Information Processing Letters 140 (2018) 37–41

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Bitcoin private key locked transactions

Sergi Delgado-Segura [*], Cristina Pérez-Solà, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas

Department of Information Engineering and Communications, Universitat Autònoma de Barcelona, Spain

ARTICLE INFO

Article history:
Received 21 April 2017
Received in revised form 11 May 2018
Accepted 19 August 2018
Available online 24 August 2018
Communicated by S. Faust

Keywords:
Bitcoin
Cryptocurrencies
Scripting
Smart contracts
Cryptography

ABSTRACT

Bitcoin smart contracts allow the development of new protocols on top of Bitcoin itself. This usually involves the definition of complex scripts, far beyond the requirement of a single signature. In this paper we introduce the concept of private key locked transactions, a novel type of transactions that allows the atomic verification of a given private key (belonging to an asymmetric key pair) during the payment execution.

1. Introduction

Bitcoin transactions require the execution of a contract, that defines the terms under which the transaction can actually be redeemed. The contract is defined by the scripts provided by the sender (locking script) and the receiver or redeemer of the transaction (unlocking script).

Different locking conditions can be defined. For instance, hash locked transactions require the redeemer to prove he knows the preimage of a given hash in order to redeem the output. So that to create this construction, the transaction output includes the value of the hash h, such that $h = H(m)$ for some m, and a script that specifically asks for the preimage of h (where H is a cryptographic hash function). Then, the output can only be spent by providing a preimage m, whose hash is exactly h.

Hash locks have recently gained popularity in the Bitcoin system because they are used, in combination with time locks, to create bidirectional micropayment channels [1,2]. This kind of channels allows to securely transfer Bitcoins between parties minimizing the amount of transactions stored in the blockchain.

The specific case of a hash locked transaction where the preimage m is a symmetric key is of special interest as a building block for protocols that operate on top of Bitcoin. For example, a possible idea behind this kind of schemes is to encrypt data using a symmetric key cryptosystem, release the encrypted data, and then create a hash locked output where the unlocking value corresponds to the symmetric key used to encrypt the data. The amount of bitcoins deposited into that output corresponds to the price that is going to be payed for the ability to decrypt (and thus obtain) the data. In order to collect the bitcoins deposited into the output, the symmetric key must be disclosed, thus allowing the decryption of the data [3].

Extending this idea, in this paper, we introduce and propose private key locked transactions where the locked value is precisely a private key from an asymmetric key pair. Note that standard scripts requiring a signature are already useful to prove the possession of a private key (because the private key is needed to create the signature). However, the scheme we propose requires not only to prove that the private key is known in order to redeem an output but also to reveal that private key.

[*] Corresponding author.
E-mail address: sdelgado@deic.uab.cat (S. Delgado-Segura).

https://doi.org/10.1016/j.ipl.2018.08.004
0020-0190/© 2018 Elsevier B.V. All rights reserved.

2. Private key locked transactions

In order to build a transaction output that requires the disclosure of a specific private key to be redeemed, we propose to use a well known vulnerability in the ECDSA signature scheme. In our case, such vulnerability becomes a property of ECDSA, since it allows us to nicely implement the key disclosure upon payment.

2.1. ECDSA vulnerability

ECDSA (Elliptic Curve Digital Signature Algorithm) is the cryptographic algorithm used by Bitcoin to create and validate digital signatures. ECDSA has a set of system parameters: an elliptic curve field and equation $\mathbb{C}$, a generator G of the elliptic curve $\mathbb{C}$, and a prime q which corresponds to the order of G. The values for these parameters are defined to be `secp256k1` [4] for Bitcoin.

Let use denote by $*$ the operation of multiplying an elliptic curve point by a scalar. Given a specific configuration of the parameters and a private key d, the ECDSA signature algorithm over the message m is defined as follows:

1. Randomly choose an integer k in $[1, q-1]$
2. $(x, y) = k * G$
3. $r = x \mod q$
4. $s = k^{-1}(m + rd) \mod q$
5. Output[1]: $sig(m) = (r, s)$

The ECDSA signature scheme is therefore probabilistic, that is, there exist many different valid signatures made with the same private key for the same message. The selection of a specific signature from the set of valid ones is determined by the election of the integer k.

There exists a well known ECDSA signature vulnerability (also present in the non-elliptic curve signature scheme of ElGamal and its popular variant, DSA [5]) by which an attacker that observes two signatures of different messages made with the same private key is able to extract the private key if the signer reuses the same k. Therefore, the selection of k is critical to the security of the system.

Indeed, given two signatures that have been created using the same k and the same private key, $sig_1(m_1) = (r, s_1)$ and $sig_2(m_2) = (r, s_2)$ with $m_1 \neq m_2$, an attacker that obtains m_1, sig_1, m_2, sig_2 may derive the private key d:

1. Recall that, by the definition of the signature scheme:
$$s_1 = k^{-1}(m_1 + rd) \mod q \Rightarrow ks_1 = m_1 + rd \mod q$$
$$s_2 = k^{-1}(m_2 + rd) \mod q \Rightarrow ks_2 = m_2 + rd \mod q$$
Note that, since r is deterministically generated from k and the fixed parameters of the scheme, the r of both signatures will be the same.
2. The attacker learns k by computing $k = \frac{m_2 - m_1}{s_2 - s_1}$
3. The attacker learns the private key d by computing $d = \frac{s_1 k - m_1}{r}$ or $d = \frac{s_2 k - m_2}{r}$

[1] A new integer k is chosen and the procedure is repeated if either s or r are 0.

Moreover, the leakage of a private key can also be produced in situations where similar k values are generated [6,7].

Taking advantage of such vulnerability to disclose a private key in Bitcoin has been previously used for timestamping in data commitment schemes [8].

Some Bitcoin wallets adopted deterministic ECDSA after this vulnerability was found to affect some Bitcoin transactions [9–11].

2.2. Private key disclose mechanism

Our proposed scheme makes use of the aforementioned ECDSA vulnerability to perform targeted private key disclosure within Bitcoin. The private key disclosure mechanism we propose allows to construct transaction outputs that need to reveal a private key in order to be redeemed, in such a way that we ensure the revealed private key is the counterpart of a certain public key.

Let $\{PK, SK\}$ be an ECDSA key pair belonging to Bob (with $Addr(PK)$ the Bitcoin address associated to it) and sig_{prev} an existing signature made with SK. Alice (that is interested in acquiring Bob's private key) needs to know the value of the previous signature sig_{prev}, in order to be able to request, afterwards, a second signature made with the same k. The previous signature may appear in the blockchain as the input script of an existing transaction. For instance, if some amount of bitcoins were sent to $Addr(PK)$ with a standard pay-to-pubkey-hash script output and Bob has already transferred those bitcoins to another output by showing a valid signature made with SK, sig_{prev} will be publicy available in the Bitcoin blockchain. Therefore, any observer will know this value and the signed message m will correspond to a transaction hash.

Once an existing previous signature sig_{prev} is known by Alice, she creates a transaction with an output that requires a second signature sig to be spent. However, instead of using the classical pay-to-pubkey-hash script, she uses a special script that forces Bob (the redeemer) not only to prove he has the private key SK associated to the given address $Addr(PK)$ by creating a valid signature, but also to deliver a signature that has exactly the same k value that was used when creating sig_{prev}. The output may also have a time lock that allows Alice to get back her bitcoins if Bob chooses not to reveal the private key.

Doing so accomplishes two purposes: on the one hand, Bob proves he knows the private key associated to the public key by generating a signature that correctly validates with that public key; on the other hand, Bob is implicitly revealing the private key associated to the same public key. Note that Bob does not directly provide the private key, but provides information from which the private key can be derived.

Moreover, the operation is atomic, in the sense that Bob gets Alice's bitcoins (the amount deposited into the output) only when Alice gets Bob's private key (derived from the two signatures by exploiting the reusage of k vulnerability).

Furthermore, unlike when revealing symmetric keys with hash locks, the private key disclosure mechanism al-