי תרדים

על מנת לאפשר עבודה תקינה במולטי-תרדינג יש למנוע מפעולות מסוימות לעבוד במקביל על מנת ליצור סדר הגיוני בתוכנית.

chain_lock

במחלקת Server קיים המנעול chain_lock שמטרתו לשמור שלא יהיו מספר בלוקים המשורשרים בו זמנית מכיוון ששרשרת הבלוקים לא יכולה להיבנות באופן מקבילי ולסדר שרשור הבלוקים יש חשיבות רבה, לכן על מנת שהתהליך לא יכשל עקב שרשור מהיר של בלוקים מנעול זה ינעל את הפונקציה try_chain_block בתחילתה וישתחרר לפני צאתה.

Connection.connection_lock

במחלקת Connection קיים המנעול הסטטי connection_lock שמטרתו למנוע מכמה אובייקטי Connections להיות מאותחלים בו זמנית, זאת על מנת למנוע מכמה אובייקטים השייכים לאותו עמית לעבור את הבדיקה הבודקת אם העמית כבר קיים בServer.peers.

המנעול ננעל בinit של Connection ומשוחרר לפני צאת הפעולה Connection.setup.

command_lock

במחלקת Connection קיים המנעול command_lock שמטרתו למנוע ממספר פקודות לפעול במקביל. כל פקודה דורשת שיחה משלה ושתי פקודות לא מסוגלות לתפקד במקביל, הדבר יגרום לכשל. לשם כך כל פקודה תנעל את מנעול זה בתחילתה ותשחרר אותו לפני צאתה. המנעול גם ננעל בinit של הConnection ומשוחרר לפני צאת התסדיר (setup).

Dreamnail – Node Application

על מנת שמשתמש יוכל לעשות שימוש בבלוקצ'יין, עליו להשתמש באפליקציה המיישמת ותומכת בעקרונות שהגדרנו. Dreamnail היא אפליקציית הNode בפרויקט זה והיא מנגישה למשתמש את היכולת לבצע את כלל הפעולות ש – Dreamveil הבלוקצ'יין מציעה באמצעות ממשק משתמש גרפי, עורכי ההעברות, צג שרשרת וניהול ארנקים.

הדף הראשי באפליקציה:

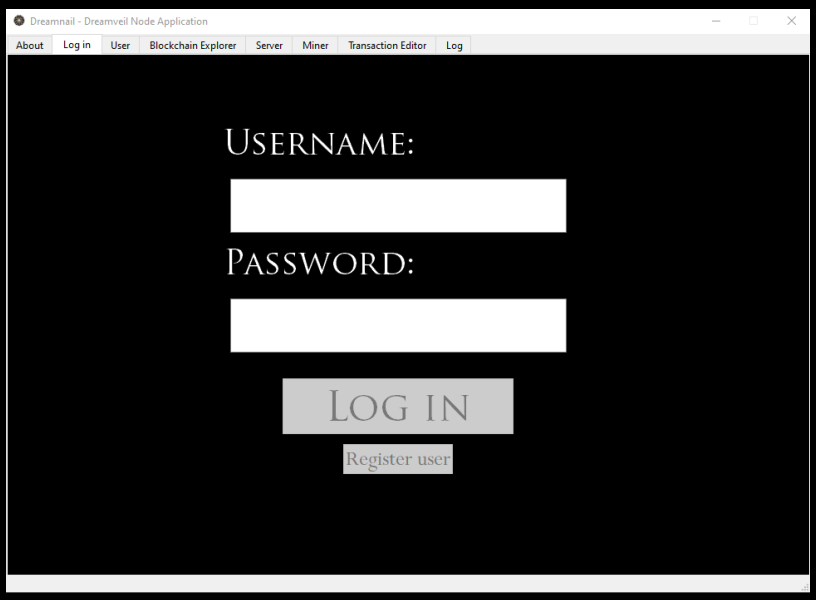
Dreamnail – __init__

פעולת האתחול של האפליקציה. מבצעת אתחול לאובייקט האפליקציה, לGUI ומקשרת את הGUI Events לפונקציות המתאימות הכתובות בתוך המחלקה. פעולת האתחול גם טוענת את קובץ הconfig של האפליקציה "node.cfg". כמו כן הפעולה גם טוענת את קובץ הבלוקצ'יין המאכסן את השרשרת הנוכחית, את קובץ בריכת העמיתים המאכסן את כתובות הIP של כל העמיתים המוכרים ביחד עם המצב האחרון שידוע עליהם וקובץ בריכת ההעברות המאכסן את כל ההעברות שעדיין לא הוכנסו לשרשרת הבלוקים.

```python
def __init__(self):
    if dreamnail.singleton is not None:
        raise Exception("Singleton object limited to one instance.")
    dreamnail.singleton = self
    QtCore.QDir.addSearchPath("resources", APPLICATION_PATH + "/resources/")
    self.exited = False
    atexit.register(self.exit_handler)

    self.app = QApplication(sys.argv)
    self.win = QMainWindow()
    self.ui = dreamui.Ui_MainWindow()
    self.ui.setupUi(self.win)

    self.win.closeEvent = lambda event: self.exit_handler()
    self.ui.tabWidget.currentChanged.connect(self.tabWidget_currentChanged)
    self.ui.loginButton.clicked.connect(self.loginButton_clicked)
    self.ui.logoutButton.clicked.connect(self.logoutButton_clicked)
    self.ui.walletAddressCopyButton.clicked.connect(self.walletAddressCopyButton_clicked)
    self.ui.serverStateButton.clicked.connect(self.serverStateButton_clicked)
    self.ui.registerButton.clicked.connect(self.registerButton_clicked)
    self.ui.usernameLineEdit.textChanged.connect(self.usernameLineEdit_textChanged)
    self.ui.passwordLineEdit.textChanged.connect(self.passwordLineEdit_textChanged)
    self.ui.minerMsgTextEdit.textChanged.connect(self.minerMsgTextEdit_textChanged)
    self.ui.minerStateButton.clicked.connect(self.minerStateButton_clicked)
    self.ui.peerPoolComboBox.currentIndexChanged.connect(self.peerPoolComboBox_currentIndexChanged)
    self.ui.blockchainNextButton.clicked.connect(self.blockchainNextButton_clicked)
    self.ui.blockchainPreviousButton.clicked.connect(self.blockchainPreviousButton_clicked)
    self.ui.gotoBlockButton.clicked.connect(self.gotoBlockButton_clicked)
    self.ui.Block1TransactionCombobox.currentTextChanged.connect(self.Block1TransactionCombobox_currentTextChanged)
    self.ui.Block2TransactionCombobox.currentTextChanged.connect(self.Block2TransactionCombobox_currentTextChanged)
    self.ui.Block3TransactionCombobox.currentTextChanged.connect(self.Block3TransactionCombobox_currentTextChanged)
    self.ui.Block4TransactionCombobox.currentTextChanged.connect(self.Block4TransactionCombobox_currentTextChanged)
    self.ui.TransactionAddOutputButton.clicked.connect(self.TransactionAddOutputbutton_clicked)
    self.ui.TransactionRemoveOutputButton.clicked.connect(self.TransactionRemoveOutputButton_clicked)
    self.ui.TransactionEditAddressLineEdit.textChanged.connect(self.TransactionEditorLineEdit_textChanged)
    self.ui.TransactionEditValueLineEdit.textChanged.connect(self.TransactionEditorLineEdit_textChanged)
    self.ui.TransactionOutputSelectCombobox.currentTextChanged.connect(self.TransactionOutputSelectCombobox_currentTextChanged)
    self.ui.transactionMsgTextEdit.textChanged.connect(self.TransactionMsgTextEdit_textChanged)
    self.ui.createTransactionButton.clicked.connect(self.createTransactionButton_clicked)

    self.ui.userLabel.setStyleSheet("QLabel { color: white; }")
    self.ui.balanceLabel.setStyleSheet("QLabel { color: white; }")

    self.ui.peersConnectedLabel.setStyleSheet("QLabel { color: white; }")
    self.ui.peerPoolLabel.setStyleSheet("QLabel { color: white; }")
    self.ui.peerStatusLabel.setStyleSheet("QLabel { color: white; }")

    self.ui.hashRateLabel.setStyleSheet("QLabel { color: white; }")

    self.ui.blockchainMassLabel.setStyleSheet("QLabel { color: white; }")
    self.ui.Block1HashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block1PrevHashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block2HashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block2PrevHashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block3HashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block3PrevHashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block4HashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")
    self.ui.Block4PrevHashLabel.setStyleSheet("QLabel { color: black; background: lightGray; font-size: 10pt; }")

    self.application_config = configparser.ConfigParser()
    self.application_config.read(APPLICATION_PATH + "\\node.cfg")
    self.VERSION = self.application_config["METADATA"]["version"]

    self.user_data = dreambench.USER_DATA_TEMPLATE.copy()
    self.miner_msg = ""
    self.edited_transaction = None

    self.server = None

    dreamnail.singleton.log("Loading bench from saved files...")
    self.blockchain, self.peer_pool, self.transaction_pool = dreambench.load_bench()
    for peer_address in self.peer_pool.keys():
        self.add_to_peer_pool_gui(peer_address)
    dreamnail.singleton.log("Finished loading bench")

    self.win.show()
    sys.exit(self.app.exec())
```

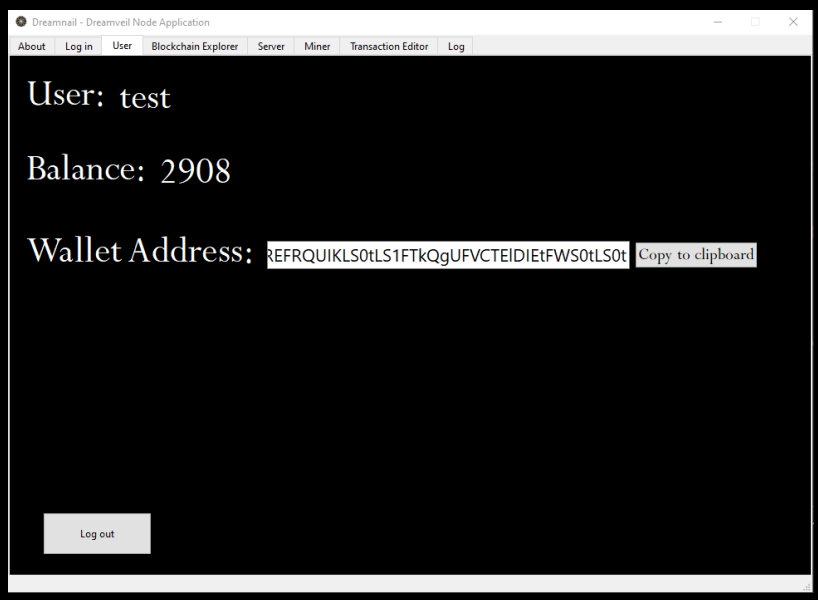
דפי האפליקציה

דף ההתחברות



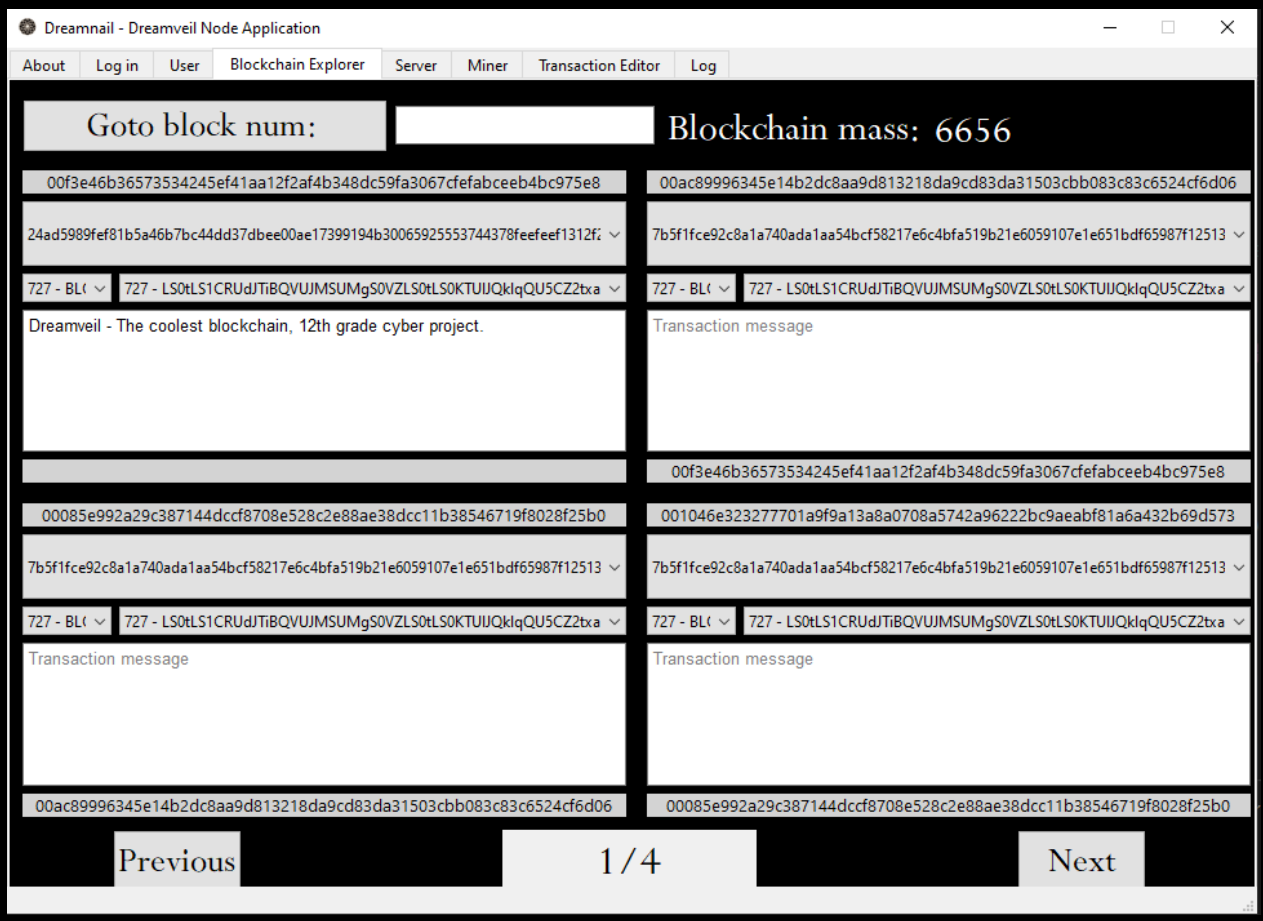
בדף זה המשתמש יכול להתחבר לארנק או ליצור אחד. שם המשתמש יהיה כשם קובץ המשתמש בעוד שהסיסמה תשומש על מנת להצפין את קובץ המשתמש. בלחיצה על הכפתור Log in האפליקציה תנסה לטעון את קובץ המשתמש ואת הארנק המשויך אליו. לחיצה על הכפתור Register תנסה ליצור משתמש חדש.

דף המשתמש



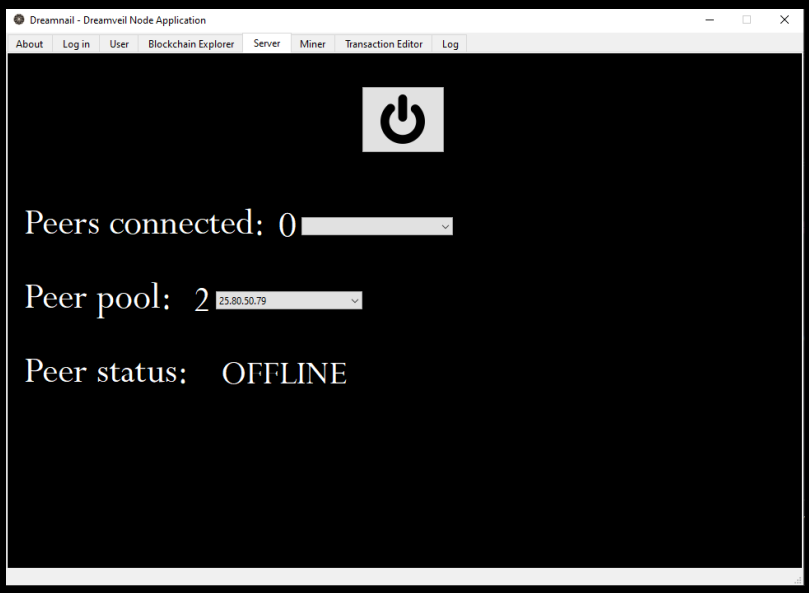
בדף זה המשתמש יכול לראות את פרטי המשתמש שלו: שם המשתמש, יתרת הארנק שלו וכתובת הארנק (הכתובת היא ציבורית ומשמשת להעברת כספים). בדף זה המשתמש גם יכול להתנתק מחשבונו.

דף חוקר השרשרת



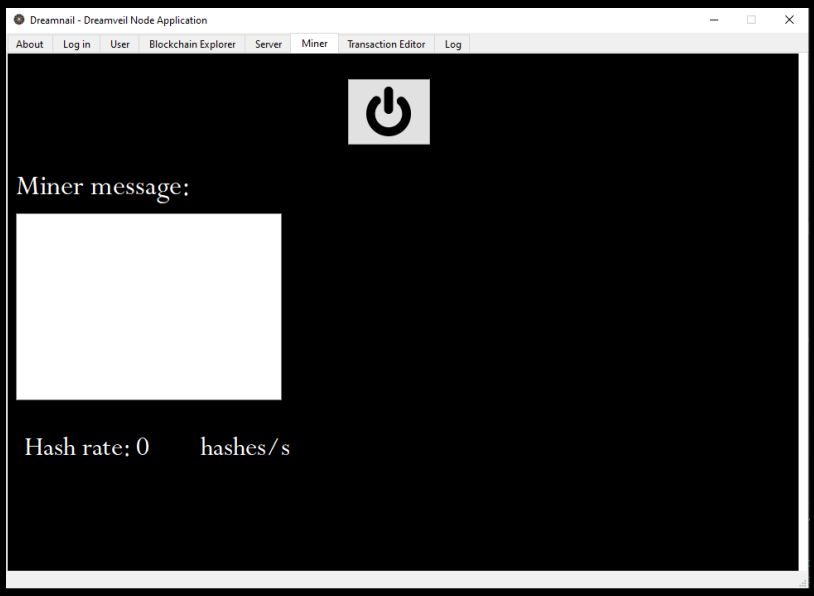
בדף זה יוכל המשתמש לעיין בשרשרת הנוכחית, לראות את הבלוקים הנמצאים בה ואת פרטיהם וגם את ההעברות הנמצאות בהם. ניתן לנווט לבלוק לפי מקומו בשרשרת או לפי ניווט באמצעות כפתורי הNext ו Previous וגם ניתן לראות את גודל מסת השרשרת הנוכחית.

דף השרת



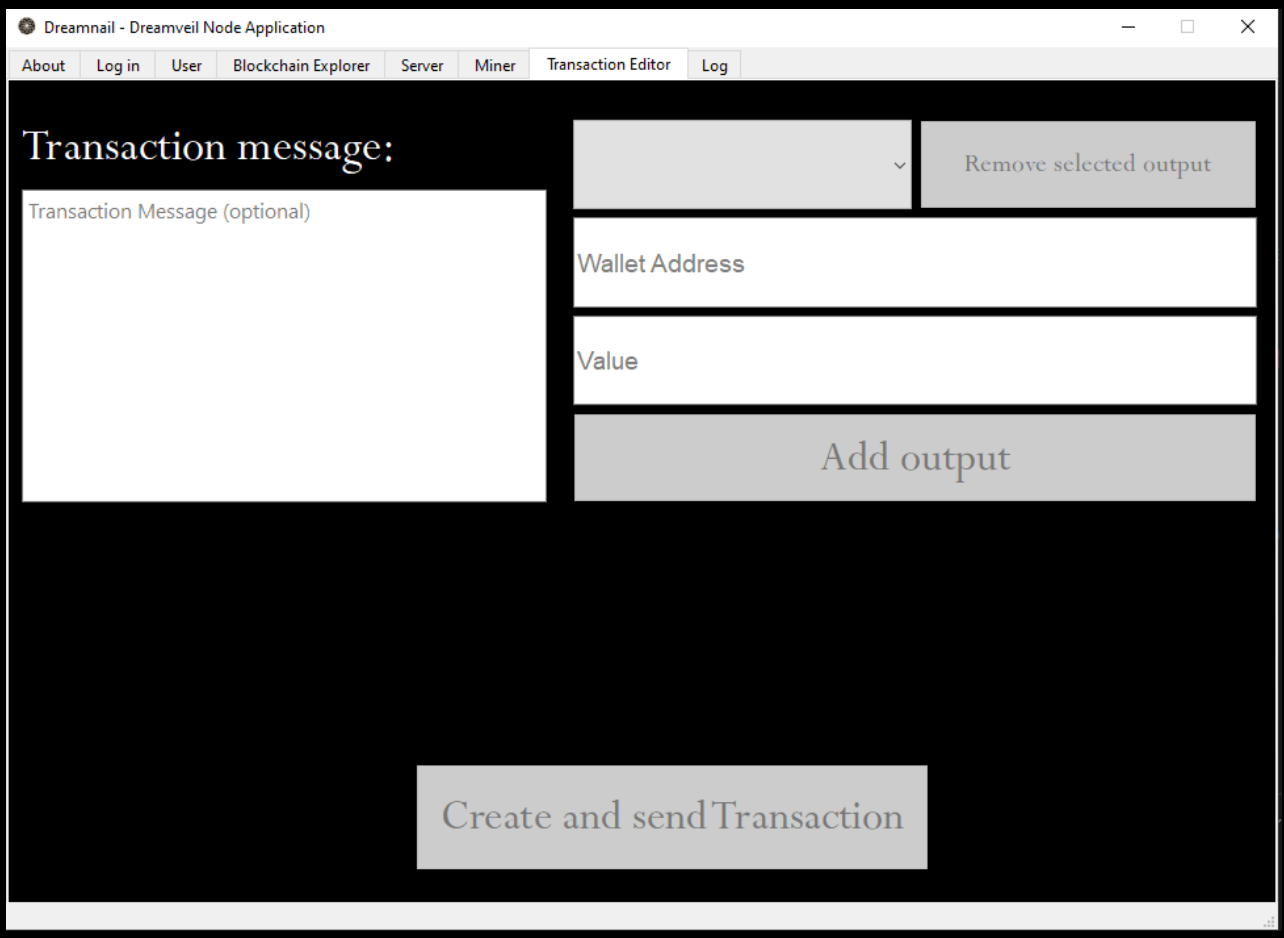
בדף זה המשמש יכול לכבות ולהדליק את השרת, לראות לאלו עמיתים הוא מחובר ולראות אילו עמיתים קיימים בבריכת העמיתים שלו ומה הסטטוס של כל אחד מהם.

דף הכורה



בדף זה יוכל המשתמש להדליק את כורה האפליקציה אשר יכרה באופן אוטומטי בלוקים. מכיוון שלכל בלוק יש העברת פרס לכורה, יוכל המשתמש להגדיר את הודעת העברת הפרס. הכורה יוסיף באופן אוטומטי את ההעברות הכי רווחיות הנמצאות בבריכת ההעברות (ההעברות בעלות מס הכורה הגבוה ביחס לאורכם). כמו כן המשתמש יוכל לראות צג המשקף את כמות הHashes שהאפליקציה מייצרת לשנייה (מהירות כרייה).

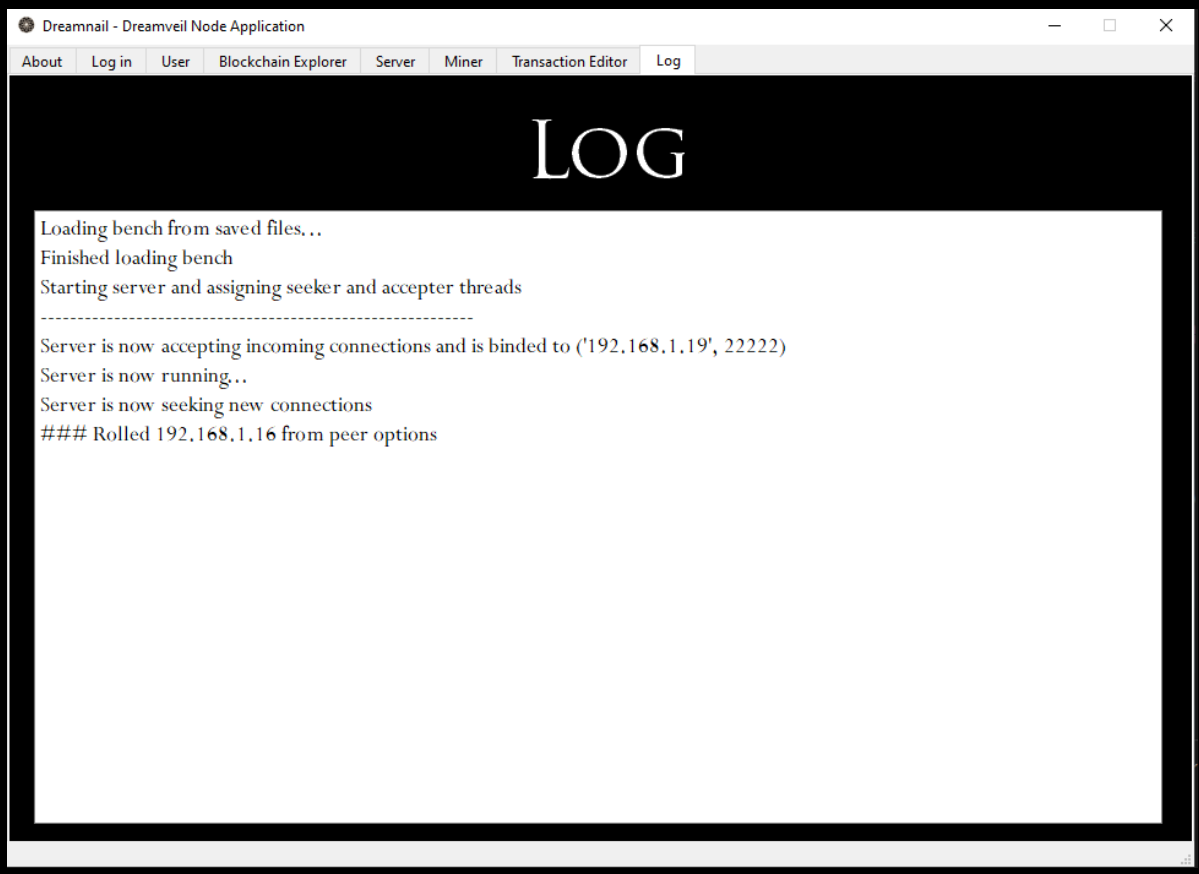
דף עורך ההעברות



בדף זה יכול המשתמש לערוך וליצור העברות שאותן האפליקציה תעביר הלאה לכל העמיתים המחוברים. ניתן להוסיף ולהסיר פלטים אל ההעברה ולשנות את הודעתה. בעת סיום עריכת ההעברה ניתן ללחוץ על הכפתור "צור ושלח ההעברה" על מנת להפיץ אותה לכל העמיתים המחוברים וגם בכדי להוסיף אותה אל בריכת ההעברות.

_המקורות להעברה המיוצרת יקבעו באופן אוטומטי על יד הtracklist בBlockchain ששומר את פעולות החשבון אשר בוצעו על ידי ארנק מסוים (ארנק המשתמש הוסף לשם בעת יצרת המשתמש). כמו כן האפליקציה תיצור פלט עודף באופן אוטומטי במידה וסכום המקורות גדול מסכום הפלטים.

דף הלוג



בדף זה יכול המשתמש לראות לוג של פעולות ותרחישים שקוראים בזמן ריצת האפליקציה.

exit_handler

מטרת הפעולה לבצע פעולות אחרונות לפני יציאת התוכנית. הפעולה תסגור את השרת ותשמור את קבצי האפליקציה: blockchain.json, peer_pool.json, transaction_pool.json. הפעולה גם תשמור את קובץ המשתמש.

```
def exit_handler(self):
    if not self.exited:
        self.close_server()
        dreambench.write_blockchain_file(self.blockchain)
        dreambench.write_peer_pool_file(self.peer_pool)
        dreambench.write_transaction_pool_file(self.transaction_pool)
        if self.user_data != dreambench.USER_DATA_TEMPLATE and
        self.user_passphrase is not None:
        dreambench.write_user_file(self.user_passphrase, self.user_data)
        self.log("Application Exit")
        self.exited = True
```

הצפנות וקירפטוגרפיה

פונקציית הגיבוב הקריפטוגרפית

פונקציית גיבוב קריפטוגרפית (נקראת גם פונקציית Hash או פונקציית ערבול) היא פונקציה חד-כיוונית שממירה קלט בכל אורך לפלט באורך קבוע. פונקציית גיבוב קריפטוגרפית בנויה כך שכל שינוי קטן בפלט יגרום לשינוי משמעותי בפלט. בפונקציית גיבוב קריפטוגרפית קשה או אפילו בלתי אפשר לשחזר את הקלט באמצעות הפלט. הפלט של פונקציית גיבוב קריפטוגרפית הוא אקראי ביחס לפלטים אחרים של הפונקציה.

פונקציית הגיבוב הקריפטוגרפית שהשתמשתי בה בפרויקט היא SHA256 בשל הבטיחות שלה והיותה הסטנדרט המומלץ לפונקציית גיבוב קריפטוגרפית.

בפרויקט השתמשתי בSHA256 הממומשת על ידי הספרייה pycryptodome.

השתמשתי בפונקציית הגיבוב הקריפטוגרפית בBlock על מנת להשוות בקלות בין תכני הבלוקים. השתמשתי בפונקציית הגיבוב הקריפטוגרפית לשם החתימה הדיגיטלית של Transaction.

בBlock אני השתמשתי בNonce על מנת להגריל פלט מפונקציית הHash. זאת מנקודת ההנחה שהפלטים הם אקראיים. השתמשתי בפונקציית הגיבוב גם בקוד אימות המסרים בפרויקט.

מערכת הצפנת מפתח ציבורי

מערכת הצפנת מפתח ציבורי היא מערכת הצפנה המורכבת מצמד מפתחות שבו האחד סודי והשני ציבורי. מערכת הצפנה זו נקראת גם אסימטרית.

בפרויקט השתמשתי בRSA הממומש על ידי הספרייה pycryptodome.

השתמשתי בRSA ליצירת הארנקים אשר הם מבוססים על RSA. מפתח הארנק המפתח הפרטי וכתובת ארנק היא קידוד בסיס 64 של האקספורט של המפתח הציבורי.

חתימה דיגיטלית

חתימה דיגיטלית היא שיטה קריפטוגרפית שמטרתה לאמת את המקוריות של הודעה או מסמך. בליבה, חתימה דיגיטלית היא חתימה באמצעות מפתח פרטי מצמד מפתחות ציבורי-פרטי על גיבוב ההודעה.

בפרויקט שלי השתמשתי בPKCS#1 V1.5 עקב הפשטות הרבה שלה והיותה בטוחה ואמינה. חשוב לציין שאופן הפעולה של PKCS#1 V1.5 מוסיף גם padding scheme משלו לחתימה הדיגיטלית.

בפרויקט השתמשתי בPKCS#1 V1.5 הממומשת על ידי הספרייה pycryptodome.

השתמשתי בחתימה הדיגיטלית בTransaction על מנת להבטיח שרק בעל הארנק יוכל לחבר העברות אשר פועלות על הארנק כמקור.

פונקציית גזירת מפתח מבוססת סיסמה

פונקציית גזירת מפתח מבוססת סיסמה היא פונקציה שמחלה פונקציה פסודו-אקראית על סיסמה יחד עם ערך salt וחוזרת על התהליך מספר רב של פעמים. הפעולה מוסיפה קושי מחשובי בחישוב המפתח שמטרתו להגן ממתקפות brute-force.

פונקציית גזירת המפתח מבוססת הסיסמה שבה השתמשתי בפרויקט היא PBKDF2 בשל היותה הסטנדרט לפונקציית גזירת מפתח מבוססת סיסמה.

פונקציית גזירת המפתח מבוססת הסיסמה שבה השתמשתי ממומשת על ידי הספרייה pycryptodome.

השתמשתי בPBKDF2 בפרויקט על מנת המרת הסיסמאות למפתחות שאיתם אוכל להצפין את קבצי המשתמש.

קוד אימות מסרים

קוד אימות מסרים הוא שם כולל לפונקציות עם מפתח סודי המשמשות לאימות מסרים. קוד אימות מסרים משמש ליצירת תג אימות באמצעות מפתח סודי להודעה שלאחר מכן באמצעות פונקציית verify התג והמפתח הסודי יהיה ניתן לאמת כי ההודעה לא שונתה שלא כדין.

קוד אימות המסרים שבו השתמשתי בפרויקט הוא HMAC-SHA256, שהוא קוד אימות מסרים מבוסס פונקציית הגיבוב SHA256.

קוד אימות המסרים שבו השתמשתי ממומש על ידי הספרייה pycryptodome.

צופן בלוקים סימטרי

צופן בלוקים סימטרי הוא צופן שמטרתו לספק חיסיון להודעה או מסמך. צופן בלוקים סימטרי מחלק את הקלט לבלוקים ומבצע טרנספורמציה בתהליך המשתמש במפתח ההצפנה ובערבול של בלוקי ההודעה אחד עם השני לאורך תהליך ההצפנה.

צופן הבלוקים הסימטרי שהשתמשתי בפרויקט שלי הוא AES in CTR mode. הסיבה לכך היא מכיוון שAES הוא סטנדרט לצופן בלוקים סימטרי. הסיבה שהשתמשתי בו בCTR mode היא עקב הפשטות של סוג צופן זה, הבטיחות שלו והחוסר בpad.

צופן הAES שבו השתמשתי ממומש על ידי הספרייה pycryptodome.

השתמשתי בAES-CTR בפרויקט שלי בהצפנת קבצי המשתמש.

dreamshield

encrypt

הפעולה מקבלת str passphrase ו str message והפעולה מצפינה את message.
הפעולה מרחיבה את passphrase עם 2PBKDF במיליון איתרציות ל32 בייטים עם salt המיוצר אקראית. 16 הבייטים הראשונים ישמשו כמפתח לצופן הבלוקים ו16 הבייטים האחרונים ישמשו כמפתח הסודי לפונקציית הHMAC. ראשית אנו מצפינים את הפליינטקסט עם AES ב CTR mode עם nonce אקראי. לבסוף אנו מייצרים את הHMAC על salt || nonce || ct

והפונקציה מחזירה את mac digest || salt || nonce || ct

```
def encrypt(passphrase:str, pt:str):
"""
A passphrase based decryption function that uses PBKDF2 with an
encrypt-then-mac scheme using AES-CTR for confidentially with HMAC for integrity.

:str passphrase: The passphrase used for the encryption of the plaintext
:str pt: The message to be encrypted using the passphrase
:returns: The binary information: HMAC || salt || nonce || ct
"""
pt = pt.encode()
passphrase = passphrase.encode()

salt = get_random_bytes(16)
expansion = PBKDF2(passphrase, salt, 32, count=1000000,
hmac_hash_module=SHA256)
nonce = get_random_bytes(8)
ct = AES.new(key=expansion[:16:], mode=AES.MODE_CTR,
nonce=nonce).encrypt(pt)

mac = HMAC.new(expansion[16::], salt + nonce + ct, digestmod=SHA256)
return mac.digest() + salt + nonce + ct
```

decrypt

הפעולה מקבלת passphrase str וct bytes והפעולה מפענחת את ciphertext. הפעולה קוראת את הMAC מה32 בייטים הראשונים של הpt. את הsalt מה16 בטים אחרי. הפעולה מרחיבה את הpasspharse כפי שנעשה בencrypt באמצעות הsalt. הפעולה קוראת את הnonce מתוך ה8 בייטים הבאים ואת הciphertext מהקובץ משאר הבייטים בקובץ. הפעולה מייצרת את הmac לפי הsalt, nonce והciphertext מהקובץ.

כעת הפעולה משווה בין הhash שחושב על ידי הmac לבין הmac שנקרא מ32 הבייטים הראשונים בקובץ. אם הם שווים, הקובץ פוענח בהצלחה אחרת תוכן הקובץ שונה או הסיסמה איננה נכונה. לבסוף הפעולה תפענח את תוכן הקובץ המוצפן על ידי חלקו הראשון של הרחבת הסיסמה באמצעות AES בCTR mode ותחזיר את הplaintext.

```
def decrypt(passphrase:str, ct:bytes):
"""
A passphrase based decryption function that uses PBKDF2 with an
encrypt-then-mac scheme using AES-CTR for confidentially with HMAC for integrity.

:str passphrase: The passphrase used for decryption
:bytes ct: The ciphertext to be decrypted
:returns: The decrypted plaintext as str
"""
passphrase = passphrase.encode()
proposed_mac = ct[:32:]
salt = ct[32:48]
expansion = PBKDF2(passphrase, salt, 32, count=1000000,
hmac_hash_module=SHA256)
nonce = ct[48:56]
encrypted_pt = ct[56::]
mac = HMAC.new(expansion[16::], salt + nonce + encrypted_pt,
digestmod=SHA256)
if not secrets.compare_digest(mac.digest(), proposed_mac):
raise ValueError("Incorrect passphrase or invalid message")

pt = AES.new(key=expansion[:16:], mode=AES.MODE_CTR,
nonce=nonce).decrypt(encrypted_pt)
return pt.decode()
```

מדריך למשתמש

הערה

יש לשים לב שעקב המאפיינים של החיבור עמית לעמית שבו הפרויקט משתמש, על מנת להיות חלק ברשת, עליך להכיר את כתובת הIP של לפחות מחשב אחד המשתמש בה. במידה ופרויקט זה היה משוחרר תחת אתר מפיץ מסודר, הקובץ peer_pool.json כבר היה מאותחל עם מספר כתובות IP למחשבים פעילים ואמינים ברשת, אך עקב העובדה שזהו פרויקט בית ספרי, לא קיימים מחשבים אלו ולכן, תאלץ אתה המשתמש לאתחל את הקובץ עם לפחות כתובת IP יחידה של מחשב המשתמש באפליקציה. אופן הכתיבה שיש לעשות זאת הוא {"IP": "UNKNOWN"} כאשר כתובת הIP תחליף את המקום בו רשום IP.

1. על המשתמש ראשית להוריד פייתון גרסה 3.10 או חדש יותר.
2. לאחר מכן יש להוריד את תיקיית הפרויקט.
3. לאחר הורדת התיקיה יש לגשת לקובץ בתוך התיקייה node; node.cfg.sample ולשנות את שמו לnode.cfg
4. יש לשנות את הערך בaddress לכתובת הIP של המחשב שלך ברשת שבה אתה מעוניין להפעיל את הפרויקט. אם זאת ברשת הלוקלית אז לIP של המחשב שלך ברשת הלוקלית (ניתן למצוא אותה באמצעות הפקודה ipconfig בcmd). אם זה ברשת הגלובלית יש לשנות את הכתובת לכתובת הציבורית של הראוטר שלך.
5. אם אינך מעוניין לעבוד ברשת הגלובלית דלג על שלב זה. כעת עליך להגדיר portforwarding בראוטר שלך. על מנת לעשות זאת רשום בדפדפן לבחירתך את כתובת הIP של הgateway שלך (ניתן למצוא אותה באמצעות הפקודה ipconfig בcmd). נווט להגדרות רשת מתקדמות, portforwarding, והגדר חוק חדש על פורטים 22222 אל כתובת הIP שלך ברשת המקומית.
6. הפעל את הvirtual environment באמצעות הפקודה activate בטרמינל הפתוח בתיקיית הפרויקט.
7. הרץ את הפקודה pip install -r requirements.txt על מנת להתקין את המודולים ההכרחיים.
8. הרץ את dreamnail.pyw ; כעת אתה חופשי להפעיל את הפרויקט כרצונך. תוכל להיעזר בחלק המדבר על האפליקציה בספר זה על מנת ללמוד את השימוש של העמודים השונים באפליקציה!

רשימת ספריות וקבצים

dreamvei.py – מודול זה נכתב על ידי והוא מגדיר את אופי הבלוקצ'יין

dreambench.py – מודול זה נכתב על ידי ומטרתו לשמור ולקרוא קבצים בקלות (הוא אינו מסוגל לפרש אותם, רק לקרוא ולכתוב)

dreamshield.py – מודול זה נכתב על ידי והוא מכיל את מימוש הפעולות encrypt ו decrypt.

dreamui.py - מודול זה מיוצר על ידי QT Designer והוא מגדיר את הGUI.

dreamui.ui – קובץ פרויקט הGUI של QT Designer.

data_structures.py – מודול זה נכתב על ידי והוא מכיל את המימוש של עץ הAVL

dreamnail.pyw – האפליקציה הראשית. מממשת את הevents של הGUI ואת מחלקת החיבור. הקובץ שיש להריץ על מנת להשתמש באפליקציה.

configparser – ספריה המסוגלת לקרוא את הפורמט cfg.

ipaddress – ספריה היודעת לעבוד ולפרש כתובות IP

os – ספריה המאפשרת לכתוב ולקרוא קבצים ולהתנהל מול מערכת ההפעלה באופן כללי.

sys – ספריה המאפשרת לבצע פעולות באפליקציה הקשורות למערכת ההפעלה, כמו למשל לסגור את התוכנית.

json – ספריה המאפשרת לקרוא ולכתוב בפורמט json

random – ספריה המאפשרת שימוש בהגרלות וכלים אקראיים

math – ספריה המספקת פעולות הקשורות במתמטיקה

timeit – ספריה המאפשרת לקבל את הזמן הנוכחי וכנספח לחשב זמן ריצה של פעולות

socket – פעולה המאפשרת פתיחת סוקט, נקודת קצה המאפשרת הזרמה של מידע ותקשורת ברשת

time – ספריה הנותנת פעולות הקשורות לזמן כמו sleep

atexit – ספריה המאשרת להגדיר פעולה כפעולת יציאה שתופעל לפני יציאת התוכנית.

pyperclip – ספריה המאפשרת להעתיק ולהדביק טקסט דרך קוד

decimal – ספריה המאפשרת עבודה עם מספרים עשרוניים או גדולים ללא איבוד בדיוק

pycryptodome – ספריה המכילה מספר רב של כלים ופרימיטיבים קריפטוגרפיים.

threading - ספריה המאפשר עבודה מרובת תרדים

PyQt6 – ספריה המאפשרת את הממשק הגרפי של QT בפייתון.

secrets – ספריה שנותנת מספר כלים קריפטוגרפים שונים.

base64 – ספריה המאפשרת קידוד ופיענוח בבסיס 64

node.cfg – קובץ ההגדרה של האפליקציה.

node.cfg.sample – קובץ טמפלייט

bench – תיקייה המכילה את קבצי האפליקציה.

venv – תיקיית הסביבה הווירטואלית

node – התיקייה המכילה את קבצי הקוד של הפרויקט ואפליקציית הפרויקט.

requirements.txt – קובץ המכיל את הספריות שיש להתקין על מנת לעשות שימוש בפרויקט

מקורות והשראות

Cryptography I של דן בונה - https://www.coursera.org/learn/crypto

3blue1brown "But how does bitcoin actually work?"

The Bitcoin Whitepaper - https://bitcoin.org/bitcoin.pdf

Technion - https://www.youtube.com/watch?v=YUHUeglX1aw

Sunny Classroom - https://www.youtube.com/user/sunnylearning

תודות

אני רוצה להודות לאסף על כך שהאמין בי, עזר לי למצוא את המוטיבציה וגם על כך שהקשיב לי במשך שעות לכל הקשקושים והשטויות שאמרתי לו בתכנון הפרויקט.

אני רוצה להודות לעמית על הלוגו הנפלא.

אני רוצה להודות לגל בראון המורה והמנחה שלי לאורך הפרויקט שליווה אותי בכל הדרך והיה שם בשבילי.

אני רוצה להודות לדן בונה על הקורס המדהים שלו ולSunny Classroom שמסוגל להסביר כל נושא קריפטוגרפי בעבר בהווה ובעתיד תוך חמש דקות בצורה מושלמת

אני רוצה להודות לגיל על כך שהוא בחור מגניב.

אני רוצה להודות אופק גנאור על ההשראה האומנותית ועל התמיכה הנפשית

אני רוצה להודות למחשב הנייד שלי שלא נשרף מכרייה של שעות

אני רוצה להודות לinabakumori על כל המוסיקה המדהימה שהכין שהתנגנה אצלי באוזניות ב12 שעות ליום בשבוע האחרון של הפרויקט ומנע ממני מלאבד את השפיות שלי.

ולבסוף אני רוצה להודות לtamaninja על היותו אגדה ושהפרויקט חלקית מוקדש בשבילו.