המתכון ליצירת מפתח ביטקוין פרטי

תוכן עניינים

רקע:

בכל ארנק ביטקוין שבבעלותכם מסתתר קוד בעל ערך מספרי שנקרא מפתח ביטקוין פרטי (רוב הארנקים הדיגיטליים מאגדים בתוכם כמה מפתחות פרטיים), את המפתח ניתן לייצג בפורמטים שונים. דוגמה לאותו המפתח בכמה מהפורמטים:

Decimal (מספר רגיל):	*72,759,466,100,064,397,073,952,777,052,424,474,334,519,735,946,222,029,294,952,053,344,302,920,927,294*
Heaxadecimal:	*A0DC65FFCA799873CBEA0AC274015B9526505DAAAED385155425F7337704883E*
Binary:	*10100000110111000110010111111111100101001111001100110000111001111001011111010100000101011000010011101000000000101011011100101010010011001010000010111011010101010101110110100111000010100010101010101000010010111110111001100110111011100000100100010000011111 0*

הערה: בצורתו הבסיסית, מפתח ביטקוין פרטי הוא בסה"כ מספר הנמצא בטווח המספרים 1 עד - 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936

השימוש במפתחות ביטקוין פרטיים למטרת ייבוא, ייצוא וגיבוי מארנק ביטקוין דיגיטלי בפורמטים המצוינים למעלה, יוצרים כמה בעיות:

- ככל שהמפתח הפרטי מכיל יותר תווים, טעיות הקלדה של משתמשים הם יותר נפוצות.
- בעת ייבוא מפתחות ביטקוין פרטיים לארנק ביטקוין דיגיטלי, טעות פשוטה בהקלדה יכולה לגרום למשתמש לייבא מפתח ביטקוין תקין שאינו שייך לו.
- ארנקי ביטקוין דיגיטליים לא יכולים לבנות מנגנון שיודע לזהות ולתקן שגיאות הקלדה נפוצות למשתמשים בעת הייבוא.

פורמט WIF (Wallet Import Format) הוא פורמט שהומצא במיוחד לארנקי הביטקוין ונועד למטרת ייבוא וייצוא של מפתחות פרטיים בצורה בטוחה למשתמשים. פורמט WIF מכיל בתוכו כמה הגנות מיוחדות שלא מאפשרות למשתמשים לייבא לארנקי הביטקוין השונים מפתחות פרטיים עם טעיות הקלדה, או לחליפין לייבא סתם מספרים רנדומליים. פורמט WIF אפילו מאפשר לארנקים להפעיל מנגנון של תיקון שגיאות הקלדה נפוצות בעת ייבוא המפתח.

בכדי להבין לעומק כיצד ניתן לייצר מפתח פרטי ולייבא אותו לכל ארנק ביטקוין, אדגים את סדר הפעולות ליצירת מפתח פרטי תקין בפורמט WIF שלב אחר שלב. לבסוף אסביר את ההבדל בין שני סוגי הפורמטים המקובלים ברשת הביטקוין *WIF uncompressed* ו- *WIF compressed*.

הנה דוגמה לאותו המפתח שהצגתי בהתחלה רק בפורמט WIF:

WIF compressed:	*L2cQMfbGpih4yTTTa3Dx4YHo4CLXqvJ5rKsggs9iswuXQYECC8aK*
WIF uncompressed:	*5K38ZKiJBMmsk9iLcaakHfMa6FoZpLKpmhyo9aZnjossPc49J7e*

חשוב לציין: בגלל שיצירת מפתח פרטי היא פעולה מתמטית, ניתן לייצר מפתח ביטקוין פרטי בפורמט WIF בעזרת דף ועט ללא שימוש במחשב. קחו בחשבון שיצירת מפתח פרטי תקין באופן ידני היא פעולה המצריכה כמה ימים של עבודה, לכן עדיף לתת למחשב הפרטי שלכם לבצע את הפעולה בצורה מדויקת ובחלקיקי שניה

שלב 1 – יצירת מספר בינארי

מפתח פרטי תקין בפורמט בינארי (בסיס 2) הוא בסה"כ מספר המכיל 256 ביט (מספר ארוך הבנוי מ- 256 תווים שונים של 0 או 1). בשביל להפוך את המפתח הפרטי מפורמט בינארי ל-WIF ולייבא אותו לארנק שלכם, נאלץ להעביר את המספר סדרה של פעולות. הנה דוגמה למספר בינארי המכיל 256 ביט שילווה אותנו לאורך כל התהליך:

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000100
```

חשוב לציין: ניתן להמיר טקסט או מספר רגיל לפורמט בינארי ולייצג אותו בעזרת קוד של 256 ביט. אם תנסו להמיר את המספר הבינארי שבדוגמה לפורמט דצימלי (מספר רגיל/ בסיס 10) התוצאה תהיה הספרה 4, מכאן שאנחנו מייצרים את המפתח ביטקוין הפרטי בעל ערך מספרי השווה ל-4.

אם נרצה לחשב את סה"כ כמות האופציות האפשריות של מספרים בינאריים באורך של 256 ביט היכולים לשמש כמפתח פרטי ברשת הביטקוין, פשוט נחשב 2 (2 = מספר האופציות שכל תו במספר הענק יכול לייצג, במקרה שלנו זה 0 ו- 1) בחזקת 256 (כמות התווים שיש במספר הענק).
התוצאה:

2^256=115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936
מאה וחמש עשרה אוקטיליון קווינדסיליון...

לצורך ההשוואה, מספר גרגירי החול המשוער בכל העולם הוא בסה"כ:

7,500,000,000,000,000,000
7.5 קווינטריליון

שלב 2 – תיחום המספר

הוספת "prefix" (פתיח קבוע) של המספר הבינארי 10000000, נועד בכדי לשמור על המספר באורך 256 ביט לצורך המשך התהליך.

```
1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0100
```

במידה ולא נוסיף prefix לתחילת המספר הבינארי, המחשב יאחסן את הקוד הבינארי ללא האפסים המובילים בכדי לחסוך במקום אחסון. אם נבצע את המשך החישובים על הנתון המכיל רק 3 ביט במקום 256 ביט + prefix, המספר עדין ייצג את הערך הדצימלי 4 אבל לא נקבל בסוף התהליך מפתח ביטקוין תקין בפורמט WIF (נקבל מפתח בעל אורך קצר מהמקובל שאינו מתחיל באחד מהתווים הבאים: 5,K,L).

שלב 3 – המרה הקוד הבינארי (בסיס 2) להקסדצימל (בסיס 16)

לצורך נוחות נבצע המרה של התוצאה מהשלב הקודם לפורמט הקסדצימל (בסיס 16) המקצר ויזואלית את המספר הבינארי אך עדין מייצג את אותו הערך. ראשית נחלק את המספר הבינארי מהשלב הקודם לחלקים של 4 ביט, עם החלקים הללו אנחנו יכולים לייצג את אותו מספר בינארי ארוך בעזרת 16 התווים הבאים: A,B,C,D,E,F,0,1,2,3,4,5,6,7,8,9.

טבלת המרה:

0000=0	0001=1	0010=2	0011=3	0100=4	0101=5	0110=6	0111=7
1000=8	1001=9	1010=A	1011=B	1100=C	1101=D	1110=E	1111=F

התוצאה של המרת הקוד הבינארי מהשלב הקודם לפורמט הקסדצימל:

80000000000000000000000000000000000000000000000000000000000000000004

במידה ונרצה ליצר מפתח ביטקוין פרטי בפורמט WIF compressed , נוסיף את הדגל (Bit Flag) "01" לסוף התוצאה ונמשיך לשלבים הבאים. נראה כך:

80000000000000000000000000000000000000000000000000000000000000000401

שלב 4 – יצירת Checksum

- הקדמה:

בשביל להבין מהו מנגנון ה-checksum הכי קל להיעזר בדוגמה. בכרטיס אשראי שברשותכם הספרה האחרונה של הכרטיס (סיפרה מספר 16) היא התוצאה של סידרת פעולות מתמטיות קבועת המתבצעת על 15 המספרים הראשונים של הכרטיס. מספר כרטיס אשראי שספרתו האחרונה איננה התוצאה של הפעולות המתמטיות הקבועות הללו, הופך את המספר כרטיס אשראי ללא תקין. בכל מערכת סליקת אשראי בעת ביצוע עסקה (עוד לפני שנוצר תקשורת עם מחשב חיצוני כלשהו) מתבצעת פעולת בדיקת

תקינות המספר לפי ה-checksum . במידה והמספר כרטיס אשראי אינו תקין, מערכת הסליקה חוסכת תקשורת מיותרת עם מסד נתונים רחוק ומפסיקה את תהליך החיוב. המנגנון הזה מהווה בעצם שכבת הגנה ראשונה לאנשים שמנסים לנחש מספרי כרטיסי אשראי או מבצעים טעיות הקלדה בעת ביצוע עסקה. טעות הקלדה של ספרה אחת מתוך ה-15 תביא לסבירות מאוד גבוה של פסילת מספר האשראי והמשך ביצוע חיוב. שיטה ה- checksum היא מאוד אפקטיבית בעיקר בעידן התשלומים באמצעות האינטרנט.

מפתחות ביטקוין פרטיים בפורמט WIF תקין מכילים בתוכם גם כן את מנגנון ה- checksum . יצירת checksum במפתחות ביטקוין מתבצעת באמצעות האלגוריתם SHA-256.

האלגוריתם SHA-256 (Secure Hash Algorithm 256) הומצא ע"י רשות הביון האמריקאית ה- NSA וכיום האלגוריתם נמצא בשימוש בעיקר בתחום אבטחת מידע והצפנה. האלגוריתם הוא בעצם סידרה של פעולות מתמטיות קבועות המאפשרות להצפין מספרים בינאריים (ליצור בעזרת חישובים מתמטיים קבועים, מספר בינארי אחד ממספר בינארי אחר). חשוב לציין שמספרים בינאריים יכולים לייצג ערכים שונים כמו טקס, מספר דצימלי, בסיס 16 וכו...
האלגוריתם שייך למשפחת האלגוריתמים המכונים "פונקציות כיוון אחד", זאת משום שלא קיימת פונקציה שיכולה לפענח את התוצאה המוצפנת בחזרה למספר המקורי.

כמה מהמאפיינים החשובים של האלגוריתם:

- אם נבצע מספר פעמים את האלגוריתם על אותו מספר בינארי, התוצאה המוצפנת תמיד תהיה זהה.
- אם נשנה אפילו מספר בינארי אחד במספר הבינארי עליו האלגוריתם מתבצע, זה עלול להשפיע על כל תו בתוצאה.
- התוצאה תמיד תהיה באורך של 256 ביט, גם אם המספר הבינארי עליו האלגוריתם התבצע הוא קטן או גדול מ-256 ביט.

לצורך ההדגמה:

את המילה "Bitcoin" ניתן גם לייצג בעזרת המספר הבינארי הבא:

01101110 01101001 01101111 01100011 01110100 01101001 01000010

אם נבצע על המספר הבינארי את האלגוריתם SHA-256 ונמיר את התוצאה הבינארית לבסיס הקסדצימל, נקבל את התוצאה המוצפנת (באורך 256 ביט) הבאה:

b4056df6691f8dc72e56302ddad345d65fead3ead9299609a826e2344eb63aa4

אם נשנה במילה "Bitcoin" את האות הראשונה ל- b קטנה, התוצאה המוצפנת (באורך 256 ביט) תהיה שונה לגמרי למרות ששינינו מספר בינארי אחד:

b88c087247aa2f07ee1c5956b8e1a9f4c7f892a70e324f1bb3d161e05ca107b6

באתר הבא ניתן להכניס ערך בפורמט HEX (הקסדצימל) ולבצע עליו את האלגוריתם SHA-256:
https://www.fileformat.info/tool/hash.htm

בשביל להבין איך האלגוריתם SHA256 בנוי ועובד, אתם מוזמנים לעיין במסמך הבא:
http://iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf

בסרטון הבא אפשר לראות איך פותרים חלק אחד מהאלגוריתם ללא השלב המקדים של פירוק המספר וללא השלב המסכם. השלב המוצג בסרטון לוקח כ-17 דקות לפתירה בעזרת דף ועט. חשוב לציין שאת השלב המוצג בסרטון צריך לבצע 63 פעמים נוספות כחלק מהאלגוריתם, לכן עדיף לתת למחשב לעשות את העבודה בשבילכם.
https://www.youtube.com/watch?v=y3dqhixzGVo

- <u>אופן הוספת ה-Checksum:</u>

1. ביצוע של האלגוריתם SHA-256 על התוצאה המוצגת בשלב מספר 3:

1E3565EAE674BF4244426DC0C271430D76622634EFBBA9FD2C10AEBC22A1C8FD

2. ביצוע של האלגוריתם SHA-256 פעם נוספת על התוצאה המוצפנת של האלגוריתם:

086EC77B0731DE235A0A4DD52243440ABA31DF32403A7C7410601E90505B0C91

3. ה-checksum הוא 8 התווים הראשונים של ההצפנה הכפולה (32 ביט ראשונים):

086EC77B

4. הוספת ה- CHECKSUM לסוף התוצאה של שלב מספר 3:

800000000000000000000000000000000000000000000000000000000000000004086EC77B

חשוב לציין: בעת ייבוא מפתח פרטי בכל ארנק ביטקוין, תוכנת הארנק ממירה את המפתח לפורמט הקסדצימל ובודקת האם ה- checksum (8 התווים האחרונים) אכן תקין ותואם למפתח עצמו. במידה וה-checksum אינו תואם לשאר המספר, הארנק יודע לזהות שנעשתה טעות בהקלדה או שהוקלדו מספרים ואותיות ללא כל משמעות. מספיק שינוי של ביט אחד במספר הענק שמוביל לפסילת בדיקת ה-checksum, כך נמנע מהמשתמש הפשוט שאינו מבין את המנגנון לשחזר מפתח פרטי שאינו שייך לו.

שלב 5 – המרה לבסיס 58

- הקדמה:

בשביל להבין מהו פורמט בסיס 58 אנחנו צריכים להבין מהו חשבון מודולרי.
אם נפעיל שעון סטופר דיגיטלי רגיל, נבחין שברגע שהמיקום המציג את השניות מגיע לספרה 59 הוא מתאפס ומתווספת הספרה 1 למיקום של הדקות. שעון הסטופר הוא הדוגמה הכי טובה להמחשה של מערכת הבנויה על בסיס 60, מערכת המכילה שני מיקומים שמסוגלים לייצג 60 מספרים (0-59) . שעון הסטופר מראה כיצד ניתן לייצג את אותה היחידה של זמן כמו למשל 95 שניות, בפורמט בסיס 60 הנראה כך:

1:35

בשביל להגיע לתוצאה אנחנו צריכים לבצע משוואה של חשבון מודולרי ומשוואה של חילוק:

95 Mod 60 = 35
int(95/60) = 1

במילים אחרות נבדוק כמה פעמים 60 נכנס ב- 95 בצורה מלאה ונחשב את השארית של החילוק. את השארית נציב במיקום הימני (המיקום של השניות), את תוצאת החילוק נציב במיקום השמאלי (המיקום של הדקות).

הערה: לכל אורך הדרך השתמשנו בכמה פורמטים הבנויים על בסיסים שונים. פורמט בינארי הוא פורמט בסיס 2, ניתן לייצג אתו כל ערך מספרים בעזרת תווים המכילים את הספרות 0 ו-1.
פורמט דצימלי (מספר רגיל) הוא פורמט בסיס 10, ניתן לייצג אתו ערך מספרי בעזרת הספרות 0-9.
פורמט הקסדצימל הוא פורמט בסיס 16, ניתן לייצג אתו כל מספר בעזרת הספרות 0-9 ובעזרת האותיות A-F המייצגות את המספרים בעלי ערך דו ספרתי.

- שיטות המרה לבסיס 58:

1. שיטת חישוב במספרים גדולים:

משום שרוב המחשבונים שלנו מוגבלים מבחינת חומרה לחישובים במספרים גבוהים, ניתן לבצע המרה בשיטה זאת רק באמצעות מחשבונים תוכנה מיוחדים או שפות תכנות התומכות בחישוב של מספרים גבוהים (שפת תכנות כמו פייתון או כל שפה אחרת שניתן להוסיף לה ספריה תומכת).

סדר הפעולות:

לוקחים את התוצאה של השלב הקודם והופכים את האותיות למספרים, נראה כך:

8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 8 6 14 12 7 7 11

מבצעים על כל מספר בנפרד הכפלה בתוצאה של 16 בחזקת המיקום של מספר, לאחר מכן מחברים את כל התוצאות ביחד, לדוגמה:

```
11*(16^0) + 7*(16^1) + 7*(16^2) +  12*(16^3) + 14*(16^4) + 6*(16^5) ...
```

התוצאה המתקבלת היא ההמרה מפורמט בסיס הקסדצימל (בסיס 16) לפורמט דצימלי (בסיס 10):

```
63,657,374,260,452,690,195,888,927,762,793,067,532,858,387,302,060,507,832,379,389,042,324,415,617,604,289,389,578,107
```

בשביל הנוחות נקרא למספר הדצימלי הענק באות "M". בכדי להבין איך להמיר את המספר הענק לבסיס 58, בואו נדמיין את דוגמת שעון הסטופר ונניח שבמקום לייצג 95 שניות בעזרת בסיס 60, אנחנו צריכים לייצג "M" שניות בעזרת בסיס 58. המשוואה הראשונה שנבצע היא חשבון מודולרי בכדי למצוא את שארית החילוק, לאחר מכן נבדוק כמה פעמים המספר "58" נכנס במספר "M" בצורה מלאה.

```
M Mod 58 = 41
int(M/58) = 1,097,540,935,525,046,382,687,740,133,841,259,785,049,282,539,690,698,410,903,092,914,522,834,752,027,660,161,889,277
```

אז מצאנו את המיקום הימני השווה ל- 41, אבל התוצאה של המיקום השני שבא אחריו עדין גדולה מ-58 ולא מתאימה לשעון שלנו. בשביל לפתור את הבעיה, נחזור על שתי המשוואות שוב רק שהפעם התוצאה של החילוק שווה ל-"M":

```
M = 1,097,540,935,525,046,382,687,740,133,841,259,785,049,282,539,690,698,410,903,092,914,522,834,752,027,660,161,889,277
M Mod 58 = 31
int(M/58) = 18923119578018041080823105755883789397401423098115489843156774388324737103925175204987
```

נחזור על הפעולה שוב ושוב, עד ש-

```
int(M/58) = 0
```

התוצאה הסופית בשלב זה היא מפתח פרטי בפורמט בסיס 58 המכיל בתוכו checksum:

```
4,16,47,16,33,39,26,5,4,26,32,57,15,0,22,16,2,11,25,52,5,2,43,47,53,12,7,50,4,41,47,3,45,13,10,2,43,13,50,49,37,10,3,9,12,7,31,41
```

2. <u>שיטת האלגוריתם של ברבה:</u>

בעזרת האלגוריתם של ברבה ניתן להגיע גם כן לתוצאה של בסיס 58 רק בעזרת מחשבונים רגילים. ברבה פיתח אלגוריתם המאפשר להמיר בצורה קלה פורמטים מכל בסיס שהוא, לפורמט של בסיס אחד מתחתיו בעזרת מטריצות. מכאן שניתן להמיר את התוצאה של שלב 5, מבסיס 16 לבסיס 2 ומבסיס 2 לבסיס ל-64 (המרות בין בסיסים כמו 2,4,8,16,32,64 אינם מצריכות חישובים מסובכים), ברגע שהמספר נמצא בפורמט בסיס 64 ניתן לבצע עליו את האלגוריתם של ברבה 6 פעמים ולהגיע לתוצאה של פורמט בסיס 58.

האלגוריתם של ברבה: https://blogdelbarba.wordpress.com/2015/11/19/number-base-conversion-from-64-to-58/

שלב 6 – פורמט WIF

כמו ברוב הפורמטים המכילים ערכים דו ספרתיים, ניתן לקבוע לכל מספר אות שונה או תו כלשהו שייצג אותו. לפי פורמט WIF שנוצר במיוחד לביטקוין, ניתן לייצג את התוצאה מהשלב הקודם בעזרת הטבלה הבאה:

Base 58	WIF	Base 58	WIF	Base 58	WIF	Base 58	WIF	Base 58	WIF
0	1	1	2	2	3	3	4	4	5
5	6	6	7	7	8	8	9	9	A
10	B	11	C	12	D	13	E	14	F
15	G	16	H	17	J	18	K	19	L
20	M	21	N	22	P	23	Q	24	R
25	S	26	T	27	U	28	V	29	W
30	X	31	Y	32	Z	33	a	34	b
35	c	36	d	37	e	38	f	39	g
40	h	41	i	42	j	43	k	44	m
45	n	46	o	47	p	48	q	49	r
50	s	51	t	52	u	53	v	54	w
55	x	56	y	57	z				

בכדי למנוע בלבול וטעיות הקלדה למשתמש בעת ייבוא, ייצוא וגיבוי המפתחות הפרטיים, הוחלט לא להשתמש בכמה מתווים הבאים: "I" (איי גדולה), "O" (אוו גדולה), "l" (אל קטנה) והספרה "0".
במידה ומשתמש הקליד בעת ייבוא לארנק דיגיטלי את המפתח הפרטי שלו והשתמש באחד מהתווים שאינם מקובלות בפורמט WIF, ארנקי תוכנה יכולים לבצע תיקון אוטומטי לאותיות ולשנות אותם לאות "o" (אוו קטנה) ולספרה "1".

לפניכם התוצאה של המרת המפתח מבסיס 58 לפורמט WIF uncompressed תקין המכיל checksum ואותו ניתן לייבא לכל ארנק ביטקוין.

5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreB4AD8Yi

במידה ובחרתם להוסיף את הדגל בסוף שלב מספר 3, תקבלו את התוצאה של מפתח פרטי בפורמט WIF compressed שנראה .

KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU75NBY2dKG

WIF uncompressed ו- WIF compressed.

כיום ברשת הביטקוין ניתן לייצר מאותו המפתח הפרטי המכיל 256 ביט, שתי כתובות שונות לקבלת ביטקוין. כתובת אחת נוצרת מהמפתח הפומבי המלא – "uncompressed public key" , הכתובת השנייה נוצרת מחלק מהמפתח הפומבי המלא ונקראת – "compressed public key". בשביל לידע את תוכנת הארנק איזה כתובת קבלת ביטקוין עליו ליצר למשתמש, אנחנו משתמשים במפתח הפרטי כסימון. מפתח פרטי בפורמט WIF uncompressed יידע את הארנק תוכנה לייצר כתובת ממפתח פומבי שאינו מקומפרס, בעוד מפתח פרטי בפורמט WIF compressed (המכיל את אותם 256 ביט המשמשים כמפתח) יידע את הארנק לייצר כתובת ממפתח פרטי מקומפרס.

חשוב לציין: כיום מומלץ להשתמש רק במפתחות פרטיים בפורמט WIF compressed משום שהכתובות ביטקוין הללו מהוות יתרון טכנולוגי.

הכוח שמאחורי המספרים הגדולים:

הכוח האמתי של ביטקוין בסופו של דבר טמון במספרים הגדולים. כהקבלה תדמיינו שארנקי הביטקוין היו מגירות פתוחות ללא מנעול שבהם מאחסנים כסף. בשביל לגנוב את הכסף מהמגירה כל מה שעליך לעשות הוא לפתוח את המגירה ולקחת את הכסף, אך מה קורה שהשידה מכילה 2^256 מגירות? האם יש לך את הכלים לפתוח כל כך הרבה מגירות? כמה זמן לוקח לפתוח את כולם?

מדענים טוענים שבשנת 2025 שטח האחסון העולמי הכולל (של כל המכשירים בעולם כולל סמארטפונים, מחשבים ביתיים, שרתים של חברות וארגונים כמו פייסבוק וגוגל וכו..) יגיע לנפח של כ-163 יחידות אחסון הנקראות zettabyte, אם נחשב את סה"כ מספר האופציות של המפתחות הפרטיים החוקיים שיכולים להתקבל ברשת הביטקוין, זאת בהנחה שכל מפתח מכיל 296 ביטים של מידע, נצטרך נפח אחסון של כ-

zettabyte 4,284,307,301,780,699,230,672,126,445,321,452,590,570,989,432,628,700,869,459

כמו שאתם בטח מבינים לבד, אין מספיק חומר גלם בעולם בכדי לייצר שטח אחסון בנפח כזה עם הטכנולוגיה של היום, שלא נדבר על כוח המחשוב שצריך בכדי לייצר כמות כזאת של מפתחות פרטיים. בהנחה שקיים מחשב שמסוגל לייצר טריליון כתובות בשנייה, מספר השנים המשוער ליצירת כל המפתחות הפרטיים הוא:

3,764,568,028,158,688,209,515,806,576,697,354,474,006,124,657,512,762,823,796

רק ברגע שנפתח טכנולוגיות אחסון מתקדמת יותר וטכנולוגיות מחשבים חזקות יותר, נאלץ לשנות את הקוד של ביטקוין שיתאים למספרים אפילו יותר גדולים.

משום שמספר מפתחות הפרטיים שיכולים להיווצר בביטקוין הוא כל כך גדול, אין דאגה שארנקים המייצרים כתובת ביטקוין בצורה פסאדו-רנדומלית ייצרו את אותו המפתח פעמיים. בהנחה שהמספר הבינארי אותו מייצר הארנק הוא באמת רנדומלי, לפגוע במפתח פרטי השייך למשתמש קיים זה כמו לפגוע באטום אחד בגלקסיה.

החשיבות של פונקציות רנדומליות:

רוב התוכנות הבנויות בכדי ליצר למשתמשים מפתחות ביטקוין פרטיים, הם בסה"כ תוכנות שמייצרות מספר בינארי רנדומלי באורך 256 ביט שאותו הם ממירות לפורמט WIF תקין. למרות שלפעמים נראה לנו שמחשבים יכולים לעשות הכל, הפונקציה שבאמת מדענים התקשו לייצר במשך שנים היא רנדומליות. למחשבים אין דעה ואין רגשות, מחשבים רק יכולים לבצע סט של פעולות מתמטיות מאוד מסובכות שהתוצאה שלהם יכולה להביא למספר הנראה כמו מספר רנדומלי.

מי שיודע איך פונקציות רנדומליות עובדות בתוכנה שנועדה לייצר מפתחות ביטקוין למשתמשים, יכול בקלות לנסות לחכות את ההתנהגות של הפונקציה ולייצר מפתחות פרטיים שכבר הונפקו למשתמשים האחרים. במילים אחרות להשתלט על מפתחות פרטיים קיימים ולגנוב מהם את הביטקוין.

אחת הדרכים הכי גרועות לייצר מספר רנדומלי במחשב הוא בעזרת פונקציות רנדומליות פשוטות. פונקציות רנדומליות פשוטות לרוב מבוססות על שעה נוכחית ועל עוד כמה פרמטרים שנועדו לערבל את הפלט.

שיטה נוספת היא אלגוריתמים פסאדו-רנדומליים (הנקראים גם כמעט רנדומליים). יש אלגוריתמים שנעזרים לייצר מספרים בעזרת תזוזת עכבר של המשתמש, כאלה שמשתמשים בחומרה פנימית של המחשב ומדידת חשמל, יש כאלה שמשתמשים בחומרה חיצונית המודדת רמות של קרינה, יש אלגוריתמים מאוד מורכבים שמשתמשים בסיד.

אחד השיטות הכי בטוחות, פשוטות וזולות לייצור מפתח פרטי היא פשוט להטיל מטבע 256 פעם. על כל תוצאה של עץ יש לכתוב את הספרה 0 ועל תוצאה של פאלי יש לכתוב 1. בעזרת שיטה זו כל בנאדם יכול לייצר לעצמו מספר בינארי רנדומלי שיכול לשמש כמפתח ביטקוין שלא תלוי בשום פונקציה.