

סקריפטים והעברות מתקדמות

כמו שראינו בפרק הקודם, על מנת שההעברה תאושר הסקריפט המורץ צריך להיות מאושר.

ישנם מספר כללים עבור סקריפט תקין. מלבד זה שתהליך החישוב צריך להמשיך ללא שגיאות, אסור לכתוב במהלך הסקריפט את הפעולה `op_return` העוצרת את התהליך באמצע. סקריפט כזה נכשל באופן אוטומטי על מנת למנוע זיופים ועצירה של סקריפטים באמצע.

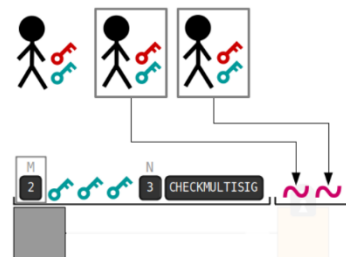
ראינו שניתן לכתוב סקריפט כך שנצטרך לאשר את הסקריפט ע"י 2 חתימות אך מה אם נרצה להוציא לפועל חשבון של שותפים בו לכל שותף יש אפשרות לבצע העברות? או מה אם נרצה חשבון (כתובת וסקריפט) בו יאושרו העברות עם 3 חתימות מתוך 7 או העברות מסובכות יותר?

בתור פתרון התחלתי, ישנה פעולת סקריפט `op_CHECKMULTISIG` כך שיכולה להוציא לפועל אישור של m חתימות קיימות מתוך n חתימות. מכאן, שפעולת `op_CHECKMULTISIG` יכולה להוציא לפועל אישור של חתימה אחת מתוך אחת עד n חתימות מתוך n וכל המקריים החלקיים שבאמצע.

כתיבת הסקריפט `op_CHECKMULTISIG` הוא באמצעות הסינטקס הבא:

```
<Sig 1><Sig 2> ... <Sig m> m <PK 1><PK 2> ...<PK n> n OP_CHECKMULTISIG
```

כיוון שקיים באג בפעולה הזאת (ונראה בהמשך שאין הרבה צורך בשימוש בה) נדרש להכניס עוד ספרה בתחילת הסקריפט כיוון שהפעולה מוציאה $m+1$ איברים לאחר המספר m . במקום לתקן את הבאג, נכנס לתוקף הסינטקס המתוקן:




```
0 <Sig 1><Sig 2> ... <Sig m> m <PK 1><PK 2> ...<PK n> n OP_CHECKMULTISIG
```

כך שנוסף 0 לפני הסקריפט. ספרה זו אינה משנה ואפשר להחליף אותה בכל ספרה אחרת כדי שהסקריפט יאושר אך לא ניתן להוריד אותה. נהוג לכתוב את הספרה 0.

How does P2MS work?


Multisig scripts are pretty straightforward to create. For the locking script:

1. Include an opcode **M** to indicate how many signatures are required.
2. Include the **public keys**.
3. Include another opcode **N** to indicate how many public keys there are.
4. Put the **CHECKMULTISIG** opcode at the end.

 scriptPubKey	<pre>OP_1 04cc71eb30d653c0c3163990c47b976f3fb3f3ccccdbdb169a1dfef58bbfbaff7d8a473e7e2e6d317b87baf8bd e97e3cf8f065dec022b51d11fcdd0d348ac4 0461cbdcc5409fb4b4d42b51d33381354d80e550078cb532a34bfa2fcfdeb7d76519a6cc62770f5b0e4ef8551946d8 a540911abe3e7854a26f39f58b25c15342af OP_2 OP_CHECKMULTISIG</pre>	P2MS
--	---	------

hex | opcodes inline | stack

To unlock a P2MS script, you just need to provide the required number of **signatures**. In this case **M** is one:

 scriptSig	<pre>OP_0 304402203f16c6f40162ab686621ef3000b04e75418a0c0cb2d8aebac894ae360ac1e780220ddc15ecd3c3507ac48 e1681a33eb60996631bf6bf5bc0a0682c4db743ce7ca2b01</pre>
---	--

hex | opcodes inline | stack

פתרון op_CHECKMULTISIG בא לפתור את בעיית כתיבת סקריפט שונה עבור כל העברה בנפרד. במידה ונרצה לשנות את החתימות בכל העברה, למשל במקרה של סוכן שמשנה את חתימת הלקוח בכל פעם, או סקריפטים מסובכים יותר, נתקל ב2 בעיות עיקריות:

סקריפט מסובך קשה לכתובה, להבנה ולשימוש ולכן מסרביל מאוד את כתיבת ההעברה. במיוחד במקרה שהסקריפט הוא מסובך אך בעל תבנית קבועה.

סקריפט מסובך וארוך מכביד מאוד את גודל ההעברה ובכך מחייב את בעלי ההעברה להעלות את העמלה על מנת שההעברה תאושר, דבר שמביא לכך שלא תמיד יהיה שווה לבעלי הביטקוין לבצע עסקאות מסורבלות כאלה.

לשם כך הומצא ב2012 כלי רב עצמה על מנת להוציא לפועל סקריפטים מסובכים כך שישמר בתבניתם המקורית ללא צורך ליצירה מחדש אך גם לא יתפסו הרבה מקום על ההעברה ולכן לא יגדילו את גודל ההעברה ואת הצורך בעמלה גבוהה. הה P2SH, Pay-to-Script-Hash, הוא קידוד באמצעות SHA1 (20 bytes) של סקריפט ההעברה ועל הקידוד הזה לבצע את HASH160 – כאמור, קידוד כפול של SHA256 ושל RIPEMD160.

המעניין בקידוד של סקריפט מסובך כזה, שלאחר שמבצעים עליו אלגוריתם Base58Check הסקריפט נהיה ככתובת ביטקוין לכל דבר. דבר זה מקל על שליחה של העברה ל"סקריפט מסובך" כזה או לתאגיד כלשהו בעל סקריפט מסובך ומקל על אישור הסקריפט כשהכסף נשלח מתוך כתובת כזו. (כתובת ביטקוין שמקורה בסקריפט מתחילה ב3, להבדיל מכתובת רגילה שמתחילה ב1).

פעולת RETURN

ישנו שימוש ברשת הביטקוין על מנת לאחסן חוזים חכמים ומסמכים. שליחה של מידע בתוך רשת הביטקוין תשמר עם תאריך השליחה ולכן גם תהווה תיעוד טוב לקיומו של מסמך ואפשר להוכיח באמצעות הרשת תאריך פרסום של חוזה או מאמר כלשהו.

סקריפט בעל הפעולה RETURN אמנם נכשל, אך שומר את ההעברה בתוך בסיס הנתונים של "כסף" שלא בוזבז. מסמך שקודד ברשת אך אינו אושר בתור העברה (כי הסקריפט נכשל) מאפשר למפרסם המסמך לגשת אליו בכל עת ללא תשלום עמלות (כיוון שהסקריפט לא אושר).

שימוש זה ברשת הביטקוין דומה לשימוש במטבע Eterium לצורך שמירת מסמכים וחוזים.

מנעולי זמן

ישנם 2 פעולות עיקריות המממשות מנעולי זמן ומאפשרות להכניס מגבלה של זמן כלשהו להוצאת עסקה לפועל. OP_CHECKLOCKTIMEVERIFY -i OP_CHECKSEQUENCEVERIFY.

הגבלת זמן נכנסת בתחילת הסקריפט ומספקת הגבלת זמן עד שהעסקה תוכל להיות "מבוזבזת" ע"י המקבל. אם אליס שולחת לבוב העברה עם מנעול זמן של 3 חודשים אז:

בוב אינו יכול להעביר את ההעברה הזאת הלאה ב3 חודשים הקרובים.

אליס יכולה ליצור העברה חדשה באותו כסף ללא הגבלת זמן (ובכך בעצם להפוך את ההעברה הראשונה לInvalid)

בוב אינו יכול למנוע מאליס לבצע את זה במסגרת מנעולי הזמן שהוצעו לו.

בשל ההגבלה הזו – שאין בטחון בהעברה עם מנעול זמן, ובכל רגע שולח הכסף יכול לפסול את ההעברה ע"י "בזבז כפול" הומצא (BIP-65) מנעול זמן חדש הנקרא:

"Check Lock Time Verify" – (CLTV).

כיוון שמנגנון מנעול הזמן פועל ברמת ההעברה, אפשר "לדרוס" את ההעברה ע"י העברה אחרת. מנגנון CLTV פועל ברמת output ולכן אינו מאפשר פסילה ע"י העברה אחרת. CLTV אינו מחליף

את מנגנון מנעול הזמן (nLocktime) אלא מוודא שהמטבע אינו יועבר הלאה בזמן שנקבע במנעול הזמן.

כדי להוציא לפועל את מנעול הזמן (של 3 חודשים למשל) בסקריפט נכתוב בצורה הבאה:

```
<now + 3 month> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <PK 1> EQUALVERIFY CHECKSIG
```

כך שהסקריפט ישוחרר 3 חודשים מרגע שליחת ההעברה.

הסקריפט בודק אם הזמן שהשולח שם בהעברה הוא \leq הזמן שהוגדר כמנעול זמן (nLocktime) – כלומר אם העסקה יצאה לפועל אחרי זמן הטיימר שהוגדר בתחילת הסקריפט. אחרת, עסקה זו מסומנת כ Invalid. המקרים בהם CHECKLOCKTIMEVERIFY נכשל הם:

1. the stack is empty; or
2. the top item on the stack is less than 0; or
3. the lock-time type (height versus timestamp) of the top stack item and the nLocktime field are not the same; or
4. the top stack item is greater than the transaction's nLocktime field; or
5. the nSequence field of the input is 0xffffffff.

אליס שהעבירה את הכסף לבוב תחת מגבלה של זמן CLTV אינה יכולה להוציא העברה נוספת שתפסול את הראשונה כיוון שהעברה מוצפנת עם המפתח של בוב, ובוב לא יכול להעביר את הכסף הלאה כיוון שלא עבר זמן מנעול הזמן ולכן העסקה עוד לא יצאה לפועל (ולא אושרה).

ברגע ש"מכשול" הזמן עובר והסקריפט ממשיך, נדרש להוציא את פרמטר הזמן מהמחסנית ולכן ישנה פקודה DROP שמוציאה את האיבר העליון במחסנית. לכן השימוש בפעולה CHECKLOCKTIMEVERIFY גורר שימוש בDROP מיד אח"כ.

מנעולי זמן כמו CLTV ו nLocktime הם מנעולים דטרמיניסטיים שתלויים בזמן שעון. במקרה ונרצה לבצע מנעולים התולים את ההעברה בהעברה אחרת או בתנאי אחר כלשהו, נצטרך להשתמש בפעולה חדשה – CHECKSEQUENCEVERIFY – (CSV).

הפעולה משווה בין מספרים סידוריים המוצמדים להעברה כלשהי ומאפשרת לבצע החלפה של העברה אחת באחרת ע"י ה nSequence field.

במקור נועד השדה הזה כדי לאפשר עריכה של העברות שעדיין לא אושרו ע"י החלפתן בהעברה אחרת בעלת מספר Sequence גבוה יותר, גרסה סופית תקבל את המספר הסידורי 2^{32} ולא תוכל להיות מוחלפת יותר. בהגבלות זמן כלשהן, שדה זה חייב להיות קטן מ 2^{32} על מנת להוציא לפועל את ההגבלה ולכן בדר"כ שדה זה הוא $1 - 2^{32}$. אפשרות זו של עריכה לא מומשה והשימוש בשדה הזה הוא להחיל מגבלות על העברות.

<https://bitcoin.stackexchange.com/questions/2025/what-is-txins-sequence>

המקרים בהם CHECKSEQUENCEVERIFY נכשל הם:

- the stack is empty; or
- the top item on the stack is less than 0; or
- the top item on the stack has the disable flag (1 < 31) unset; and
 - the transaction version is less than 2; or
 - the transaction input sequence number disable flag (1 < 31) is set; or
 - the relative lock-time type is not the same; or
 - the top stack item is greater than the transaction input sequence (when masked according to the BIP68);

השימוש בשדה זה, אם כן, נועד על מנת שהסקריפט ישווה בין הזמן שכתוב בתוך הסקריפט לזמן שמצוין בשדה על מנת למנוע מההעברה להיות מאושרת לפני הזמן, ועל מנת למנוע מלהוציא את הכסף באופן אחר. (בדומה לCHECKLOCKTIMEVERIFY עם השדה nLocktime).