A New Look at the Refund Mechanism in the Bitcoin Payment Protocol

Sepideh Avizheh[1], Reihaneh Safavi-Naini[1], and Siamak F. Shahandashti[2]

[1] University of Calgary, Calgary, Alberta, Canada
(sepideh.avizheh1,rei)@ucalgary.ca
[2] University of York, York, United Kingdom
siamak.shahandashti@york.ac.uk

Abstract. BIP70 is the Bitcoin payment protocol for communication between a merchant and a pseudonymous customer. McCorry et al. (FC 2016) showed that BIP70 is prone to refund attacks and proposed a fix that requires the customer to sign their refund request. They argued that this minimal change will provide resistance against refund attacks. In this paper, we point out the drawbacks of McCorry et al.'s fix and propose a new approach for protection against refund attacks using the Bitcoin multisignature mechanism. Our solution does not rely on merchants storing the refund request, and unlike the previous solution, allows updating the refund addresses through email. We discuss the security of our proposed method and compare it with the previous solution. We also propose a novel application of our refund mechanism in providing anonymity for payments between a payer and payee in which merchants act as mixing servers. We finally discuss how to combine the above two mechanisms in a single payment protocol to have an anonymous payment protocol secure against refund attacks.

1 Introduction

Since the introduction of Bitcoin [12] in 2008, it has been widely adopted by merchants and as of February 2015, the number of merchants who accept Bitcoin as a payment method has passed 100,000 [8]. BIP70 [1] is the payment protocol between a pseudonymous customer and a merchant with PKI certificate. The protocol, provides a number of properties to improve the interaction between the two (e.g. allows merchant's address to be human-readable) and providing the necessary guarantees (e.g. provides a proof of payment to the customer that can be used for dispute resolution). One important feature of the protocol is that the customer can specify refund addresses that will be used by the merchant in the case of overpayments, or refunds for cancelled orders.

McCorry, Shahandashti and Hao [11] however showed two *Refund attacks* that are referred to as, Silkroad trader attack and Marketplace trader attack separately, on BIP70 that exploits the one-way authentication of the protocol. In Silkroad trader attack, a malicious customer uses Refund mechanism of payment to an honest merchant to pay to a Silkroad trader. This is by simply

putting the address of the Silkroad trader as the refund address, and cancel the transaction within the allowed period. Authors carefully analysed the steps of the attack and showed its feasibility by implementing it. They also considered a second attack, called Marketplace trader attack, in which a rogue trader plays the role of Man-in-the-Middle (MITM) between the customer and a reputable merchant and effectively direct the customer's payment to its own address after sending a message to change the refund address. In both these attacks, the analysis of blockchain data will not reveal the attacks. In McCorry et al.'s solution the customer signs the refund addresses by the public key he has used in the payment (the signature is called proof of endorsement) and so effectively links the refund addresses to the customer address. This prevents the first attack because customer cannot deny not knowing the Silkroad trader, and discourages the second one that required the merchant to update the refund address through the email.

The solution is carefully described and analyzed. One important shortcoming of the solution however, is that, providing protection against the first attack required the merchant to maintain a database of all proof of endorsements and signed transactions to be used in the case of dispute: that is merchant being able to provide a proof that they have followed the customer's request and they are not in collusion with the Silkroad trader. This is what we refer to as *stateful* solution. The solution will be costly to implement locally (because of possible system failure or subversions) and need additional (possibly cloud-based) reliable storage and management. Note that the proofs in the database indicates the relation of the customers to refund addresses which should be kept confidential which will add to the cost of maintenance.

Our contributions. In Section 3, we introduce an alternative approach which we call *stateless*, meaning that the merchant does not need to keep the database. The solution idea is as follows: To reply to a refund request, merchant creates two transactions; one that locks the bitcoins to both the refundee and the customer, and one that sends the bitcoins to the customer alone, after a lock time has passed. To claim the bitcoins, the customer should sign the transaction for the refundee. This proves that the customer knows the refundee and the refundee's address is authenticated. If the refundee and the customer do not know each other, as is the case in the Marketplace attack, the first transaction is never unlocked, but the customer can use the second transaction to redeem the bitcoins. We will discuss how the scheme can be modified when there are more than one transaction issuer (this is the refund condition that was considered by McCorry et al.). The details of this solution is given in Section 3. In the case of Marketplace trader, our solution does not put any restriction on the BIP70. This is in contrast to the previous solution that had the restriction that refund address change not to be accepted through email. Such a restriction which is not specified by the BIP70 would likely to be ignored in practice, rendering the solution insecure.

In section 4 we consider a novel application of Refund mechanism for providing anonymity for payments. The main observation is that refund addresses

in Bitcoin can effectively provide a level of indirection that if carefully used, can decouple the payer and payee. In our proposal, the merchant provides a mixing service that allows the customers to pay for other services and to other merchants using a combination of overpayment and refund. By "mixing" transactions of multiple customers the linkage of transactions using their payer and payee fields, as well as values, will be removed.

Finally, in Appendix A we discuss how the above two mechanisms can be combined to provide an anonymous payment protocol with security against refunds attacks.

2 Preliminaries

2.1 The BIP70 Payment Protocol

BIP70 [1] is a payment protocol that defines the sequence of messages communicated between a customer and a merchant. The protocol consists of three messages: *payment request*, *payment*, and *payment acknowledgment*. It proceeds as follows.

After the customer selects an item from the merchant's website and clicks to pay, the merchant responds by sending a *payment request* message. This message contains *payment details*, the information related to merchant's X.509 certificate (PKI type and PKI data), as well as the signature of the merchant on the hash of the payment request. Here *payment details* consists of the Bitcoin address that the customer should send the bitcoins to, the time that request has been created, an expiration time, a memo containing notes to the customer, a payment URL, and finally the merchant data which is used by the merchant to identify the payment request.

The customer's Bitcoin wallet verifies the signature and the merchant's identity, the information in the payment details, such as the time of the request creation and expiry, displays the merchant's identity, the amount to pay, and the memo to the customer and asks the customer whether they want to continue. If confirmed, the wallet will create the necessary Bitcoin transactions for the payment and broadcast them to the peer-to-peer network. Then, a *payment* message is sent to the merchant. This message consists of the merchant data from the *payment details* in payment request, one or more valid Bitcoin transactions, the *refund_to* field which specifies a set of refund amount and address pairs to be used in the case of a refund request, and a note for the merchant (memo).

When the merchant receives the payment message, it verifies that the transactions satisfy the payment conditions, broadcasts the transactions, and sends back a *payment acknowledgment* message. This message contains a copy of the payment message and a final memo including a note on the status of the transaction.

BIP70 does not specify how the payment request message should be downloaded, but requires that the payment and payment acknowledgment messages are communicated over a secure channel (such as HTTPS).

BIP70 does not explicitly define a refund protocol. It is implicitly assumed that if the customer requests a refund identifying the payment by the *merchant data* field, the merchant issues a refund transaction which sends the refund amounts to the corresponding refund addresses specified in the *refund_to* field of the *payment* message. Figure 1 shows the communication flow in BIP70 and its implicit refund procedure.

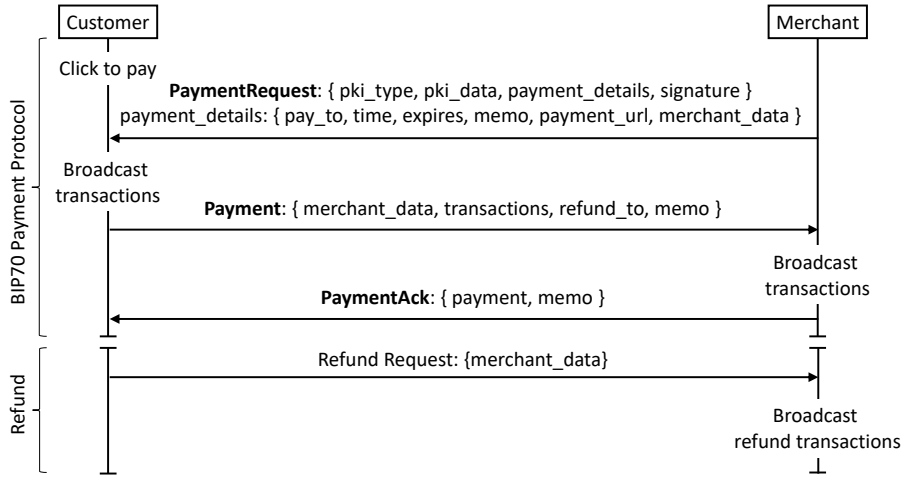

Fig. 1. The BIP70 payment protocol and its refund procedure.

2.2 Refund attacks

McCorry et al. propose two attacks on the refund process of BIP70 [11]. These attacks work even if a secure channel such as HTTPS is used for communication between parties. We briefly describe these two attacks in the following.

Silkroad Trader attack. The refund addresses provided by the customer (in the *refund_to* field) are in no way endorsed and can be repudiated at a later time. This means that a malicious customer may abuse the refund mechanism to relay their payment to an illicit trader (here called the Silkroad trader) through an honest merchant. The customer simply provides the illicit trader's address as the refund address to the merchant and thus when a refund is requested, the merchant will send the refund to the illicit Trader. The customer can later deny abusing the refund mechanism and the merchant will have no way to prove they have been cheated. Figure 2 shows the interaction among parties in this attack.

Marketplace Trader attack. Some merchants allow customers to specify new refund addresses upon a refund request. The customer requesting the refund is not authenticated. This means that any entity who has knowledge of the

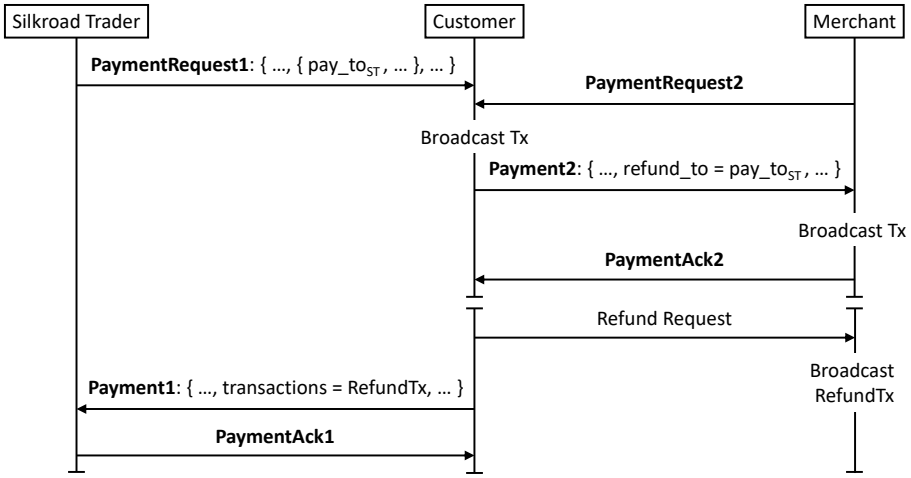

Fig. 2. The Silkroad Trader attack.

payment identifier (specified in the *merchant data* field of the payment details in the payment request message) can request a refund to any arbitrary account. This is the basis for the Marketplace Trader attack, in which a rogue trader acts as relaying man-in-the-middle for the payment request message between the merchant and the customer. Hence, the rogue trader is able to find out *merchant data*. At a later time, the rogue trader requests a refund to an arbitrary address and is able to steal the funds. Figure 3 shows the interactions among the parties in this attack.

2.3 McCorry et al.'s Solution to Refund Attacks

McCorry et al. propose to include in the payment message a "proof of endorsement" for refund addresses. To do this, each customer address involved in the payment protocol is required to produce a digital signature on (and therefore "endorse") a corresponding refund address. Employing this solution, at the end of a successful payment protocol, the merchant will be in possession of a proof of endorsement for each refund address. Such a proof can be presented and verified by a third party in case of a Silkroad Trader attack to implicate the malicious customer. Besides, since such a proof of endorsement is valuable for merchants, McCorry et al. argue that it will discourage merchants to accept new refund addresses unless accompanied by a proof of endorsement, resulting in reducing the possibility of Marketplace Trader attacks.

As noted earlier, maintaining a secure and robust database to store proof of endorsement messages is a security bottleneck of the system and can particularly become expensive for smaller merchants with limited resources. McCorry et al. noted other limitations of their solution due to Bitcoin inherent problems, including transaction malleability. In particular, the customer can tamper with

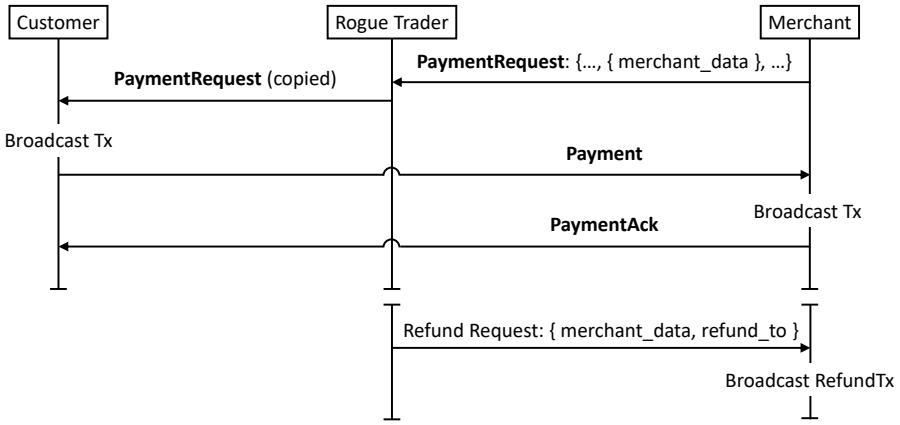

Fig. 3. The Marketplace Trader attack.

the transaction by re-signing and then broadcasting it to the network. Re-signing will change the transaction hash and the proof stored in the database will not match. Hence, for effective protection against attacks, merchants will need to also store payment transactions as well as payment request and payment messages which are required to verify the proof of endorsement. Therefore, the actual storage overhead of McCorry et al.'s solution is much larger than only keeping proofs of endorsement.

2.4 Multisignature and Time-Locked Transactions in Bitcoin

Although it is convenient to think of Bitcoin transactions as sending funds to certain account addresses, technically what the transaction specifies is a set of redemption criteria in a certain script language. Any subsequent transaction which satisfies the redemption criteria may authorize the transfer of funds made available in the original transaction.

The most popular script is "Pay to Public Key Hash" (P2PKH), which requires a signature corresponding to an address, hence effectively sending the bitcoins to the address. Typical Bitcoin transactions use this script.

Another popular and more versatile script is "Pay to Script Hash" (P2SH), which requires satisfying a script, the hash of which is listed. P2SH can be used to implement a diverse range of transactions including *multisignature* transactions. A k-of-n multisignature transaction requires k signatures corresponding to k addresses within a set of n specified addresses to be present to redeem the funds in the transaction.

An interesting script which can be combined with the ones discussed above is one that effectively freezes the transaction funds until a time in the future to create a so-called *time-locked* transaction. The funds in a time-locked transaction

cannot be spent by any other transaction until a certain (absolute or relative) time in the future.

3 A new approach for protection against Refund attacks

We propose a solution to Refund attack that does not require the merchant to maintain a secure and robust database, and is resilient to Bitcoin transaction tampering. We consider two types of attackers.

- **Online attacker**, intercepts the communication channel and sees all the input/output messages of a merchant.
- **Offline attacker**, has only access to the blockchain data.

To simplify our description we first assume BIP70 communication is over HTTPS and so we only need to consider an offline attacker. We then show how to secure the protocol against an online attacker.

Our goal is to provide the following properties for Refund mechanism.

Stateless. No information that links the customer and the refund addresses will be stored at the merchant's database.

Robust. Refund mechanism works correctly against a strong attacker who breaks into the merchant's system and arbitrarily delete or tamper with the stored data.

We also aim to stay with the specification of BIP70 and do not add extra restriction including *not* accepting refund address by email. Note that Refund addresses are valid for two months from the time of the payment [1], and during this period the customer should be allowed to change the refundees' addresses for example when an existing refundee has lost their wallet. Coinbase and Bitpay [7, 6] both accept refund address updates via email.

3.1 Our Solution

We use capabilities of Bitcoin transactions, namely multisignature and time-locked transactions, to link the refund addresses to the customer. Thus the merchant's evidence for innocence is stored in an immutable distributed database.

The Refund mechanism works as follows. The merchant creates a 2-of-2 multisignature transaction, and hence binds the refund amount to both the customer and refundee. Then, to make the protocol robust in the case that one of the addresses is not available, a second transaction is created. This second transaction is a time-locked transaction, and the customer is its only recipient. Merchant uses a lock time for this transaction to give priority to the first transaction. If the customer and the refundee could not collaborate to redeem the bitcoins, the customer is able to claim them after the lock time. Note that the lock time creates a delay in the system only if the customer does not know the refundee, which uses the setting of the Marketplace trader attack. In other words, the second transaction is a back up for system robustness (see Figure 4).

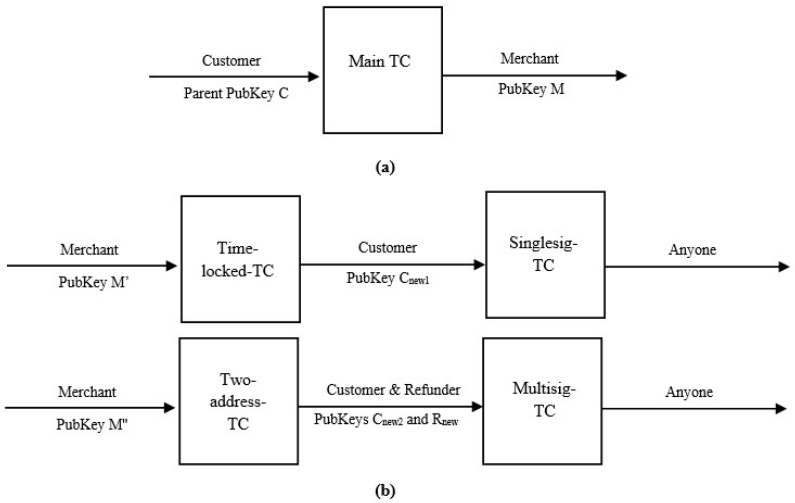

Fig. 4. (a) The main transaction. (b) The proposed Refund mechanism, in which merchant locks the transaction to the customer and the refundee and if they spend this transaction later, the merchant is assured that they know each other.

In addition, to satisfy address freshness[3] the merchant deterministically creates fresh addresses from the public key of the customer and then masks them with a Diffie-Hellman key derived using the fresh address of the customer and the private key of the merchant. This is to ensure that only the merchant can see the relation of the Refund key to the payment transaction. To derive fresh addresses we have assumed that customer has a deterministic wallet based on BIP32 [16]. Most Bitcoins wallets support BIP32 and so this is a reasonable assumption. A deterministic wallet generates a tree of public/private key pairs on elliptic curve E; for example for a 1 level tree, it creates 2^{31} *hardened* and 2^{31} *non-hardened* keys. Hardened keys, are public keys that their associated private keys can only be known before generation of public key. Non-hardened keys however allow anyone to derive a valid public key from them, while the owner of the parent private key can generate the respective child private key. In our protocol, the customer address in the payment transaction is a non-hardened public key which is used as a parent key by the merchant to derive child public keys. The customer knows the respective private keys and can also create the Diffie-Hellman key using the child private key and the public key of the merchant. Hardened keys can be used for refund addresses. The step by step process is give below (see Figure 5):

Key generation. Customer wallet software generates a tree of public/private key pairs using BIP32 [16]. Each private key is an integer in F_q, where $q = p^n$

[3] It is recommended that Bitcoin transactions use fresh addresses [4]. That is, users should create fresh public keys for each transaction and do not reuse addresses. This is for better privacy.

is a prime power, and each public key is a point on the elliptic curve E over F_q. Let $Pk_c = cP$ be a non-hardened public key. A child public key can be computed by anyone using this parent public key as follows:
$Pk'_c = Pk_c + H(Pk_c, chaincode, index)P$; where Pk'_c is a child public key, chain code is the 256 rightmost side of the previous hash using $HMAC\_SHA512$, and index is the index number of the generated child key in the tree (see [16] for details).
Pk'_c is the derived child public key, which is $Pk'_c = c'P$, but only the customer who knows the parent private key can compute the child private key, c'.

Click to pay. Customer visits the merchant website and chooses an item, then clicks on "pay".

Payment request. Merchant sends the payment request message including their public key, $Pk_m = mP$. This public key is unique for each customer.

Payment message. After authenticating and authorizing the merchant, the customer chooses one of their non-hardened public keys, Pk_c, and generates a payment transaction, that we call *MainTC*; this transaction transfers the cost of the chosen item to the merchant. Then, the customer creates a payment message based on *MainTC*, and specifies the refund addresses, $(Pk_{r_1}, Pk_{r_2}, \ldots, Pk_{r_n})$, and the amount refund for each address.

Payment ack. The merchant detects *MainTC*, and returns an acknowledgement message to the customer.

Refund request. Within a 2 month period from the payment request [1], the customer can use the addresses provided in *refund_to* field to receive their refund. In this case, the merchant does the following:

1. Derives child keys, Pk'_{c_i} for $1 \leq i \leq n+1$, from the parent key Pk_c.
2. Masks the child keys as $Pk''_{c_i} = Pk'_{c_i} + H(mPk'_c)P$ for $1 \leq i \leq n+1$.
3. Creates and broadcasts two transactions, RefundTC1 and RefundTC2. RefundTC1 is a P2SH transaction which sends bitcoins to whom provides signatures using Pk''_{c_j} and Pk''_{r_j} for $1 \leq j \leq n$. RefundTC2 is a P2PKH transaction which sends the bitcoins to $Pk''_{c_{n+1}}$ with a determined lock time, e.g. one week.

3.2 Protection against Silkroad trader attack

In Silkroad trader attack, the customer wants to remove their link to the Silk Road by using a victim merchant. In our approach the refund transaction has been locked to the customer and the refundee (that is Silk Road here), and so if they redeem the bitcoins in the next transaction it means that they know each other and the linkage is disclosed. This linkage however is hidden from those who observe the Bitcoin blockchain and so the merchant is the only one who knows this linkage. Thus he can prove that a payment transaction and a refund transaction are linked to each other by, first deriving the child keys from the payment transactions and adding the respective Diffie-Hellman key (based on the public key of the merchant and the customer's child key) to them, then finding the refund transaction with this child key, and finally showing that the

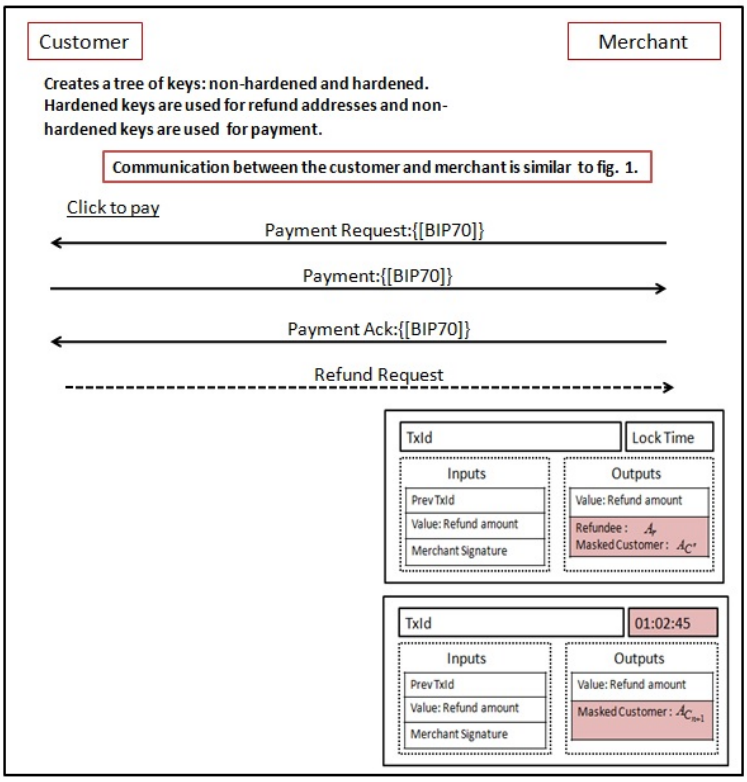

Fig. 5. Protection against Refund attacks; Communication flow is similar to BIP70 [1]. Upon a refund request, merchant creates two transactions to lock the bitcoins to both the customer and the merchant. He also generates a transaction which pays the same amount to the customer only; this transaction has a lock time.

transaction has been spent by the customer and the refundee, and so they must know each other. The merchant does not need to store anything in their database and the proofs are robustly preserved.

3.3 Protection against marketplace trader attack

If the customer provides a refund address during BIP70 protocol run and later update it via email, the merchant just uses the newest refund address and locks the refund amount to both the refundee and the customer. If the customer knows the refundee and signs the transaction for them to redeem the bitcoins, the transaction will be finalized; else if the rogue trader sends their own address to the merchant, they cannot later claim the bitcoins since the customer will not sign the transaction. The customer can claim the refund, if they notice the attack, after the lock time. Thus the approach protects the customer and provides the possibility to update the refund addresses.

3.4 Discussion

Our goal is to minimize the storage cost of the merchant, preferably limit to a constant, while providing robustness against refund attacks. In the following, we show how each of the mentioned properties will be satisfied.

Minimizing protocol state information. A merchant may store information that are communicated during a payment protocol for various reasons, including bookkeeping, refund or exchange, or statistics about customers and/or products. Here we do not consider bookkeeping that is mainly for accounting purposes and/or the ability to honour refund or exchange policies. Nor we consider data storage that are for statistical analysis purposes. As noted in [11], using bitcoin payment protocol requires merchant to store evidences to help them protect against refund attacks. We refer to this last type of information as the *state information of bitcoin payment protocol*. This information must be kept for sufficiently long time to protect against refund attacks.

Consider a single run of the payment protocol. The merchant must store transaction ids of the MainTC, and both the refund transactions that are Time-locked-TC and Two-address-TC. Merchant must also store the chain code and the indices that they have used to derive the refund child keys, and their own public key that is used for masking. In the case of a dispute, the merchant can retrieve the main and the refund transactions, from the blockchain, using their stored transaction ids, use the chain code, index and the public key of the customer in the MainTC, to derive the related child key, and use their own private key m, to re-create the masked address, $Pk''_c = Pk'_c + H(mPk'_c)P$. If the transaction with this address is a spent transaction, it implies that the customer with the public key Pk_c and child public key Pk'_c, is connected to the refundee. Therefore, to expose the linkage between a customer and a refundee, merchant must store some pointers to the evidence that are transaction ids, the chain code, the index needed for child key, and his own private/public key. Each pointer is 32 bytes and so 3×32 bytes are needed for the three transactions ids, and 32 bytes for the chain code and 2×32 bytes for two indexes, one for the child key of customer in Time-locked-TC and one in Two-address-TC, resulting in 192 bytes in total. Note that the merchant can always use a deterministic approach for the child key indexes, in which case they do not need to store them (storing the number of indexes he has used is enough) and so the storage size will be reduced to 130 bytes and will be independent of the number of refundees.

In the following we compare the storage cost of our proposed protocol with that of [11] (see table 1). In [11], proof of endorsement is a signature that must be locally stored. Verification of this signature needs information about the main transaction and the communicated messages including, the main transaction inputs, refund address, refund value, the memo from the customer, and payment request message. Merchant also needs to store the transaction ids. Let L_S denote the required storage for the signature (size of the signature). The main transac-

tion input with one signer is at least 146 bytes [4]. Other values are, refund address which is 20 bytes (doubled hash), refund value is 8 bytes, memo and payment request message sizes are denoted by L_Pay and can reach 50,000 bytes. Finally a transaction id is 32 bytes. Thus in total, for each refundee, $238+L_S+L_{pay}$ bytes must be stored at the merchant, and this cost grows linearly with the number of refundees. Thus the total storage for n refundees will be $210+n\times 28+L_S+L_{pay}$ bytes which is significantly higher than our scheme.

Table 1. Storage size (in bytes) of our approach vs. [11]

Scenario	[11]	Our approach
1 refundee	$238 + L_S + L_{pay}$	130
n refundee	$210 + L_S + L_{pay} + 28n$	130

Robustness. The payment protocol must work correctly if an attacker breaks into the system and arbitrarily delete or tamper with the stored data. In [11], if the local database that stores the signature (proof of endorsement) is crashed, the evidence of the collusion will be removed and the merchant will become completely vulnerable. In our proposed approach however merchant can exhaustively search all main transactions that are stored on the blockchain, and compute all possible values for the customers' refund addresses, and find the customer who is connected with a particular refundee. To do so, merchant first finds all transactions that have been issued to his address. These transactions, i.e. MainTC transactions, are detectable through merchant's certified addresses. Additionally, merchant derives the respective child key using all of their possible values for chain code and indexes. They also try all their private keys to mask the child key and compare the result with the customer address connected to the refundee. If they match, MainTC, public key, index and merchant's private key are stored as evidences that connects the customer to refundee.

3.5 Multi-signer payment transaction

So far we assumed that payment transaction is generated by a single customer. In the following we show that a functionality similar to [11], for multiple signer case can be provided.

When a payment transaction is created by multiple signers, endorsing the refund address by a single signer has the danger of allowing them to steal the bitcoins. McCorry et al.'s solution is resilient against this attack because each key that is used in the payment transaction should endorse its own refund address. In our scheme however, the customer does not provide any signature before the refund and so the merchant does not know which refund address belongs to

[4] Previous transaction hash is 32 bytes, previous Tx-out index is 4 bytes, Tx-in script length is 1-9 bytes, public key is 33 bytes in compressed format, signature is 72 bytes, sequence number is 4 bytes.

which signer in the *MainTC*, and blindly locks the bitcoins to all of the signers. Although this solution prevents Refund attacks and provides Statelessness and robustness, it is not efficient in the sense that the refundee must interact with all of the signers (that he may not know) to claim the bitcoins. In addition, one of the signers may refuse to sign. We use the following *refund_to* field in the payment message; if there are n customers with public keys Pk_{c_1}, Pk_{c_2},...,Pk_{c_n} and p refund addresses as $Pk_{r_1}, Pk_{r_2}, ..., Pk_{r_p}$, *refund_to* field will be $\{(Pk_{c_1}, Pk_{r_1}, v), (Pk_{c_2}, Pk_{r_2}, v_2), ...\}$. This binding is authenticated later in our protocol, and so no signature is needed at this stage.

We review possible attacks after introducing this modification. In Silkroad trader attack, the customer may modify *refund_to* field and put the Silk Road address as a refund address of another signer. Since the merchant will lock the refund transaction to the victim co-signer, the bitcoins cannot be claimed by the Silkroad trader since the victim co-signer does not know the Silkraod trader. The co-signer can solely claim the bitcoins through the second transaction (P2SH transaction) issued for them by the merchant. This is not what the customer or the Silkroad trader desire and so they will not plan for this attack. In Marketplace trader attack, a co-signer may intend to change the refund addresses after the payment is finalized to steal the bitcoins. In this case, he will present a new refund address for each key used in the payment transaction. Again the merchant locks the bitcoins to the main customer and the new refundee and so the bitcoins will stay in locked form since the attacker cannot obtain the signature of the main customer on that transaction. Furthermore, customer can redeem the bitcoins from the second issued transaction (P2SH transaction).

Note that,

1. A co-signer cannot change the value of refund through email. If the value is changed, the merchant will lock the bitcoins to all of the customers to ensure that they know about the change.
2. When the payment transaction is a multi-signature transaction, each of the signers who has authorized the payment transaction should introduce at least one refund address. Even if two parties agree on one refund address, both should introduce it and merchant should lock the refund to both of them through 3-of-3 multi-signature output (or n-of-n if the number of them is $n - 1$). If a signer refute to introduce a refund address, Silkroad trader attack becomes probable.

3.6 Comments on the proposed solution

If HTTP communication is used, we can use one of the following approaches to preserve the unlinkability of the payment transaction and the refund transaction: (i) encrypt refund_to field in payment message (or any sensitive information), or (2) generate a refund transaction, derive child keys of the refund addresses and then mask them, using the same algorithm introduced for anonymizing the customer address (this is for single-signature payment transactions).

For the efficiency of our scheme, we can add a flag in the payment message sent from customer to merchant to indicate whether the customer wants to have

the second transaction (the P2PKH) for redeeming the bitcoins or not. This option is to reduce the burden on merchant and it is a sensitive information that must be sent in an encrypted format. Whenever the customer uses the refund address that he knows, he can choose this option, but note that in this case the update ability of refund addresses through email should be disabled to impede the Marketplace trader attack.

Fortunately, transaction malleability is not any issue in our approach, since the proof does not depend on any information before the broadcast, instead proof is based on transactions that have been accepted in blockchain whereas they are suspicious.

4 Transaction privacy using refund mechanism

Despite using pseudonym for senders and receivers of transaction, it has been shown that transactions can be linked [2, 13, 14] and combined with other data possibly reveal user identities. There have been a number of approaches for providing anonymity [15, 9, 10, 3]. Stealth address schemes [15] guarantee address anonymity against an online attacker who intercepts the communication link and sees the Bitcoin address of the payee when it is sent to payer to create the transaction. By stealth address technique, payer adds a Diffie-Hellman key to the payee's address in a way that the corresponding private key is still known by payee. In CoinSwap [10] a party uses an intermediary node to send the payment to payee; the goal is anonymity against online attacker. In CoinJoin [9] a number of parties agree to create one transaction together, they also use values with equal worth to provide value anonymity against an offline attacker. In Fair Exchange [3] two people exchange their bitcoins with each other to achieve coins with a history that is unrelated to them, to resist against an offline attacker. Each of these solutions can be considered as a traditional mixing service. Users can also mix their coins through a mix server (e.g. bitmixer.io [5]), which receives their coins and pay them back a fresh coin, although mixer receives a fee from the user. This technique, however has a problem, user should trust the mix server that they will not steal his money.

Refund mechanism provides a level of indirection that can be used for adding privacy to transactions. We propose to use merchants as a trusted mixing servers by using a modified BIP70 protocol refund's policy. To use this service, the customer visits the website of a merchant and selects an item for purchase. By using overpayment and the recipients' addresses as the refund addresses, the sender can send payment to refundees in an anonymous way. Alternatively, they can send the desired amount to the merchant and later cancel their order for the refund. In these situations, merchant acts as an intermediary to allow the customer to pay the bitcoins to the recipients indirectly. Merchant can also split the value to smaller chunks and mix the refund transactions of different entities to provide value and time anonymity respectively. Reputable merchants are generally trusted and are expected to follow the protocol. Note that the merchant does not know if the refundee in a refund transaction is a customer

and cannot relate the output bitcoin addresses to the user. Merchants can benefit for such service by requesting a fee for it.

In our proposed protocol, the merchant receives inputs from customers' transaction, and issues transaction with outputs based on the addresses in the *refund_to* field of the payment message. Merchant generates child keys of the respective refund addresses, split the values of refund to smaller equal chunks and sends the partitioned values to child keys. Merchant considers the refund address as a parent key and uses its child keys for refund transactions to have a fresh address for each chunk. To provide confidentiality for the parent refund key (this is needed because we assume online attacker exists and the communication is HTTP), the merchant encrypts it using the Diffie-Hellman key generated by the public key of the merchant and the private key of the customer, which had been used in the payment. For secure mixing, the time relationship between the input and the output of the mix must be protected. Otherwise an adversary who intercepts the merchant's channel can link the two using the time information of the merchant input and output. In the following protocol the merchant mixes the refunds of different customers and hides the time relation between inputs and outputs of the mix service.

Key generation. Customer wallet software generates a tree of public/private key pairs based on BIP32 [16]. Assume that one of the non-hardened public keys is $Pk_c = cP$.

Click to pay. Customer visits the merchant website and chooses an item, then clicks on "pay".

Payment request. Merchant sends the payment request message including his public key, $Pk_m = mP$. This public key is unique for each customer.

Payment message. After authenticating and authorizing the merchant, customer chooses one of the non-hardened public keys, Pk_c, and generates *MainTC* that sends the cost of the chosen item to Merchant. Then, he creates a payment message with refund addresses, $Pk_{r_j}, \forall 1 \leq j \leq p$, for p refund address, and the amount of bitcoins each address should receive. Then he encrypts the *refund_to* field using Diffie-Hellman key $H(cPk_m)$.

Payment ack. Merchant detects *MainTC*, and returns an acknowledgement message to customer.

Refund request. Within a predetermined distance from payment request (can be defined in *refund_to* field according to the agreement between merchant and customer), customer can use the addresses provided in *refund_to* field to receive his refund. In this case, merchant

1. Splits the values of different customers to k partition such as v_{11}, v_{12}, and v_{1k} for v_1, and v_{21}, v_{22}, and v_{2k} for v_2 and so on.
2. Computes the Diffie-Hellman key of each customer as $H(mPk_{ci}) = H(c_iPk_m)$ for decrypting the refund addresses.
3. Derives child keys, $Pk'_{r_{li}}$ $\forall 1 \leq l \leq k$, and $\forall 1 \leq i \leq n$, for n customers.
4. Mixes chunks of n customers (v_{11}, v_{12},$v_{1k}, v_{21}, v_{22}, ..., v_{kn}$).
5. Creates and broadcasts a few transactions that pay the chunks to child keys of refund addresses.

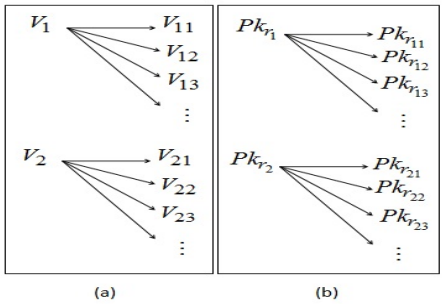

Fig. 6. (a) Splitting the value for each refundee, (b) Deriving child keys of each refund address.

4.1 Discussion

Our approach is close to CoinJoin [9] compared to other approaches. CoinJoin has two versions; in the first version users should agree and join to create one transaction. Because this co-joined transaction mixes inputs and outputs of different users, an offline attacker can not distinguish the relation between them, provided that the inputs are all in the same range. In the second version of CoinJoin, users trust in a third parties and they sends their inputs and intended outputs to them. Third party mixes the input and outputs of different users and create one transaction. Then send it back to each user to sign it. In this procedure, the third party learns the relation between input/output and IP addresses of the users. This approach cannot withstand against online attackers who intercepts the communication channel of the third party. To hide the Ip address from online attacker, Tor or VPN should be used [9].

In our scheme, Customers start a normal purchase from a merchant, and send the excess transferred amount, or alternatively the whole payment, to the refundee through merchant. First of all, it should be noted that refund request can be sent via a different communication channel such as email (by providing payment acknowledgement), so online attacker will not see the refund request. Second, an online attacker never can be assured that the customers have really made a purchase or not, or even which item they have bought, since it is possible to make overpayments to hide the exact amount of payment. Third, In CoinJoin the final transaction consisted of output users who have been introduced by the input users, meaning that anyone knows that the output address has a relation with one the inputs, and furthermore the values in the input and output can leak information about the linkage of them. However, In our scheme any relation between the inputs and outputs in the sense of time, value, and address are removed and a global passive attacker who monitors communication links and the blockchain cannot achieve any information (see Figure 7). Fourth, Merchants will not steal the bitcoins of the customers because of their reputation. This feature is necessary for any indirect payment; in CoinSwap [10], which is also an indirect payment method, first a commitment transaction is created to impede theft and then the transfer is done. But in our scheme even if merchant does not deliver

the bitcoins to refundee, customers can present their payment acknowledgement message to judge and prove the theft. Effectively, using merchant as a mixing server extends the search space for linking transactions and ensures a higher anonymity compared to CoinJoin.

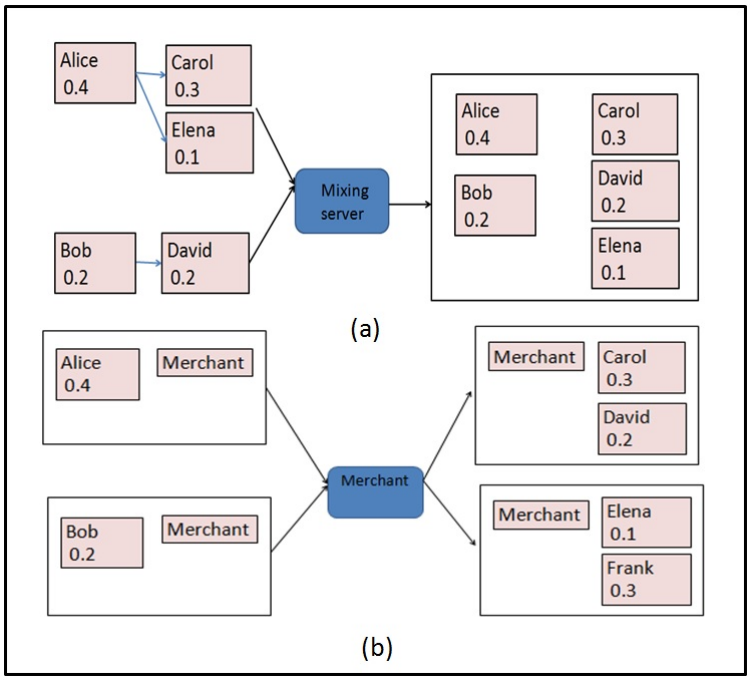

Fig. 7. (a) Coinjoin: Alice wants to send bitcoins to Carol and Elena, and Bob to David. CoinJoin mixer creates a transaction in which Alice and Bob are the payers and Carol, Elena, and David are payees. The value of each output reveals its linkage to one of the inputs. (b) Our scheme: Alice wants to pay to Elena and Carol, and Bob to David. So, Alice pays the bitcoins to merchant, and introduces Carol and Elena as refundees, through BIP70. Bob also pays the bitcoins to merchant and introduces David as refundee. Merchant creates a few refund transactions which are mixed and randomly chosen in each transaction to hide the linkage of payment and refund transactions in the sense of time, value, and address.

Comment: To have an anonymous payment protocol which can withstand refund attacks as well, we can combine the abovementioned protocols. The scheme is given in Appendix A. The channel in this scheme is assumed to be HTTPS.

5 Conclusion

In this paper, we introduced a new approach to mitigate Refund attacks against BIP70. We argued that our solution is stateless so the merchant does not need

to store any refund request or proof in the database. This protocol is also robust and no one can remove the proofs accidentally or deliberately, or tampered with them. In addition, with this payment protocol, ability of updating refund addresses through email becomes possible without the fear of marketplace trader attack. In addition, we also proposed a new application for Refund Mechanism. We argued that merchants can act as a mixing server to provide anonymity for a payer and payee. This scheme removes relation between the inputs and outputs in different transactions (payment transaction and refund transaction). Finally, we combined the mentioned schemes to have an anonymous payment protocol which is also secure against Refund attacks.

References

1. Andresen, G., Hearn, M.: Bip 70. `https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki` (07 2013), online, accessed on February 2017
2. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 34–51. Springer (2013)
3. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to betterhow to make bitcoin a better currency. In: International Conference on Financial Cryptography and Data Security. pp. 399–414. Springer (2012)
4. bitcoinwiki: Address reuse. `https://en.bitcoin.it/wiki/Address_reuse` (4 2017), online, accessed on May 2017
5. bitmixer: High volume bitcoin mixer. `https://bitmixer.io/` (2014), online, accessed on September 2017
6. BitPay: Can bitpay refund my order? `https://support.bitpay.com/hc/en-us/articles/203411523-Can-BitPay-refund-my-order-` (2015), online, accessed on February 2017
7. Coinbase: How can i refund a customer with the api? `https://support.coinbase.com/customer/en/portal/articles/1521752-how-can-i-refund-a-customer-with-the-api-` (2015), online, accessed on February 2017
8. Cuthbertson, A.: Bitcoin now accepted by 100,000 merchants worldwide. `http://www.ibtimes.co.uk/bitcoin-now-accepted-by-100000-merchants-worldwide-1486613` (2 2015), online, accessed on March 2017
9. Maxwell, G.: Coinjoin: bitcoin privacy for the real world (2013). URl: https://bitcointalk. org/index. php
10. Maxwell, G.: Coinswap: Transaction graph disjoint trustless trading. CoinSwap: Transactiongraphdisjointtrustlesstrading (October 2013) (2013)
11. McCorry, P., Shahandashti, S.F., Hao, F.: Refund attacks on bitcoins payment protocol. In: International Conference on Financial Cryptography and Data Security. pp. 581–599. Springer (2016)
12. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
13. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: Security and privacy in social networks, pp. 197–223. Springer (2013)
14. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International Conference on Financial Cryptography and Data Security. pp. 6–24. Springer (2013)

15. Todd, P.: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html (01 2014), online, accessed on March 2017
16. Wuille, P.: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (02 2017), online, accessed on February 2017

A An anonymous BIP70 secure against Refund attacks

In this section, we aggregate the schemes in previous sections to have a unique scheme which is resilient against Refund attacks [11] and an offline attacker who want to breach privacy of users.

Key generation. Customer wallet software generates a tree of public/private key pairs using BIP32 [16].

Click to pay. Customer visits the merchant website and chooses an item, then clicks on "pay".

Payment request. Merchant sends the payment request message including his public key, $Pk_m = mP$. This public key is unique for each customer.

Payment message. After authenticating and authorizing the merchant, customer chooses one of the non-hardened public keys, Pk_c, and generates a transaction, that we call it *MainTC*; this transaction sends the cost of the chosen item to Merchant. Then, he creates a payment message based on *MainTC*. In payment message customer determines a few refund addresses, $(Pk_{r_1}, Pk_{r_2}, \ldots, Pk_{r_n})$, and the amount of bitcoins each address should receive in case of order cancellation or overpayment. Then he encrypts the *refund_to* field using Diffie-Hellman key $H(cM_1)$.

Payment ack. Merchant detects *MainTC*, and returns an acknowledgement message to customer.

Refund request. Within a predetermined distance from payment request, customer can use the addresses provided in *refund_to* field to receive his refund. In this case, merchant

1. Splits the values of different customers to k partition such as v_{11}, v_{12}, and v_{1k} for v_1, and v_{21}, v_{22}, and v_{2k} for v_2 and so on.
2. Computes the Diffie-Hellman key of each customer as $H(mPk_{c_i}) = H(c_iM)$ for decrypting the refund addresses.
3. Derives child keys for each customer Pk'_{c_j}, $Pk'_{r_{il}}$, $\forall 1 \leq l \leq k$, $\forall 1 \leq j \leq k+1$.
4. Masks child keys as $Pk''_{c_j} = Pk'_{c_j} + H(c'_jM_2)P$ for $\forall 1 \leq j \leq k+1$.
5. Creates and broadcasts two transactions for each chunk/address, one that locks the bitcoins to refund addresses and the respective customer and one that can be used by the customer to claim the bitcoins.