International Journal of Information Security manuscript No.
(will be inserted by the editor)

Double-Spending Prevention for Bitcoin Zero-Confirmation Transactions

Cristina Pérez-Solà ✉ · Sergi Delgado-Segura · Guillermo Navarro-Arribas · Jordi Herrera-Joancomartí

Received: date / Accepted: date

Abstract Zero-confirmation transactions, i.e., transactions that have been broadcast but are still pending to be included in the blockchain, have gained attention in order to enable fast payments in Bitcoin, shortening the time for performing payments. Fast payments are desirable in certain scenarios, for instance, when buying in vending machines, fast food restaurants, or withdrawing from an ATM. Despite being quickly propagated through the network, zero-confirmation transactions are not protected against double-spending attacks, since the double spending protection Bitcoin offers relies on the blockchain and, by definition, such transactions are not yet included in it. In this paper, we propose a double-spending prevention mechanism for Bitcoin zero-confirmation transactions. Our proposal is based on exploiting the flexibility of the Bitcoin scripting language together with a well-known vulnerability of the ECDSA signature scheme to discourage attackers from performing such an attack.

Keywords double-spending · Bitcoin · cryptocurrency · blockchain · ECDSA.

Mathematics Subject Classification (2010) 68M14

1 Introduction

Double spending, or spending a currency token more than once, is the main security issue that digital currencies have to deal with. Unlike physical money, where the physical token is hard to copy, and once it has been spent it passes effectively to the recipients' hands, digital currency tokens can be easily copied and double spent if security mechanisms are not correctly applied.

Bitcoin deals with this double spending problem by building an append-only ledger, the blockchain, that is replicated in every single Bitcoin full node. The blockchain is made of blocks that are stacked on top of each other. Blocks are made of entries, which contain some source (inputs) and destination (outputs). Entries in the blockchain are called transactions, and they are used to transfer bitcoins between different users, typically identified by their Bitcoin addresses. Bitcoin transfer is performed by using an unspent output of a previous transaction (UTXO) as the input of a new one. Therefore, Bitcoin transactions consume previous outputs and generate new ones.

Transactions are broadcast over the Bitcoin P2P network aiming to reach every single node. Nodes will validate the correctness of the received transactions and include the correct ones in their mempool. Such validation includes, among others checks, that the transaction does not claim any spent funds. Finally, every valid transaction should be eventually included in a new block. Hence, double spending is prevented once a transaction is part of the blockchain, since it has been proven that no previous transaction spends from the same outputs, and future transactions will be prevented to do so.[1]

However, transactions are not automatically included in blocks. In the time between transaction broadcasting and its inclusion, transactions are known as zero-confirmation transactions, and they are just stored in

Department of Information Engineering and Communications, Universitat Autònoma de Barcelona,
CYBERCAT-Center for Cybersecurity Research of Catalonia
E-mail: {cperez, sdelgado, gnavarro, jherrera}@deic.uab.cat

[1] Although it can be argued that Bitcoin transactions are not final since blockchain forks may always occur, throughout this paper, to simplify the discussion, we assume that once a transaction appears in the blockchain it is final.

the mempool of the nodes that have received them. Therefore, during this time window, different transactions spending the same outputs can be spread through the Bitcoin P2P network.

Having received a transaction spending an unspent output, the default behaviour of a node will be to store the transaction in his local mempool and drop any other incoming transactions trying to spend from the same source. However, different nodes could receive different transactions spending from the same source, and double spending could be attempted. For instance, suppose two transactions (tx_1 and tx_2) that spend the same output from a previous transaction (tx_0) are created by an attacker A. tx_1 is used to pay for some goods to B, while tx_2 is used to return the funds to the attacker. In this scenario, if A can make B believe that tx_1 is the only transaction spending from tx_0's output, but tx_2 finally becomes included in a block, the attack is successful. Figure 1 depicts the aforementioned example.

Notice that only one of the double-spending transactions will be included in a block due to the double spending protection that Bitcoin achieves by design. However, in case tx_2 is included in a block, tx_1 will be discarded and the double-spending attack will succeed.

In this paper, we propose a solution to mitigate the double spending problem for Bitcoin zero-confirmation transactions. In our proposal, any single observer who identifies a double spending attempt may take part and punish the attacker. Moreover, our solution discourages the attacker even to attempt the double-spending, because doing so makes him risk losing an amount of bitcoins bigger than the double-spent amount. Our solution benefits fast-payment scenarios, like in-shop purchases or trading platforms, where the transfer bitcoin-product/service cannot wait until the transaction is confirmed in the blockchain. Finally, although this paper is focused in Bitcoin transactions for conciseness, a similar construction can be developed for other cryptocurrencies with zero-confirmation transactions and based on ECDSA signatures, for instance, Litecoin or Dogecoin.

The paper is structured as follows. First, Section 2 reviews the state of the art in double-spending protection for fast payment transactions within Bitcoin. Then, Section 3 introduces the most important concepts about Bitcoin transactions, needed to understand our contribution. Next, Section 4 explains our proposed mechanism for discouraging double spending attempts. Section 5 provides implementation details and Bitcoin scripts to deploy the proposed scheme. After that, Section 6 analyses the benefits each party obtains when applying the proposed protocol. Finally, Section 7 presents the conclusions and provides guidelines for further research.

2 State of the art

Double spending attacks on zero-confirmation transactions in Bitcoin were first analyzed by Karame et al. [1, 2]. The authors showed how, with some reasonable assumptions and without the need of special computation nor much network overhead, an attacker has a great probability of succeeding with a double spending attack. Moreover, the authors also showed how basic countermeasures such as waiting a few seconds before accepting the payment or adding observers that report back to the payee are not enough on their own to avoid these type of attacks. Additionally, the authors proposed to modify the Bitcoin protocol rules so that nodes forward double spending transactions instead of dropping them. By doing so all nodes may be notified of double spending attempts. However, such a mechanism eases denial of service attacks. Moreover, nodes receiving both transactions will not be able to distinguish between the original and the double spending one.

The approach of monitoring observers has been implemented by companies such as GAP600 [3] in order to provide risk evaluation for accepting zero-confirmation transactions. Nevertheless, the company does not provide details regarding the technique used to perform the analysis nor the evaluation.

Regarding mitigation of double spending attacks, Decker et al. [4] proposed some other countermeasures that can reduce the merchant's likelihood of being deceived by an attacker: requiring the merchant to be connected to a large random sample of nodes of the network and not accepting incoming connections. By applying such countermeasures the merchant ensures that the attacker cannot send the transaction directly to him nor should be able to identify his neighbours.

Other research studies have proven that this kind of attacks was actually possible, and not only was the attacker able to identify the merchant's neighbours, but also to make the merchant connect only to nodes controlled by the attacker [5–7].

Ultimately, none of the proposed countermeasures forces attackers to stop trying, nor punishes them by not doing so. However, if some penalty could be applied to anyone who tries to perform such an attack, and this penalty could be implemented by any node which detects it (instead of just dropping the transactions), attackers may be discouraged even to try it.

The idea of using penalties to discourage misbehaviour in protocols on top of cryptocurrencies is not

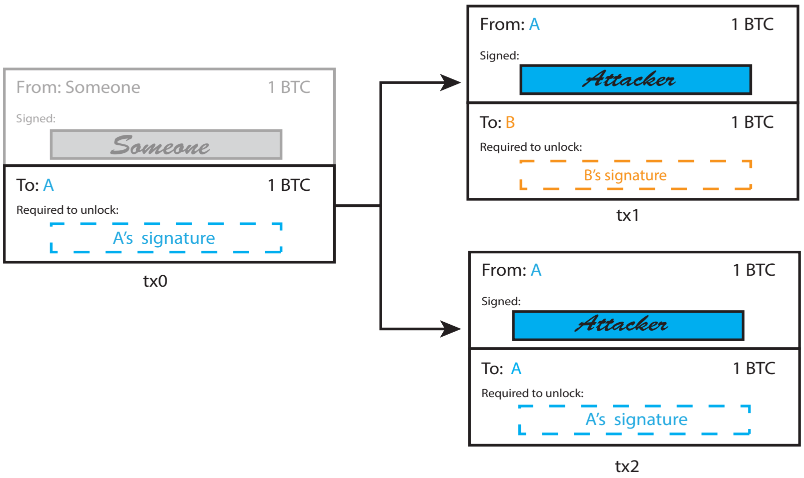

Fig. 1 Double-spending transactions.

new [8]. However, recent proposals, such as the Lightning network [9], have popularized this approach, that is now used in many other systems (e.g. mental poker playing [10,11] or proof-of-space schemes [12]). In such a way, and alternatively to other works [4], we address the double spending problem in fast payments by introducing a penalty for the attacker.

This penalty involves the disclosure of a private key. A similar penalty mechanism has also been proposed in [13] as a generic penalty for parties issuing contradictory statements. Our proposal presents a simpler solution, which does not require the use of additional protocols other than Bitcoin.

3 Background on Bitcoin transactions

As we have already stated, Bitcoin transactions are the tool for value transfer in the Bitcoin protocol. Bitcoin transfer is performed by using an unspent output of a previous transaction (UTXO) as the input of a new one. Therefore, Bitcoin transactions consume previous outputs and generate new ones.

The input of a Bitcoin transaction is formed, mainly, by three fields. The two first ones, namely previous transaction id and previous output index, uniquely identify the UTXO that is being claimed by the input. The third field, identified as `scriptSig`, provides the conditions to be met by the transaction to be valid and the payment be correctly performed. Such conditions were defined in the transaction output that is going to be spent. Both conditions, the specification in the output and the fulfillment in the input, are codified using the Bitcoin scripting language [14], a stack-based language defined in the Bitcoin protocol.

The output of a Bitcoin transaction includes two fields. The first one represents the value (in satoshis) that the output is holding.[2] The second field, named `scriptPubKey`, defines the conditions to be fulfilled to spend output.

Even though transactions within Bitcoin are, in most cases, bound to Bitcoin addresses (locking conditions) and redeemed providing digital signatures (unlocking conditions), the Bitcoin scripting language is flexible enough to allow the definition of many other scripts that encode different conditions under which the outputs may be spent, as we describe in the next section.

3.1 The Bitcoin scripting language

In order to spend a UTXO, the locking conditions specified by the script in its `scriptPubkey` field have to be met. The fulfilment of these conditions is provided by the values included in the `scriptSig` field of the input that spends the UTXO. To evaluate if an input can spend a corresponding output, the code included in the `scriptPubKey` is appended to the values included in the `scriptSig`, and the complete set of instructions is executed. Only if the execution evaluates to true, the input is able to spend the UTXO. Notice that this general approach allows not only to spend UTXO based on digital signatures but also to create much richer constructions, the so-called smart contracts.

[2] Notice inputs does not contain any value. The whole value of an output is consumed when it is used as an input of a new transaction. The difference between the amount claimed by the inputs and the ones specified in the new outputs is the fee collected by the miner.

Smart contracts can indeed specify complex conditions.[3] For instance, besides the popular Pay-to-Public-Key-Hash (P2PKH) and the Pay-to-Public-Key (P2PK) outputs where a standard digital signature must be provided to spend the UTXO, there exist multi-signature constructions, where a UTXO is locked under n public keys, and at least m matching signatures must be provided. Furthermore, even more, general constructions can be deployed using Pay-to-Script-Hash (P2SH) outputs. P2SH outputs encode an ad-hoc set of instructions in the form of a script. The hash of such script is set as locking conditions in the `scriptPubKey` field of an output. An input spending such an output must contain the corresponding script, along with any necessary data to make the whole script evaluate to true.

3.2 Digital signatures on Bitcoin

Digital signatures in Bitcoin are performed with the Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA has a set of system parameters: an elliptic curve field and equation $\mathtt{C}$, a generator G of the elliptic curve $\mathtt{C}$, and a prime q which corresponds to the order of G. The values for these parameters are defined to be `secp256k1` [15] for Bitcoin.

Let $*$ be 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. If either s or r are 0, go back to step 1.
6. Output: $sig(m) = (r, s)$

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 [16, 17]) 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 selected on step 1. Therefore, the selection of k is critical to the security of the system.

[3] An interested reader could refer to [14] for additional information about Bitcoin smart contracts and script types.

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

1. Recall that, by 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 created from k and the fixed parameters of the scheme, the r values 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 $d = \frac{s_2 k - m_2}{r}$.

Moreover, the leakage of private key information is not only restricted to the case where the exact same k values are used, but also to situations when similar k values are generated [18, 19].

Some Bitcoin wallets adopted deterministic ECDSA after this vulnerability was found to affect some Bitcoin transactions [20–22]. In deterministic ECDSA [23] the determinism in the algorithm comes precisely from the selection of k, which is derived from the message to sign and the private key using an HMAC construction. In this way, for a fixed message and private key, the value k is always the same. The HMAC construction ensures that the mapping between the message and k appears random from the point of view of an observer that does not know the private key. Key generation and signature validation algorithms remain the same as in the non-deterministic version of ECDSA. As we will see, our proposal can be used both with deterministic and non-deterministic ECDSA.

Taking advantage of this vulnerability to perform a private key disclosure in Bitcoin has been previously used for timestamping in data commitment schemes by Clark and Essex [24].

3.3 Bitcoin transactions propagation

As we already pointed out, Bitcoin transactions are propagated through a peer to peer network. Every node of the network broadcasts the transactions he generates and propagates transactions received from other network peers. As a protection against denial of service attacks, every peer performs different validations on every received transaction before its propagation,[4]

[4] See [14] for all the validation details.

like data format validation, digital signature verification, or correctness checks over the values involved in the transaction. Besides such validations, the node also validates that the received transaction does not spend an already spent output, either by a transaction already in the blockchain or by a transaction included in the node's mempool. In case that some validation fails, the node drops the transaction and, therefore, the transaction is not propagated any further.

However, since Bitcoin Core 0.12, Bitcoin includes a replace-by-fee mechanism (RBF) that allows transactions to signal replaceability. This mechanism was introduced to allow to increase the fee of an already broadcast transaction to boost its odds of inclusion in the blockchain. Therefore, if a transaction is flagged as replaceable with RBF, it can be replaced from the node's mempool by a newer transaction that spends the same outputs but includes a higher fee.[5] Moreover, this new transaction will also be further propagated throughout the network, since it is considered a valid transaction that replaces the previous one. This feature ensures that when a transaction is flagged as a RBF, a double spending transaction of the same UTXO will be propagated further than a double spending of a regular (non-RBF) transaction.

4 Double-spending prevention mechanism

Our proposed scheme discourages signers from performing double spending attacks in zero-confirmation transactions used in fast payment scenarios. Fast payment scenarios are those where the merchant delivers the goods or services when seeing the payment transaction in the Bitcoin network, without waiting for the transaction to be confirmed. Examples of such scenarios are onsite shopping where the buyer cannot wait 10 minutes to leave the shop after purchase or in trading platforms where a timely transaction can save/earn you money.

In our scenario, we assume that the adversary is the buyer that pays for some goods to a merchant, and that may have incentives to try to double-spend the payment in order to finish the interaction with both the goods and the payment amount.

We assume that he can perform a double-spending attack by generating multiple transactions that spend from the same output and broadcast them selectively in the Bitcoin P2P network. Additionally, we also assume all peers of the network have the same capabilities, that is, they can generate and broadcast double spend transactions (if they know the private key needed to generate a signature).

[5] Notice that this only affects transactions in the mempool, since transactions included in the blockchain are final and thus not replaceable.

In order to discourage double-spending attacks, we propose a mechanism to construct special transaction outputs. Such outputs can be spent with a single signature but have the property that if two different signatures for the same output are disclosed (for instance, in two different transactions spending the same output as a double-spending attack), the private key used to sign the transaction is revealed. This mechanism allows any observer to generate a third transaction spending the same output and sending the amount to an address controlled by himself.

To allow such a construction, we propose a new Bitcoin script that we call **fixed-r pay-to-pubkey script (FR-P2PK)**. This script is a variant of the standard pay-to-pubkey (P2PK) script where a signature is required to redeem, but a FR-P2PK script adds an additional condition to be able to spend the output: the signature must be made with a specific r value. Then, due to the ECDSA vulnerability described in Section 3.2, this particular condition discourages double-spending of that output. Indeed, if the sender generates another transaction that spends the same output and propagates it through the P2P network, the sender risks losing all the funds deposited in the address, because any peer that captures both transactions will be able to derive the private key. Recall that the value r is computed from the value k as described in Section 3.2, giving a one to one correspondence between them.

4.1 Basic prevention mechanism

Let Alice be a user that wants to take advantage of the proposed double-spending prevention mechanism and let $\{PK_a, SK_a\}$ be an ECDSA key pair belonging to Alice.

The double spending prevention mechanism is made of two phases: *initialization*, that is performed before the payment is made, and *fast-payment*, where the payment is executed.

4.1.1 Initialization

The initialization phase is performed beforehand. During this phase, Alice generates a *funding transaction* that transfers some funds from an output in her control to a FR-P2PK output also under her control. In order to do so, Alice chooses a random integer k and a public key PK_a (for which she knows the associated secret key SK_a), constructs the FR-P2PK output, and sends some funds to an output in her control (see Figure 2).

Alice broadcasts the funding transaction and waits for the transaction to be confirmed, upon which the initialization phase is considered terminated.

A single funding transaction may include multiple FR-P2PK outputs (with different public keys) in order to allow Alice to use the proposed prevention mechanism multiple times. Therefore, there is no need to perform the initialization phase before every fast-payment, as long as Alice has enough funds stored in FR-P2PK outputs. Alice may repeat the initialization phase if she runs out of unspent FR-P2PK outputs. Additionally, notice that Alice remains in control of all the funds deposited by the funding transaction, and she is able to transfer them back to a standard output whenever she wants.

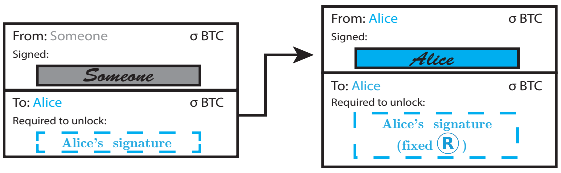

Fig. 2 Creation of the funding transaction.

At some point in the future, Alice wants to send some amount of bitcoins to another user Bob. Alice does not want to wait for the confirmation of the transaction and Bob is not willing to accept the transaction without confirmation. So they decide to use the proposed double-spending prevention mechanism, executing the fast-payment phase.

4.1.2 Fast payment

Alice creates a *fast-payment transaction* that pays to Bob spending from the FR-P2PK output of the funding transaction. The input script in the fast-payment transaction forces Alice (the redeemer) not only to prove she has the private key SK_a associated with the given public key PK_a by creating a valid signature, but also to deliver a signature that has been made using the specific k value that Alice chose during the initialization phase (see Figure 3). Alice broadcasts the fast-payment transaction to the Bitcoin P2P network.

Then, when Bob sees the fast-payment transaction in his mempool, he can validate that the output script of the funding transaction spent by the fast payment transaction is indeed a FR-P2PK script. If the validation is correct, Bob knows that if Alice tries to double spend the transaction she takes the risk of losing the Bitcoins of that output.

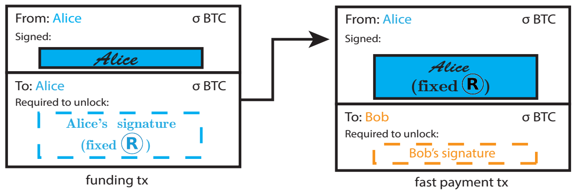

Fig. 3 Fast payment transaction.

If Alice decides to try to double spend the fast-payment transaction (see Figure 4, double-spending attempt), she needs to create a *double spending transaction* that also spends the FR-P2PK output of the funding transaction. This double spending transaction has to be valid, so it needs to include a (second) signature made with SK_a and the k value chosen on the initialization phase. Hence the moment the double spend transaction is created, there exist two different signatures made with the same private key SK_a using the same r. The signatures will be indeed different since the signed content (i.e. the transactions) will also be different. Recall that, because of the ECDSA vulnerability mentioned above, knowing two different signatures made by the same private key with the same k value is enough to derive the private key used to sign.

As a consequence, if Alice broadcasts the double spend transaction, she risks losing her funds. This happens because any observer that receives both transactions (the fast-payment transaction and the double-spending transaction) will be able to derive Alice's secret key SK_a and, as a consequence, create a third transaction, the *penalty transaction* that also spends the FR-P2PK output of the funding transaction but that sends the bitcoins to the observer. Note that this strategy may be performed simultaneously by any observer, ending with multiple penalty transactions, as it is depicted in Figure 4.

4.2 Disincentive-based prevention mechanism

The basic prevention method described in the previous section has a clear drawback. Suppose that Alice performs a purchase to Bob's shop and Bob accepts a fast transaction from Alice. When such transaction is received, Bob delivers the goods to Alice. However, once Alice has the goods, she may try to perform a double-spending attack. In case an observer sees both the fast-payment transaction and the double-spending

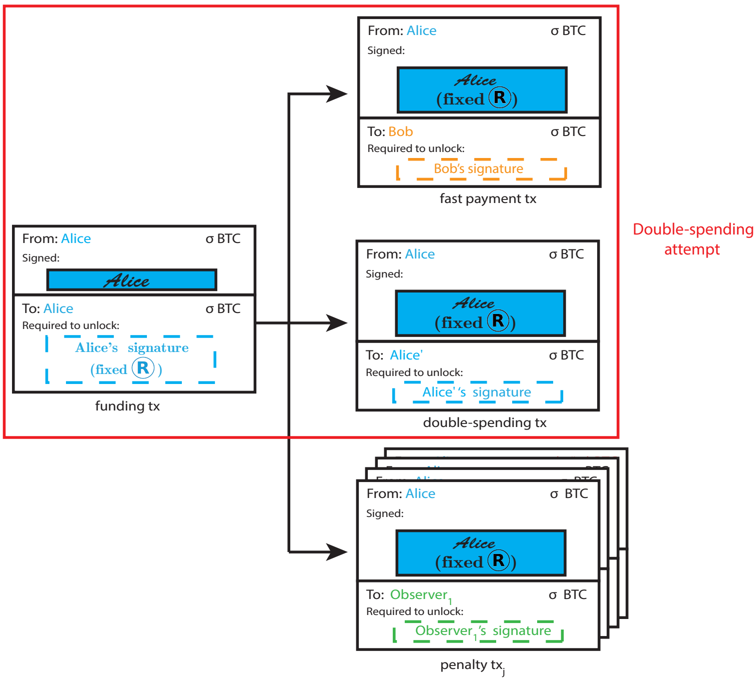

Fig. 4 Transactions involved in the scheme.

transaction, he constructs the penalty transaction, and manages to get it accepted in the blockchain, Alice loses her funds but Bob does not receive the payment. In that case, Bob may have complied with his part of the agreement (e.g. delivered the bought goods) but will not receive the agreed amount of bitcoins in exchange. Alice would have paid the agreed amount of bitcoins to a third party (the observer) instead of paying them to Bob, but she would remain in possession of the goods. The observer would obtain the total amount of the transaction. As a consequence, Alice will not lose anything by trying the double-spend (just the amount she is already willing to pay for it), and thus the proposed method may not be discouraging enough to prevent double spending attempts.

However, a minor modification to the method may be enough to discourage Alice from attempting any double-spend: enforcing that the amount deposited to the FR-P2PK output of the funding transaction is higher than the paid amount by a certain factor λ. Recall that a Bitcoin UTXO must be spent in its totality (i.e. it is not possible to spend a part of a UTXO). Therefore, if the FR-P2PK output has an amount bigger than what Alice must pay to Bob, Alice proceeds to create the fast-payment transaction including two different outputs: one that pays to Bob the agreed amount, and the other that pays back to her the change. This has no consequences on the normal operation of the protocol, that is, if Alice well-behaves, Bob ends up with his payment and Alice gets her change back. However, because the whole FR-P2PK output is spent, if Alice tries to double-spend the fast payment she risks losing the entire amount of the FR-P2PK output, and not only the amount paid to Bob.

As we will discuss in Section 6, by adjusting the λ factor Alice's penalty for double-spending can also be adjusted (and thus Bob's confidence on the fast payment).

Moreover, note that the fast-payment transaction may also have multiple inputs spending different FR-P2PK outputs which in turn allows Alice to perform payments of different amounts and with varying levels of penalty without having to freeze a high quantity of bitcoins into FR-P2PK outputs.

Notice that Bob does not get refunded in case of fraud. The objective of this work to counteract potential frauds by penalizing Alice. An alternative approach to refund Bob could be designed but remains out of the scope of the current proposal.

4.2.1 The role of the observers

The funding transaction is confirmed before starting the fast-payment phase, so any full node of the network is aware of its existence. Moreover, because it has an output with an easily identifiable script, the FR-P2PK script, any observer aware of the specification of our proposed mechanism is able to identify the transaction as a funding transaction belonging to our protocol. Therefore, such an observer will be able to monitor his mempool, looking for transactions that spend the FR-P2PK output. Once a transaction spending from the FR-P2PK output is seen, the observer is able to actively listen to the network, searching for any other transaction spending the same output. If the observer is able to catch a double-spending transaction, he should be able to construct a penalty transaction, moving the funds to an address controlled by himself. If the observer does not capture a double spending transaction, he may stop this active listening period and return to its normal behaviour when a transaction spending the FR-P2PK output is included in the blockchain.

To achieve the maximum level of propagation, and thus to spread awareness of the double-spending attempt, *fast-payment transactions* are flagged with replace-by-fee (RFB). By forcing a customer to flag a transaction in such a way a merchant enables the forwarding of future *double-spending transactions*, that could be detected by a higher number of nodes, increasing the odds of an observer receiving two instances of the transaction and, therefore, publishing a *punishment transaction*. Note that even though transactions are flagged with RBF, some actors may not have incentives to propagate duplicate transactions. This is the case, for instance, of miners that are aware of the protocol, that may prefer to mine their own penalty transaction instead of propagating a third-party penalty transaction (after having sent any of the other transactions). However, many other actors would indeed propagate penalty transactions, for instance: non-miner observers, that need to send their penalty transaction to a miner for inclusion in a block; nodes not participating in the protocol (e.g. nodes running the standard bitcoin client), that propagate RBF transactions by default; or nodes just aware of one of the transactions, that would not even be aware of which of the transactions it is.

5 Implementation details

In this Section, we describe how to construct the FR-P2PK output of the funding transaction as well as the inputs of the transactions that spend it, taking into account Bitcoin's signature format and scripting language.

First of all, notice that it would be possible to encapsulate the proposed FR-P2PK script into a standard P2SH output. However, doing so makes the funding transaction no longer recognizable as belonging to our protocol by external observers. Therefore, an observer that is aware of the existence of our protocol would be able to detect that the mechanism is being used only after one of the transactions spending the encapsulated FR-P2PK is seen in the network. This transaction will include the FR-P2PK script in the `scriptSig` (input script). The moment the observer processes this script, he can start the active listening period in which he looks for other transactions spending from the same funding transaction output. Because timing is critical in our scenario, we argue that using directly a FR-P2PK output in the funding transaction is the best alternative.

In a Bitcoin transaction, signatures are represented by a single hexadecimal value, which corresponds to the DER encoding of the two-element sequence of the r and s integers that make up an ECDSA signature. Figure 5 describes the format of a Bitcoin signature. Each integer r and s (see Section 3.2 regarding the notation) is encoded with three different fields: a 1-byte field with the `0x02` flag denoting the integer type, a 1-byte field with the size l of the integer (in bytes), and an l-byte field with the integer value itself. Then, the signature includes a 1-byte field with a flag denoting a sequence (`0x30`), a 1-byte field with the length of the sequence, the sequence of the two integers, and finally a 1-byte field with the hash type, a flag that indicates the parts of the transaction that are hashed and signed.

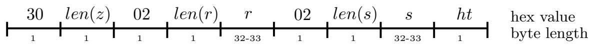

Fig. 5 Bitcoin signature format.

Both r and s are 32 byte integers. However, when the first bit of any of the values is set (that is, the first byte is $>$ `0x7f`), an additional byte (`0x00`) is added in front of the value, thus making it 33 byte long. The reason is that DER rules interpret this first bit as a sign, and therefore not adding `0x00` would cause the value to be interpreted as negative. Therefore, Bitcoin signatures range from 71 to 73 bytes. For the sake of simplicity, let us assume that we are dealing with 71-byte signatures, that is, signatures where both r and s are 32 byte long (Figure 6a).

Taking into account the format of signatures in Bitcoin, the script of a FR-P2PK output is defined as follows:

```
ScriptPubKey:
    OP_DUP <pubKey> OP_CHECKSIGVERIFY
    OP_SIZE <0x47> OP_EQUALVERIFY
    <sigmask> OP_AND <r> OP_EQUAL
ScriptSig:
    <sig>
```

where

– <pubKey> is the public key that will be used to validate the signature,
– <sigmask> is a 71-byte array that has ones in the positions where r and ht are specified and zeros in the rest of positions, and
– <r> is the 71-byte array that represents the integer r in DER format in the positions where it is found in a signature, 0x01 in the ht field, and zeros in the rest of positions.

Figures 6b-c show the construction of <sigmask> and <r>, respectively. Regarding the construction of the byte array <r>, on the one hand the integer r is derived uniquely from the randomly chosen k value (recall Step 3 of the ECDSA signature generation algorithm in Section 3.2). Note that any value of k may be used by the protocol, what matters is that it is fixed beforehand (that is, before the signature is made). On the other hand, the hash flag tag ht is set to 0x01, which corresponds to SIGHASH_ALL. This flag signals that the signed content corresponds to the entire transaction (except the signature scripts themselves). By enforcing that signatures cover the entire transaction, we ensure that a double spending attempt will include a signature different from the one found in the fast-payment transaction, and thus that observing both transactions indeed allows deriving the private key.

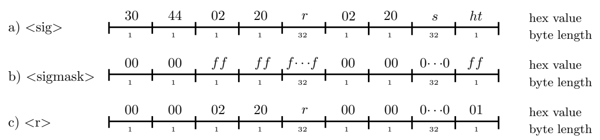

Fig. 6 Values used in the proposed script.

In order to redeem a FR-P2PK output, the input's scriptSig only needs to include a single signature. The signature must be correct when validated with the specified public key and must be performed with a specific r value. Otherwise, the validation will fail.

Let's analyse how does the script perform these validations. First, the script duplicates the signature (with OP_DUP). This is needed in order to be able to perform different validations over the same signature value. Then, the signature is validated against the specified public key using the check signature operand (<pubKey> OP_CHECKSIGVERIFY). After that, the length of the signature is checked (OP_SIZE <0x47> OP_EQUALVERIFY). Finally, a bitwise AND operation between the signature and <sigmask> is computed (with <sigmask> OP_AND), and the result is compared with <r> (<r> OP_EQUAL). If both values are equal (that is, the signature was made using the specified r and with a hash flag of 0x01), the script terminates successfully; otherwise, the script terminates with a False value on the stack, making it fail.

Note that the only way to ensure that the script succeeds is by providing a valid signature that matches the specified <r>. Therefore, two different transactions spending the same FR-P2PK output would include two different signatures made with the same private key and the same k, and thus by obtaining the two transactions, one is able to infer the private key that was used to create the signatures.

We have created a funding transaction in the Bitcoin testnet that exemplifies the proposed prevention mechanism. Following the transaction naming used in Figures 2, 3, and 4, the funding transaction[6] contains the output with a FR-P2PK script. The output of the funding transaction is currently unspendable due to the fact that it uses an OP_AND opcode that is disabled in the standard Bitcoin software.

6 Proposal analysis

In this section, we provide an analysis of the possible outcomes of performing a payment with the proposed mechanism. The analysis measures the benefits of each party taking part in the system to show how it discourages double-spending attacks. Table 1 summarizes the notation used in this section.

Table 1 Notation summary

Symbol	Meaning
τ_f	Fast-payment transaction
τ_d	Double-spending transaction
τ_{p_j}	Penalty transaction j
$Pr[\tau_x \in \mathcal{B}]$	Probability that transaction τ_x is included in the blockchain
σ	Payment amount
$\lambda \cdot \sigma$	Funding transaction output amount
γ	Value of goods

Our analysis makes the follow assumptions. First of all, we assume that Alice always generates the fast payment transaction since it is the triggering action for the payment. Once the fast payment has been generated, we assume that Bob sees the payment and, at that time and acting honestly, he delivers the goods to Alice. Furthermore, to focus the analysis in the proposed mechanism, we assume that at least one of the transactions of the system τ_f, τ_d, or τ_{p_j} will be confirmed. Although transactions do include fees, fees are

[6] https://www.blocktrail.com/tBTC/tx/8e27cae62d1df357b65b634a8482672d85f71804a5c7fc392050517a5bfeb04f

intentionally excluded from the computation of the payoffs, since they do not directly affect the result of our evaluation. The usage of the replace-by-fee mechanism, that will allow the propagation of multiple instances of the transaction, forces double-spending transactions to include a higher fee than the original transaction. However, it could be argued that a higher fee will incentive miners to mine a double-spending transaction instead of a fast-payment transaction, affecting not only the payoff but also the odds of either succeed or fail in the attack. Nonetheless, if a miner sees two instances of the transaction (fast-payment and double-spending) he would maximize his payoff as an observer by generating a third transaction (penalty) that pays all the amount to himself (instead of choosing to include any of the previous ones depending on the fees). Moreover, notice that our analysis can be seen as an upper-bound on the attacker's gains since including fees in the computation only lowers the attacker's payoff, but never increases it.

Of course, due to the double-spending protection of Bitcoin for on-chain transactions, at most one of these transactions gets into the blockchain, that is, the events $\tau_f \in \mathcal{B}$, $\tau_d \in \mathcal{B}$, and $\tau_{p_j} \in \mathcal{B}$ are mutually exclusive. Finally, notice that $Pr[\tau_f \in \mathcal{B}] + Pr[\tau_d \in \mathcal{B}] + \sum_j Pr[\tau_{p_j} \in \mathcal{B}] = 1$, since such probabilities depend on the distribution hash rate devoted to the interests of every set of users (Alice, Bob and the rest of the network, acting as observers) and we can assume that such sets will be disjunct.

Taking into account these assumptions, in Figure 7 we have represented how the possible final three states of the protocol we propose may be reached. If the fast payment transaction τ_f gets confirmed, then Bob receives the payment for the goods, Alice receives the change (the amount deposited to the funding transaction minus the payment) and the goods, and the observer does not intervene. If the double spending transaction τ_d is confirmed, then Alice gets everything (the full amount deposited in the funding transaction and the goods) and therefore both Bob and the observers do not obtain anything. Finally, if one of the penalty transactions τ_{p_j} is confirmed, then Alice obtains the goods but loses the full deposited amount that goes to the observer. Figure 7 also describes the possible paths that end up in each of the states.

We define the payoff $\mathcal{P}$ of any party participating in the protocol as the gains (or losses) obtained by deviating from the correct operation of the protocol. That is, the payoff of all parties (Alice, Bob, and the observers) will be 0 when no double spending is attempted (leftmost box in Figure 7).[7] In that case, there will be an equilibrium, since Alice pays the specified price for some goods and obtains the goods in exchange; Bob delivers the goods and gets paid for them, and the observers do not intervene. On the contrary, if Alice tries to double spend the payment, the equilibrium may be altered and the payoff will reflect the gains or losses each party assumes.

Then, Bob's payoff function $\mathcal{P}_B$ is given by the following expression:

$$\begin{aligned}\mathcal{P}_B = {} & Pr[\tau_f \in \mathcal{B}] \cdot (\sigma - \gamma) - \\ & - Pr[\tau_d \in \mathcal{B}] \cdot \gamma - \\ & - Pr[\tau_{p_j} \in \mathcal{B}] \cdot \gamma = \\ = {} & Pr[\tau_f \in \mathcal{B}] \cdot \sigma - \gamma\end{aligned}$$

Note that, for fixed σ and γ, Bob's payoff only depends on $Pr[\tau_f \in \mathcal{B}]$. Recall that our mechanism tries to disincentivize Alice from double-spending the payment transaction, but does not directly benefit the merchant (regardless of the λ value used by the protocol).[8]

In a similar way, Alice's payoff $\mathcal{P}_A$ is given by:

$$\begin{aligned}\mathcal{P}_A = {} & Pr[\tau_f \in \mathcal{B}] \cdot (\gamma - \sigma) + \\ & + Pr[\tau_d \in \mathcal{B}] \cdot (\gamma) + \\ & + Pr[\tau_{p_j} \in \mathcal{B}] \cdot (\gamma - \sigma\lambda)\end{aligned}$$

Alice's maximum payoff is, therefore, γ, and is obtained when Alice's successfully double spends the transaction, thus keeping the goods γ without paying anything. However, Alice's minimum payoff (that is, maximum losses) depends on λ, a parameter that can be adjusted in our protocol. Therefore, by adjusting λ, the protocol allows to tune Alice's losses, and so the risks she assumes by trying to perform a double spending attack. The bigger the λ is, the higher the risks Alice's faces on a double spend attempt.

Finally, an observer's j payoff is given by the expression:

$$\mathcal{P}_{O_j} = Pr[\tau_{p_j} \in \mathcal{B}] \cdot (\sigma\lambda)$$

Figure 8 shows the evolution of the parties payoffs as a function of $Pr[\tau_f \in \mathcal{B}]$ and $Pr[\tau_d \in \mathcal{B}]$[9] for the case where $\sigma = \gamma$ (the value of goods is equal to the price it is paid for them), for different values of the parameter λ. The payoff dimension is measured based on the value

[7] Here we assume that the price of the goods is equal to the value of the goods. If the price paid is higher than the cost, B's payoff is positive and reflects the benefit obtained from the sale.

[8] Note, however, that Bob may also act as an observer himself, being able to create a penalty transaction and trying to gain the observer's payoff.

[9] Since $Pr[\tau_f \in \mathcal{B}] + Pr[\tau_d \in \mathcal{B}] + \sum_j Pr[\tau_{p_j} \in \mathcal{B}] = 1$, fixing the first two probabilities uniquely determines the third operand.

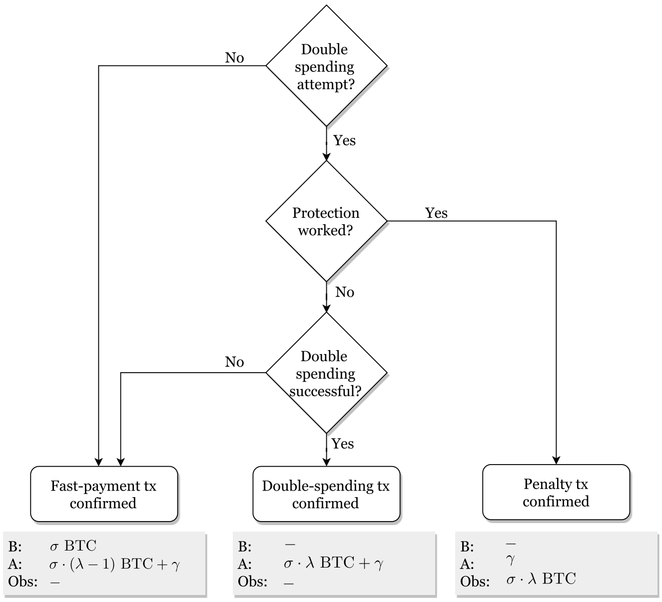

Fig. 7 Diagram showing the protocol's final states and the paths leading to them.

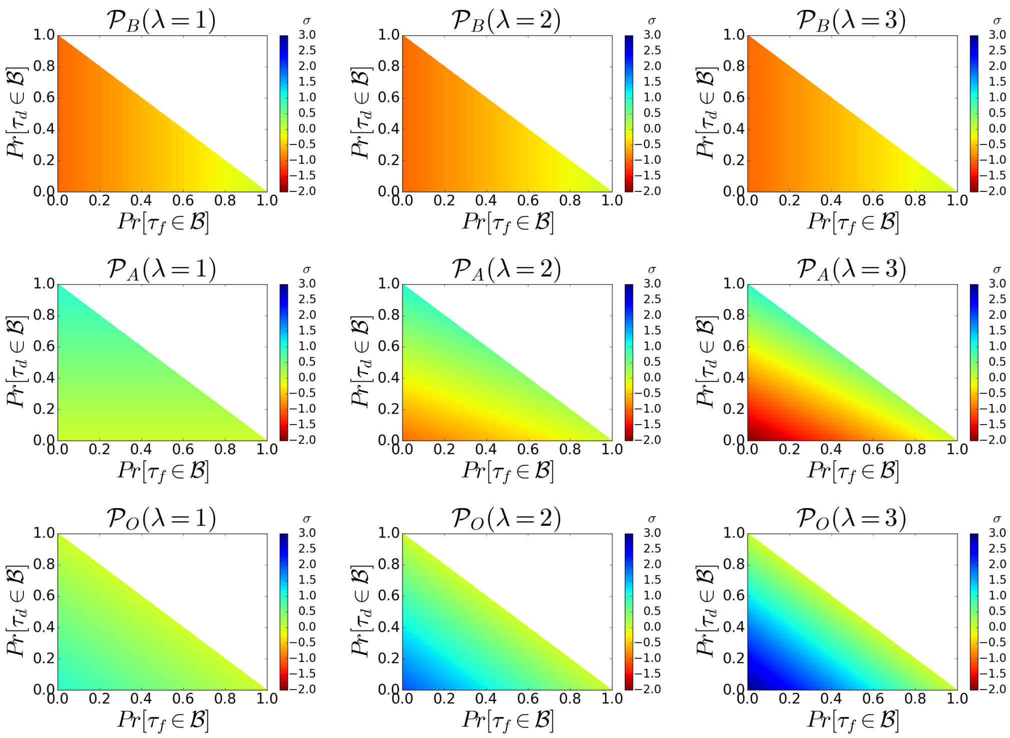

Fig. 8 Parties payoffs for $\sigma = \gamma$.

σ. That means that a payoff of 3 implies a benefit of 3 times the value of σ while a payoff of -3 implies a lost of 3 times the value of σ. The payoff results are thus proportional to σ.

The graphics show that, as expected, when there is no double spending attempt ($Pr[\tau_f \in \mathcal{B}] = 1$) there is an equilibrium in the parties' payoffs, and in all graphics we obtain $\mathcal{P}_B = \mathcal{P}_A = \mathcal{P}_{O_j} = 0$ (green zone). Note that, as λ increases, Bob's payoff (first three graphics) remains exactly the same since his payoff is independent of the parameter λ. On the contrary, Alice's payoff (next three graphics) depends on λ. With $\lambda = 1$, Alice's payoff is always positive or zero: Alice does not lose anything by trying to double spend and may even gain something if the attack is successful. That situation is the basic prevention mechanism described in Section 4.1. However, by increasing λ the scenario changes radically for Alice: the probabilities range at which Alice gains something from the attack decrease fast and, at the same time, for some probability values she even starts to get a negative payoff (that is, she has to assume losses). Finally, notice that the observer's payoff (last three graphics) is never negative, and his gains increase with λ.

Notice that our analysis does not assume any specific values on the probabilities $Pr[\tau_f \in \mathcal{B}]$, $Pr[\tau_d \in \mathcal{B}]$, and $Pr[\tau_{p_j} \in \mathcal{B}]$. However, as we have already indicated, such probabilities depend on the hash distribution of the Bitcoin network among mining the transactions, τ_f, τ_d, and τ_{p_j}. For that reason, in case the hash rate devoted to τ_d with respect the rest is low, the graphics show that Alice's payoff, for values $\lambda > 1$, is moving in the red zone thus being negative (Alice is losing money). The greater the λ value the bigger the red zone.

Providing values for the exact probabilities that parameterise our analysis is far from trivial because these probabilities are affected by the personal decisions of nodes that participate in the network. These personal decisions may not even attend to economical incentives only, but also political or even non-rational ones. That being said, we can, of course, imagine a specific scenario with rational actors and estimate the probabilities for that individual case. For instance, we can assume that all miners are aware of the protocol and try to optimize their immediate profit (that is, they would try to obtain the reward of a penalty transaction if they are able to do so). Additionally, we can also assume both the fast payment and the double-spending transactions are propagated to all miners of the network. Then, an attacker that controls the currently largest mining pool will have $Pr[\tau_d \in \mathcal{B}] = 0.24$ (BTC.com has 24.6% of the hash-rate as of today).[10] On the other hand, $Pr[\tau_f \in \mathcal{B}]$ will be 0, since no miner would mine the fast-payment transaction (each miner would mine their own penalty transaction). For $\lambda = 3$, this scenario results in an attacker's payoff of $\mathcal{P}_A = -1.28\sigma$, while the sum of the observers payoffs will be $\sum_{\forall j} \mathcal{P}_{O_j} = 2.28\sigma$. The second largest pool will be the *best* observer, with a payoff of $\mathcal{P}_{O_0} = 0.51\sigma$.

7 Conclusions

The speed at which payments in blockchain based cryptocurrencies can be performed is lower bounded by the block generation interval, which in Bitcoin is fixed to 10 minutes. In order to provide fast payments, one of the alternatives used in these scenarios is to rely on zero-confirmation transactions, i.e., transactions that have been seen on the network but have not yet been included in the blockchain. Experimental analysis have shown that, in Bitcoin, most of the transactions propagated through the network reach 75% of the nodes in less than 8 seconds [25], which is two orders of magnitude faster than the block production interval. However, zero-confirmation transactions are not secured by the standard Bitcoin double-spending protection mechanism, since this mechanism is applied to transactions included in the blockchain and zero-confirmation transactions are not yet in blocks by definition.

In this paper, we have presented a mechanism to secure fast payments within Bitcoin by reducing the risk of double-spending attacks in transactions with zero confirmations. The proposed mechanism discourages double spending attempts by creating a special type of outputs that enforce private key disclosure in case of a double-spending attempt. Any Bitcoin network user may act as an observer and obtain a reward by detecting double-spending attempts. The reward the observer receives is equal to the price the attacker pays as punishment for having tried to double spend a transaction and may be fixed by the receiver beforehand. Nevertheless, the attacker may have enough incentive to double-spend even when that will likely result in a loss for her, but pursuing Bob's lost, that could be more damaging. Such scenario may appear when the wealthy of the attacker is much higher than the receiver. However, since our proposal is an optional alternative to accept fast payments, Bob may opt-out of using our system in case his possible loss would be more than he can afford.

Further research will be focused on experimentally testing the proposed approach in a Bitcoin-like P2P network, in order to quantify the probabilities of each transaction entering the blockchain depending on the exact capabilities of the attacker (both in terms of network connectivity and hash power) and the percentage of nodes of the network that are aware of the existence of the prevention mechanism. In turn, this would allow us to better evaluate the risks the merchant is facing with each transaction and to study the overhead of transactions relayed through the network by the usage of our protocol.

[10] `https://web.archive.org/web/20180522080137/https://blockchain.info/pools`

Compliance with Ethical Standards

Funding: This work is partially supported by the Spanish ministry under grant number TIN2014-55243-P and the Catalan *Agència de Gestió d'Ajuts Universitaris i de Recerca* (AGAUR) grant 2014SGR-691.

Conflict of Interest: The authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. G.O. Karame, E. Androulaki, S. Capkun, in *Proceedings of the 2012 ACM conference on Computer and communications security* (ACM, 2012), pp. 906–917
2. G.O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, S. Čapkun, ACM Trans. Inf. Syst. Secur. **18**(1), 2:1 (2015). DOI 10.1145/2732196. URL http://doi.acm.org/10.1145/2732196
3. GAP600. Gap600 bitcoin transactions guaranteed. http://gap600.com/ (2017)
4. T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, S. Welten, in *Proceedings of the IEEE Internation Conference on Peer-to-Peer Computing (P2P)* (Trento, Italy, 2013., 2013)
5. A. Biryukov, D. Khovratovich, I. Pustogarov, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (ACM, 2014), pp. 15–29
6. A. Biryukov, I. Pustogarov, in *Security and Privacy (SP), 2015 IEEE Symposium on* (IEEE, 2015), pp. 122–134
7. J.A. Kroll, I.C. Davey, E.W. Felten, in *The Twelfth Workshop on the Economics of Information Security (WEIS 2013)* (2013)
8. I. Bentov, R. Kumaresan, in *International Cryptology Conference* (Springer, 2014), pp. 421–439
9. J. Poon, T. Dryja, The Bitcoin lightning network: Scalable off-chain instant payments. Tech. rep. (2015). https://lightning.network
10. I. Bentov, R. Kumaresan, A. Miller, in *International Conference on the Theory and Application of Cryptology and Information Security* (Springer, 2017), pp. 410–440
11. B. David, R. Dowsley, M. Larangeira, in *Proceedings of the 2018 International Conference on Financial Cryptography and Data Security* (2018)
12. S. Park, A. Kwon, G. Fuchsbauer, P. Gazi, J. Alwen, K. Pietrzak, in *Proceedings of the 2018 International Conference on Financial Cryptography and Data Security* (2018)
13. T. Ruffing, A. Kate, D. Schröder, in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (ACM, 2015), CCS '15, pp. 219–230
14. A.M. Antonopoulos, in *Mastering Bitcoin: unlocking digital cryptocurrencies* (O'Reilly Media, Inc., 2014), chap. 5
15. Certicom Research, Sec 2: Recommended elliptic curve domain parameters. Tech. rep., Certicom Corp. (2010)
16. A.J. Menezes, P.C. Van Oorschot, S.A. Vanstone, *Handbook of applied cryptography* (CRC press, 1996)
17. C. Paar, J. Pelzl, *Understanding cryptography: a textbook for students and practitioners* (Springer Science & Business Media, 2009)
18. P.Q. Nguyen, I.E. Shparlinski, Journal of Cryptology **15**(3), 151 (2002)
19. M. Bellare, S. Goldwasser, D. Micciancio, in *Annual International Cryptology Conference* (Springer, 1997), pp. 277–291
20. N. Schneider. Recovering Bitcoin private keys using weak signatures from the blockchain (2013)
21. F. Valsorda. Exploiting ECDSA failures in the Bitcoin blockchain (2014)
22. Bitcoin.org. Android security vulnerability (2013). URL https://bitcoin.org/en/alert/2013-08-11-android
23. T. Pornin. Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA). RFC 6979, Internet Request for Comments, The Internet Society (2013)
24. J. Clark, A. Essex, in *Financial Cryptography and Data Security*, *Lecture Notes in Computer Science*, vol. 7397, ed. by A.D. Keromytis (Springer Berlin Heidelberg, 2012), pp. 390–398. URL http://dx.doi.org/10.1007/978-3-642-32946-3_28
25. C. Decker. Data propagation: How fast does information move in the network? http://bitcoinstats.com/network/propagation/ (2017)