
Article:
Modesti, Paolo, Shahandashti, Siamak F. orcid.org/0000-0002-5284-6847, McCorry, Patrick et al. (1 more author) (2021) Formal Modelling and Security Analysis of Bitcoin's Payment Protocol. Computers & Security. 102279. ISSN 0167-4048

https://doi.org/10.1016/j.cose.2021.102279

Formal Modelling and Security Analysis of Bitcoin's Payment Protocol

Paolo Modesti[a,*], Siamak F. Shahandashti[b], Patrick McCorry[c], Feng Hao[d]

[a] *Department of Computing and Games, Teesside University, UK*
[b] *Department of Computer Science, University of York, UK*
[c] *PISA Research, UK*
[d] *Department of Computer Science, University of Warwick, UK*

Abstract

The Payment Protocol standard BIP70, specifying how payments in Bitcoin are performed by merchants and customers, is supported by the largest payment processors and most widely-used wallets. The protocol has been shown to be vulnerable to refund attacks due to lack of authentication of the refund addresses. In this paper, we give the first formal model of the protocol and formalise the refund address security goals for the protocol, namely refund address authentication and secrecy. The formal model utilises communication channels as abstractions conveying security goals on which the protocol modeller and verifier can rely. We analyse the Payment Protocol confirming that it is vulnerable to an attack violating the refund address authentication security goal. Moreover, we present a concrete protocol revision proposal supporting the merchant with publicly verifiable evidence that can mitigate the attack. We verify that the revised protocol meets the security goals defined for the refund address. Hence, we demonstrate that the revised protocol is secure, not only against the existing attacks, but also against any further attacks violating the formalised security goals.

Keywords: Bitcoin, Bitccoin Security, Bitcoin Payment Protocol, Payment Security, Refund Attack, Formal Modelling, Security Analysis, OFMC, AnB

1. Introduction

Bitcoin [38], the world's most successful cryptocurrency, is a popular payment method and is currently processing on average more than 300k transactions

*Corresponding author
Email addresses: p.modesti@tees.ac.uk (Paolo Modesti), siamak.shahandashti@york.ac.uk (Siamak F. Shahandashti), feng.hao@warwick.ac.uk (Feng Hao)

This is an accepted manuscript to appear in Computers & Security. Please cite as: Modesti, Shahandashti, McCorry, and Hao. "Formal Modelling and Security Analysis of Bitcoin's Payment Protocol". To appear in Computer & Security, Elsevier, 2021.

per day. Payment Processors such as BitPay and Coinbase offer online store integration on platforms such as Shopify, OpenCart, and WordPress eCommerce for sending and receiving bitcoins. This service is popular amongst merchants willing to accept cryptocurrencies as a form of payment as it automatically converts bitcoins to fiat currency and removes the risks involved in Bitcoin's price volatility. The total Bitcoin transaction volume of such payment processors has reportedly been worth around $10M per day on average in 2019 [13].

Major Payment Processors that mediate the service between user wallets and merchants require user wallets to implement the BIP21 URI Scheme [42] and BIP70 Payment Protocol [1] Bitcoin community standards.

BIP21 provides a two-step payment procedure in which the user follows a link that triggers the user's wallet to automatically prepare a payment of the correct amount to the correct address, both embedded within the URI. BIP70 goes further and enables authenticated communication with the merchant and improved payment experience including receipt notifications, refund addresses, and payment acknowledgement. BIP70 hence improves on BIP21 in two major areas: security and usability. Since BIP21 does not provide any form of authentication, it is open to man-in-the-middle attacks. Malicious third-party scripts and extensions, viruses, or malicious Tor exit nodes have been reported to mount such attacks and change the receiver's Bitcoin address to route funds towards adversaries [46, 10]. BIP70 however, builds in the authentication of the merchant using X.509 certificates. Besides, it improves on the usability of Bitcoin payments as customers are no longer required to manually handle Bitcoin addresses, consisting of 26–35 random-looking alphanumeric characters, used to send and receive bitcoins. Instead, the customer can verify the merchant's identity using a human-readable name, coming from the certificate's 'common name' field, before authorising a payment. A refund Bitcoin address is also sent to the merchant by the user's wallet that should be used in the event of a refund. Indeed, BitPay has reported a sharp reduction of payment errors, including under- and over-payments, as a result of the adoption of the BIP70 Payment Protocol [11]. BIP70 is hence supported by major payment processors such as BitPay and Coinbase, along with popular Bitcoin development libraries such as BitcoinJ.

In a recent setback in the widespread adoption of BIP70, support for it was removed from the Bitcoin Core client, the Bitcoin reference implementation, in version 0.20.0 released in June 2020 [8]. However, the continued use of the protocol is still supported by BitPay and Coinbase wallets, many other popular (software) wallets (see e.g. [26]), and major hardware wallets such as Trezor.

From a high-level point of view, the Payment Protocol essentially provides two pieces of evidence that can be used by the parties involved to prove they have followed the protocol without malice: the *Payment Request* sent by the merchant to the customer which is digitally signed by the merchant, and the payment transaction broadcast by the customer and included in the blockchain which is digitally signed by the (pseudonymous) customer. In our previous work [32], we demonstrated that a third piece of evidence is further required: endorsement of the refund address(es) by the customer. Without this third

piece of evidence two types of attacks would be possible:

- An attack that allows a customer to request refunds to an illicit trader's Bitcoin address for a previous payment, e.g. by cancelling their order. This is called the *Silkroad Trader* attack and leverages the fact that the customer can later deny providing an illicit trader's address for refund purposes.

- An attack that allows a rogue trader to forward a *Payment Request* from an honest merchant to a customer and later request refunds from the merchant to their own Bitcoin address. This is called the *Marketplace Trader* attack and leverages the fact that refund requests are not authenticated.

Responding to the disclosure of the above attacks, payment processors tightened their refund address communication policy and to some extent mitigated the Marketplace Trader attack. The Silkroad Trader attack however, is still an open issue and addressing it would require a revision of the Payment Protocol.

We focus on the Silkroad Trader attack and extend our previous work [32] to provide the first formal model of BIP70. The Payment Protocol security analysis, which is performed using the symbolic model-checker OFMC, confirms the above attack as an Authentication Attack. Similarly, we also verify that the revised Payment Protocol in [32] fixes the vulnerability. This is the first formal model for an application of the Bitcoin blockchain and is complementary to other work in the research community which includes formal models for Nakamoto-style consensus protocols [21], formal verification of the runtime and functional correctness [7, 25, 44, 29] for smart contracts in other cryptocurrencies due to substantial thefts, and formal languages for writing Bitcoin script [40].

Contributions. This work builds on and extends our previous work (McCorry et al. [32]) focusing on the *Silkroad Trader* attack. The contributions presented are summarised below:

- We present the first formal model of the Bitcoin Payment Protocol. It utilises communication channels as abstractions conveying security goals that allow us to specify a model that is tractable and can be analysed more efficiently by the model-checker OFMC.

- We demonstrate, by model-checking, that the protocol is vulnerable to authentication attacks of the refund addresses. This attack was informally presented in [32] as the *Silkroad Trader* attack.

- We validate the revised Payment Protocol proposed in [32] and confirm that it prevents the *Silkroad Trader* attack. Moreover, we propose a simpler alternative fix, and the model-checking shows that no further attacks to the identified security goals can be performed once any of the two fixes is applied.

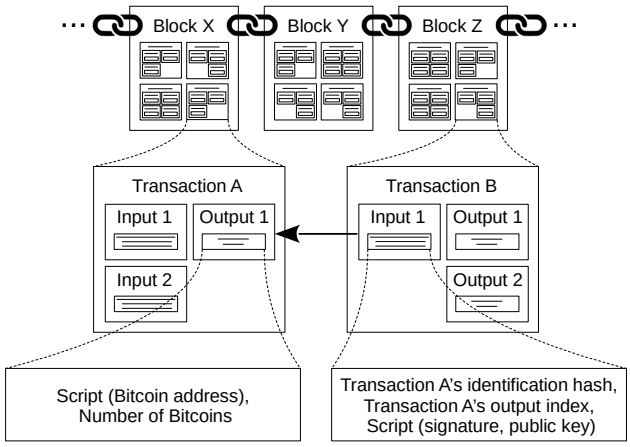

Figure 1: Information stored in the inputs and outputs of Bitcoin transactions

2. Background

We briefly discuss background information about Bitcoin, the formal modelling technique used in this paper, and related work in these areas before presenting the Payment Protocol standard in the next section.

2.1. Bitcoin

We briefly introduce three key concepts: *Bitcoin addresses*, a form of pseudonymous identification, *transactions*, a mechanism used to record the transfer of bitcoins, and the *blockchain*, a decentralised data structure storing all transactions on the network.

A *Bitcoin address* is an identifier in the Bitcoin network and is computed from the hash of an Elliptic Curve (EC) public key. An address serves as a pseudonymous identified of the user in possession of the corresponding private key. The corresponding private key can be used to claim bitcoins sent to a user and to authorise payments to other users using the Elliptic Curve Digital Signature Algorithm (ECDSA). Given the output length of the hash functions involved in the computation, the probability of collision is negligible, so these identifiers can be safely assumed unique within the network.

A *transaction* records the transfer of bitcoins. It consists of one or more inputs, specifying the source of bitcoins being spent, and one or more outputs, specifying new owner's Bitcoin address and the amount being transferred (see Figure 1). To authorise the payment, the sender must specify an input consisting of the previous transaction's identification hash and an index to one of its outputs, and provide the corresponding public key and a valid digital signature. The inputs and outputs are controlled by means of scripts in a Forth-like language specifying the conditions sufficient to claim the bitcoins. The standard script is the pay-to-pubkey-hash requiring a single signature from a Bitcoin address to authorise the payment.

The *blockchain* stores the complete transaction history of the network with a secure time-stamp [38] arranged in blocks of transactions. This append-only data structure (*ledger*) is stored in a distributed fashion by most users of the network. Appending new transactions requires solving a *proof of work* puzzle which is computationally difficult to solve but easy to verify if a solution is given. Nodes that solve proofs of work are called *miners*. They receive rewards in Bitcoin for their computational effort.

2.2. Formal Modelling Approach

Our approach to formal modelling and security analysis of the BIP70 Payment Protocol involves the symbolic model-checker OFMC [5] (version 2020), and the specification of a model in *AnB* [34], a formal language in the style of *Alice and Bob* narrations. An important reason for adopting this methodology is that in the specification of the protocol, it is possible to model communication channels as abstractions conveying security goals like authenticity and/or secrecy, without the need to specify the concrete implementation used to enforce such goals. The protocol modeller and the verification tool can then rely on the assumptions provided by such channels. This allows to specify a simpler model that is tractable by the verifier, and can be analysed more efficiently. In fact, modelling the underlying channel cryptographic implementation explicitly will lead model the checker towards a state-explosion problem and/or face out-of-memory errors.

Another important feature is that in AnB channels, agents can be identified by pseudonyms rather than by their real identities, similarly to what happens in secure channels like TLS without client authentication. Such ability to model and verify a range of different channels makes AnB suitable for the verification of payment protocols like BIP70, since in such protocols secure channels (HTTP over TLS) are used and agents use pseudonyms (e.g. ephemeral public keys).

Moreover, since we model BIP70 protocol on top of these channels, we need to discuss whether the vertical composition of such protocols is secure.

Channel as assumption. In general, OFMC allows specifying three type of channels in AnB: *authentic*, *confidential*, and *secure*, with variants that allow agents to be identified by a pseudonym rather than by a real identity. The supported standard channels are:

1. $A \rightarrow B : M$, an insecure channel from A to B, under the complete control of a Dolev-Yao intruder [16];
2. $A \bullet\!\!\rightarrow B : M$, an authentic channel from A to B, where B can rely on the fact that A has sent the message M and meant it for B;
3. $A \rightarrow\!\!\bullet\, B : M$, a confidential channel, where A can rely on the fact that only B can receive the message M;
4. $A \bullet\!\!\rightarrow\!\!\bullet\, B : M$, a secure channel (both authentic and confidential).

Pseudonymous channels [35] are similar to standard channels, with the exception that one of the secured endpoints is logically tied to a pseudonym instead of a real name. The notation $[A]_\psi$ represents that an agent A is not identified by its

real name A but by the pseudonym ψ. Usually ψ can be omitted, simplifying the notation to $[A]$, when the role uses only one pseudonym for the entire session, as it is in our case and in many other protocols.

For example, $[A] \bullet\!\!\rightarrow B : M_1$ denotes an authentic channel from A to B, where B can rely on the fact the an agent identified by a pseudonym has sent a message M_1 and this message was meant for B. If during the same protocol run, another action like $[A] \bullet\!\!\rightarrow B : M_2$ is executed, B can rely on the fact that the same agent (identified by the same pseudonym) has also sent M_2, and again the message was meant for B.

Assuming that B does not know already the real name of A, the execution of these two actions does not allow B to learn the real identity of A (unless this information is made available during the protocol execution), but B has a guarantee that he was communicating with the same agent during both message exchanges. The term *sender invariance* is used to refer to this property, and the most common example is the TLS protocol without client authentication.

Vertical Protocol Composition. Since in our model BIP70 runs on top of abstract channels providing security guarantees (such as the TLS without client authentication), we are in effect vertically composing TLS and BIP70 protocols. Strictly speaking, we should consider HTTPS, but since the security guarantees are provided by TLS, the model can simply abstract the HTTP messages.

In general, given a secure protocol P_1 that provides a certain channel type as a goal and another secure protocol P_2 that assumes this channel type, their vertical composition $P_2[P_1]$ is not secure as attacks may be possible even when the individual protocols are all secure in isolation. Sufficient conditions for vertical composition have been established [36] and, in essence, they require the disjointness of the message formats of P_1 and P_2, and that the payloads of P_2 are embedded into P_1 under a unique context to define a sharp borderline. According to Mödersheim and Viganò [36], these conditions and the other minor conditions are satisfied in practice by a large class of protocols. As the specific implementation of the underlining protocol is not part of the BIP70 specification (P_2), but P_2 only assumes that the communication occurs on channels that guarantee a secret communication with server authentication (P_1), we make our analysis under the assumption that the conditions sufficient for vertical composition specified in [36] are satisfied.

2.3. Related Work

An overview of related research on Bitcoin payment protocols and formal methods applied to Bitcoin and blockchain technologies is given below.

2.3.1. Bitcoin Payment Protocols

The Payment Protocol is designed for 'on-chain' payments in which all the transactions required for the intended payment are appended to the blockchain. Inherent and practical limitations on global transaction rates translate into serious scalability issues for Bitcoin and other cryptocurrencies. This has served as the main motivation for a line of work on 'off-chain' payment channels (See [30]

for an overview), in which payments are optimistically carried out with limited interaction with the blockchain and 'on-chain' transactions are only used to resolve party failures or to settle disputes. Recent proposals in this area include AMCU [19] for Bitcoin (and other cryptocurrencies with restricted scripting capabilities) and Sprites [33] for Ethereum. The most widely deployed of such networks are Lightning [41] for Bitcoin and Raiden (see `raiden.network`) for Ethereum. Although these alternative payment methods have a growing user base, their overall usage remains comparatively low (see e.g. statistics in [43, 9]).

Lack of methods for post-payment communications that are securely bound to the original payment transaction has been acknowledged by the community. In [31], the authors propose to bootstrap authenticated key exchange protocols between the sender and the receiver of an existing transaction leveraging the signatures recorded on the blockchain. Such a protocol will provide a secure channel between the parties to a transaction and can be used for secure post-transaction communications including arranging refunds. However, such protocols have not been deployed in practice.

An early solution to the lack of refund address endorsement by Hearn [23] suggested endorsement by any key that authorised the original payment transaction. However, this solution was shown to be prone to the 'malicious co-signer attack' [32]. Subsequent to [32], where the vulnerability of the Payment Protocol to refund attacks was demonstrated, alternative mitigation methods have been also proposed. In [4], Avizheh et al. proposed another solution based on multi-signature and time-locked transactions. The idea is that in case of a refund request, the merchant prepares two refund transactions: one that requires signatures from both the refundee and the (original) customer to be claimed (a 'multi-signature' transaction), and another that enables the customer to claim the refund after certain period of time (a 'time-locked' transaction) in case the customer does not authorise the former refund transaction. This would reduce the amount of log keeping the merchant needs to implement to protect itself against refund attacks compared to the solutions we proposed in [32]. However, Avizheh et al.'s solution would need substantial changes to the Payment Protocol standard, whereas our modifications are designed to require minimal changes to the standard.

Another noteworthy related service is the Ethereum Name Service (see `ens.domains`). This service provides a secure binding between the domain name of a merchant and their cryptocurrency addresses, supported by Ethereum smart contracts. Such a service is, in effect, akin to a distributed DNS service. A customer only needs to input the domain name of a merchant as the recipient of a payment in a wallet that supports the Ethereum Name Service look-up protocol. The wallet would be able to find the corresponding authenticated address via communication with the Ethereum blockchain. Although such services address the low usability and lack of authentication of *merchant* Bitcoin address, they do not offer any solution for customer refund address authentication.

In summary, while the Payment Protocol remains the dominant Bitcoin payment method, especially with major payment processors, *customer refund address authentication* remains an open problem in practice. Existing proposals

for secure post-payment authentication are not deployed and alternative refund address authentication mechanisms require substantial changes to the established standard. Our proposed modifications to the standard are minimal and can be readily adopted. Furthermore, all previous works addressing refund address authentication have followed the 'design-break-fix' paradigm in which solutions merely guarantee that specific attack strategies do not apply anymore. We break from this paradigm and provide a formal modelling of the required security properties along with a verification of our proposed solutions satisfying those formalised properties. Such verification provides a guarantee of security *independent of the attack strategy* for a general class of adversaries specified by their capabilities and goals, namely the Dolev-Yao attacker model [16]. Interestingly, the model is also realistic. In fact, Herzog [24] proved that there are many significant cases in which the Dolev-Yao adversary can be a valid abstraction of all realistic adversaries.

2.3.2. Formal Modelling and Verification

Given the growing interest in blockchain and cryptocurrencies, these technologies have been the subject of studies by the formal methods community as well, Bitcoin in particular. These works either focus on Bitcoin transactions, blockchain, and their security properties, or consider other components of the cyrptocurrency ecosystem such as consensus mechanisms, smart contracts, and wallets. None of these works attempt formalising any of the Bitcoin protocols built *on top of the core Bitcoin protocol* such as the Payment Protocol. However, we briefly review these works in the interest of completeness.

Among the previous works considering the formalisation of the core Bitcoin ecosystem are the following. Garay et al. [21] presented a formal modelling of the Bitcoin backbone, the protocol used at the core of Bitcoin's transaction ledger. They formalised and proved basic properties they called 'common prefix', 'chain quality', and 'chain growth', analyzing applications that can be built on top of the backbone protocol, focusing on Byzantine agreement (BA) and on the notion of a public transaction. Atzei et al. [3] proposed a formal model for Bitcoin transactions that abstractly describes their essential aspects, and at the same time enables formal reasoning. The model allows formally proving several well-formedness properties of the Bitcoin blockchain, for instance that each transaction can only be spent once. Chaudhary et al. [14] also considered the success probability of a double spending attack, which is linked to the computational power of the attacker. As the validation of Bitcoin transactions requires the successful execution of scripts, Klomp and Bracciali [27] worked on the formal verification for the Bitcoin validation framework, proposing a symbolic verification theory and a toolkit for the verification.

Other researchers have investigated different components of the cyrptocurrency ecosystem. For example, Duan and al. [17] presented the model-based formalization, simulation and verification of a blockchain protocol by using the SDL formalism of Telelogic/Rational Tau considering aspects such as security and safety of blockchain. The work also provides support for assessing different network consensus algorithms as well as on the topology of blockchain networks.

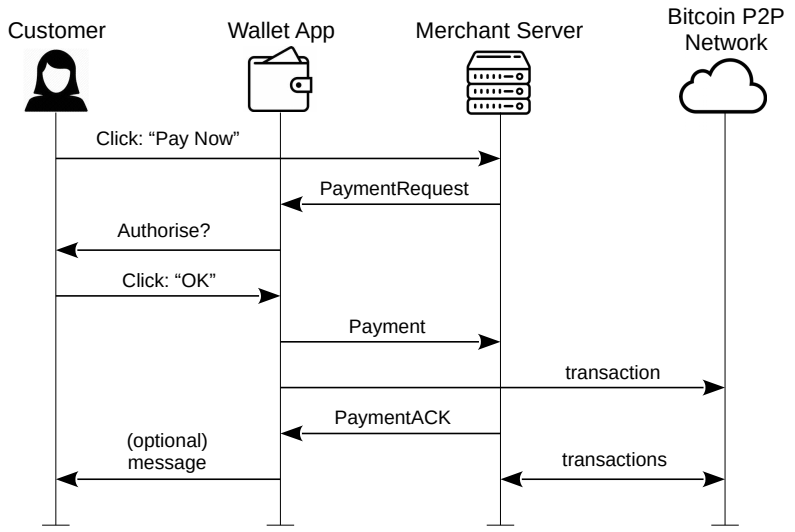

Figure 2: Overview of the Payment Protocol (adapted from [1])

Additionally, a number of works focused on formal modelling of wallets [45, 2] and smart contracts [22, 18].

In summary, no previous work has considered formal modelling and verification of protocols built on top of the core Bitcoin protocol such as the Payment Protocol. Our previous work [32] has shown that even if the core Bitcoin protocol is considered secure, vulnerabilities can exist in the design of the Payment Protocol which is built on top of the core Bitcoin protocol. It remained an open problem to formally model the amended Payment Protocol and verify that it satisfies the intended security properties assuming that the core Bitcoin protocol is secure. In this paper, we address this gap.

3. The Payment Protocol

The Payment Protocol was proposed in 2013 by Andresen and Hearn in BIP70 [1] and later adopted by the Bitcoin community as a standard. The authors present the goal of the protocol as follows:

> *“This BIP describes a protocol for communication between a merchant and their customer, enabling both a better customer experience and better security against man-in-the-middle attacks on the payment process.”*

The communication channel between the customer and merchant is strongly recommended to be over HTTPS leveraging the merchant’s X.509 certificate issued by a trusted Certificate Authority. This allows the customer to authenticate messages from the merchant.

Identifier	Description
$\mathrm{B_M}$	merchant Bitcoin address for the current transaction, a public key freshly generated by M with the corresponding private key denoted by $\mathsf{inv}(\mathrm{B_M})$
$\mathrm{B_{C_i}}$	customer C_i Bitcoin address for the current transaction, a public key freshly generated by C_i, with the corresponding private key denoted by $\mathsf{inv}(\mathrm{B_{C_i}})$
$\mathrm{R_{C_i}}$	refund address of customer C_i
฿	number of Bitcoins for the current transaction
$฿_{\mathrm{C}_i}$	number of Bitcoins to be refunded to $\mathrm{R_{C_i}}$ in case of a refund
t_1, t_2*	timestamps indicating *Payment Request* creation and expiry times, resp.
m_{M}* m_{C}*, m'_{M}*	memo messages included in the *Payment Request* (by M), *Payment* (by C), and *Payment Acknowledgement* (by M) messages
u_{M}*	payment URL
z_{M}*	payment id provided by the merchant

Table 2: Notation – Identifiers used to denote the data exchanged (optional parameters are starred)

Figure 2 outlines an overview of the messages exchanged and actions performed during the protocol execution. The protocols begins with the customer clicking on the 'Pay Now' button on the merchant's website to generate a Bitcoin payment URI. This URI allows to open the customer's Bitcoin wallet and download the *Payment Request* from the merchant's website. The wallet app can then verify the digital signature on the *Payment Request* using the public key of the merchant and the validity of the associated certificate. Given successful verification of the signature, the merchant's name in a human-readable format, extracted from the X.509 certificate's 'common name' field, and the bitcoin amount requested are shown to the customer requesting for authorisation of the payment. When the user authorises the payment, the wallet puts together a payment transaction and broadcasts it to the network. Besides, it includes the transaction and refund addresses within a *Payment* message which is sent back to the merchant. The merchant then replies to the customer wallet with a *Payment Acknowledgement* and once the payment transaction is detected on the blockchain the customer receives a confirmation of the payment.

3.1. Modelling BIP70

Our general approach to formal modelling, similarly to [20], requires the analysis of the protocol specification and its informal security requirements. In particular, in order to verify the BIP70 protocol, we build a model with n ($n \geq 1$) customers $C_1, \ldots, C_n$ and one merchant M. We assume that these

agents can trade over Bitcoin and that the identity of the merchant is known to all of them. Strictly speaking, since multiple customers might be co-operating in the payment with a single merchant, our model requires that at least one of customers knows the merchant name. We also assume that, prior the run of the protocol, the merchant does not know the identity of these customers and the communication between agents and merchants does not require a mechanism which explicitly discloses the real identity of the client. Such one-way authentication can be customarily achieved using HTTPS, as in BIP70. In this case, the client is guaranteed that messages are exchanged with the authenticated server, but the server is only guaranteed that the communication channel is shared with the same pseudonymous agent. In our case, we denote the pseudonym of the agent C_1 during the protocol run by $[C_1]$.

In the model, we also assume that C_1 is the only agent that communicates with the merchant, while other agents communicate with C_1 using a secure channel (or out-of band) to collaboratively setup an order for the merchant. This reflects the actual usage scenario of the Payment Protocol in which payment may be made from multiple psudonymous Bitcoin addresses, belonging to one or multiple actual entities, and it is the responsibility of the customer communicating with the merchant to assemble the payment transaction in coordination with all the Bitcoin address holders. The model employs two kinds of channels:

- $[C_1] \bullet\!\!\rightarrow\!\!\bullet\, M$ denotes a secure (secret and authentic) channel between the client C_1 and the merchant M, with the peculiarity that M can bind the other end point to a pseudonym $[C_1]$ rather than to the real identity of C.
- $C_i \bullet\!\!\rightarrow\!\!\bullet\, C_j$ denotes a secure channel between the clients C_i and C_j.

We use the identifiers listed in Table 2 to denote the data exchanged.

Moreover, we denote the hash function used in generating Bitcoin addresses by $\mathcal{H}$. Let us introduce the following definitions used in the protocol specification:

- $\omega_i = \mathbb{B}_{\mathrm{C}_i}, \mathcal{H}(\mathrm{B}_{\mathrm{C}_i})$: the previous transaction outputs for customer C_i;
- $\tau_{\mathrm{C}_i} = \mathsf{tr}\,(\omega_i)$: the previous transaction for customer C_i. Future transactions depend only on unspent/spendable transaction outputs; we consider here a function tr that returns a transaction parameterised on the output used by C_i in the current transaction;
- $\pi_{\mathrm{C}_i} = \mathsf{sign}_{\mathsf{inv}(\mathrm{B}_{\mathrm{C}_i})}(\mathcal{H}(\tau'_{\mathrm{C}_i}), \mathrm{B}_{\mathrm{C}_i})$: the transaction input endorsed by C_i;
- $\pi = \pi_{\mathrm{C}_1}, ..., \pi_{\mathrm{C}_n}$: the transaction input, a list transaction inputs endorsed by the customers;
- $\mathit{PaymentRequest} = \mathsf{sign}_{\mathsf{inv}(\mathsf{sk}(\mathrm{M}))}(\mathcal{H}(\mathrm{B}_\mathrm{M}), \mathbb{B}, t_1, t_2, m_\mathrm{M}, u_\mathrm{M}, z_\mathrm{M})$: the *Payment Request*, a message digitally signed with $\mathsf{inv}(\mathsf{sk}(\mathrm{M}))$, the private key of M. The associated public key utilised to verify the digital signature, that we denote as $\mathsf{sk}(\mathrm{M})$, is certified by a Certificate Authority and stored in a X.509 certificate;

- $\mathrm{RA}_{\mathrm{C}_i} = (\mathrm{R}_{\mathrm{C}_i}, \mathbb{B}_{\mathrm{C}i})$: the refund address and amount for customer C_i;
- $\tau_{\mathrm{C}} = \pi, (\mathcal{H}(\mathrm{B}_{\mathrm{M}}), \mathbb{B})$: one or more valid transactions, where π represents the inputs, and $(\mathcal{H}(\mathrm{B}_{\mathrm{M}}), \mathbb{B})$ represent the output.

3.1.1. Agents' Initial Knowledge

The initial knowledge of a model with one merchant M and two customers $\mathrm{C}_1, \mathrm{C}_2$ is as follows:

- $\mathrm{C}_1 : \mathrm{C}_1, \mathrm{C}_2, \mathrm{M}, \mathcal{H}, \mathsf{tr}, \mathsf{sk}, \mathrm{paynow}$
- $\mathrm{C}_2 : \mathrm{C}_1, \mathrm{C}_2, \mathcal{H}, \mathsf{tr}, \mathsf{sk}$
- $\mathrm{M} : \mathrm{M}, \mathcal{H}, \mathsf{tr}, \mathsf{sk}, \mathsf{inv}(\mathsf{sk}(\mathrm{M})), \mathrm{paynow}, t_1, t_2$

Each agent has an identity and access to the hash function $\mathcal{H}$, the symbolic function tr and a symbolic function sk for modelling digital signatures.

In particular, the sk function allows customers C_i to retrieve $\mathsf{sk}(\mathrm{M})$ the public key of agent M from a repository, and verify the corresponding X.509 certificate, provided that they know the name of M.

$\mathsf{inv}(\mathsf{sk}(\mathrm{M}))$ represents the private key of M and is known only by M. It should be noted that in the *AnB* language, $\mathsf{inv}()$ is a private function. Therefore, neither other agents nor the intruder can use inv to retrieve any agent's private key.

Initially, M does not know the identities C_1 and C_2, while C_1 and C_2 know each other as they need to collaborate to build the transaction. However, only C_1 knows C_2 since C_1 will be the only customer interacting with the merchant. Finally, various constants $(t_1, t_2, \mathrm{paynow})$ are available to agents according to the protocol specification.

The initial knowledge can be easily generalised for n customers; it should be noted that a customer does not need to know all other customers prior the protocol run, but at least one. As customers can coordinate as they wish (including out-of-band communication), only one customer will need to interact with the merchant.

3.1.2. Security Goals

We expect the following security goals to hold after the protocol execution:

- **Goal 1:** *Refund Addresses Authentication.* M has a guarantee that all refund addresses $\mathrm{R}_{\mathrm{C}_i}$, for all $i = 1..n$ are provided by and linked to the customers involved in the transaction; In *AnB*, we denote the goal as:

  M `weakly authenticates` C_i `on` $\mathrm{R}_{\mathrm{C}_i}, \mathbb{B}_{\mathrm{C}_i}$ (for all $i = 1..n$)

- **Goal 2:** *Refund Address Agreement and Secrecy.* All refund addresses $\mathrm{R}_{\mathrm{C}_i}$ are secret and known only by the merchant and the customers involved in the transaction. In *AnB*, we denote the goal as:

  $(\mathrm{R}_{\mathrm{C}_1}, \ldots, \mathrm{R}_{\mathrm{C}_n})$ `secret between` $\mathrm{M}, \mathrm{C}_1, \ldots, \mathrm{C}_n$

Note that the Payment Protocol is built on top of the core Bitcoin protocol and blockchain and the question we consider is whether the Payment Protocol is secure assuming the core Bitcoin protocol is secure. Therefore, we do not model the security goals that are expected to be guaranteed by the core Bitcoin protocol and blockchain, such as the double-spending prevention, and assume that they are satisfied. In fact, as discussed earlier, the security properties of the core Bitcoin protocol and blockchain have been formally proven in previous works [3, 14]. By the same token, we do not explicitly consider the security issues at the lower layers of the networking stack such as eavesdropping, prediction and fixation, since our work only concerns the application layer and assumes that protocols such as TLS are secure. The approach of considering the security properties of different layers in isolation is sound, provided that the conditions of the vertical composition theorem [36] are satisfied as discussed in Section 2.2. It should be noted that the secrecy goal (2) prevents eavesdropping, and that known prediction and fixation vulnerabilities have been addressed by more recent versions of TLS [6, 15].

3.1.3. Protocol Actions

Given their initial knowledge, agents are involved in a sequence of message exchanges over the designated channel. On the sender side, agents should have enough information to compose the message. On the recipient side, every agent must decompose the incoming messages (for example, decrypting the message or verifying a digital signature) according to their current knowledge, including knowledge acquired during previous steps. For simplicity, we assume that all public keys are available, at a certain point of the protocol execution, to the agents and the intruder.

$[\mathrm{C}_1] \bullet\!\!\rightarrow\!\!\bullet \mathrm{M}$	: paynow	C_1 *clicks* 'Pay Now'
$\mathrm{M} \bullet\!\!\rightarrow\!\!\bullet [\mathrm{C}_1]$	: *PaymentRequest*	*Payment Request*
$\mathrm{C}_1 \bullet\!\!\rightarrow\!\!\bullet \mathrm{C}_2$	: $\mathrm{R}_{\mathrm{C}_1}\mathrm{M}$, *PaymentRequest*, $\mathrm{B}_{\mathrm{C}_1}$	$\mathrm{C}_1, \mathrm{C}_2$ *cooperate -*
$\mathrm{C}_2 \bullet\!\!\rightarrow\!\!\bullet \mathrm{C}_1$	: $\mathrm{R}_{\mathrm{C}_2}, \pi_{\mathrm{C}_2}$	*- to build a transaction*
$[\mathrm{C}_1] \bullet\!\!\rightarrow\!\!\bullet \mathrm{M}$	: $z_{\mathrm{M}}, \tau_{\mathrm{C}}, \mathrm{RA}_{\mathrm{C}_1}, \mathrm{RA}_{\mathrm{C}_2}, m_{\mathrm{C}}$	*Payment*
$\mathrm{M} \bullet\!\!\rightarrow\!\!\bullet [\mathrm{C}_1]$	: $z_{\mathrm{M}}, \tau_{\mathrm{C}}, \mathrm{RA}_{\mathrm{C}_1}, \mathrm{RA}_{\mathrm{C}_2}, m_{\mathrm{C}}, m'_{\mathrm{M}}$	*PaymentACK*

3.1.4. Protocol Message Details

The Payment Protocol standard specifies the format of the *Payment Request*, *Payment*, and *Payment Acknowledgement* messages. The standard only *recommends* running the protocol over HTTPS, however in this paper we assume this is always the case. Discussed attacks apply regardless of whether HTTPS is used. Although the standard supports payment via multiple transactions, we discuss the details of the messages here for the case where the customers pays

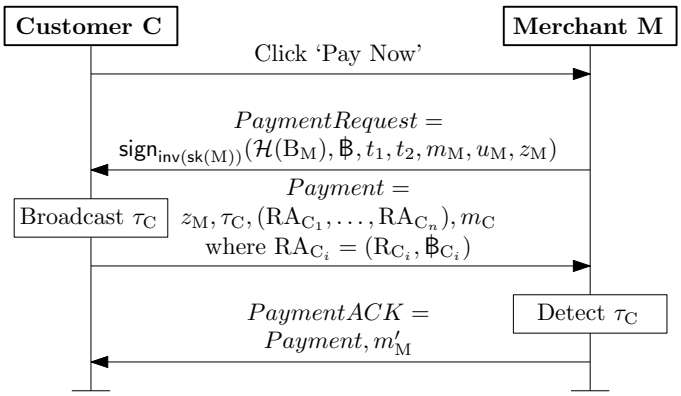

Figure 3: Expanded message contents for the Payment Protocol for C and M. Messages are sent over an HTTPS communication channel. We use the notation $\mathsf{sign}_{\mathsf{inv}(\mathsf{sk}(\mathrm{M}))}(X)$ to denote both the message X and the digital signature on the message by the private key $\mathsf{inv}(\mathsf{sk}(\mathrm{M}))$.

through only one transaction. The proposed solutions and formalisation results can be easily extended to the case where a payment is made through multiple transactions. The protocol messages are as follows:

- The *Payment Request* consists of the recipient's Bitcoin address $\mathcal{H}(\mathrm{B_M})$, requested Bitcoin amount ฿, timestamps t_1, t_2 corresponding to the request creation and expiry times, a 'memo' field m_M for a note showed to the customer, the payment URL u_M where the payment message should be sent, and an identifier z_M for the merchant to link subsequent payment messages with this request. All of the fields are collectively signed by the merchant using their private key denoted by $\mathsf{inv}(\mathsf{sk}(\mathrm{M}))$ corresponding to their X.509 certificate public key.

- The *Payment* message consists of the merchant identifier z_M, the payment transaction τ_C, a list of pairs of the form $\mathrm{RA}_{\mathrm{C}_i} = (\mathrm{R}_{\mathrm{C}_i}, ฿_{\mathrm{C}i})$ each containing the refund address $\mathrm{R}_{\mathrm{C}_i}$ and the amount to be paid to that address $฿_{\mathrm{C}i}$ in case of refund, and an optional customer 'memo' field m_C.

- The *Payment Acknowledgement* consists of a copy of the *Payment* message sent by the customer and an optional 'memo' m'_M to be shown to the customer.

The Payment Protocol messages are shown Figure 3. Note that the *Payment* message, and specially the refund addresses provided therein, are not signed by the customer, and although protected by HTTPS, they can be subsequently repudiated by the customer. This is the underlying weakness that allows the Silkroad Trader attack.

4. Verification of the Silkroad Trader Attack

We now discuss the Silkroad Trader attack, proposed in [32] and verify, by model-checking, the violation of the security goals. The Silkroad Trader attack allows a customer to route Bitcoin payments through an honest merchant to an illicit trader and later deny their involvement as we discuss below. We also demonstrate that this attack can be captured as an authentication attack within our formal model of the Payment Protocol.

4.1. Silkroad Trader Attack

As mentioned earlier, the refund addresses provided by the customer are not digitally signed. This means that a malicious customer will be able to order refunds to any arbitrary address without being required to provide any undeniable authorisation. The Silkroad Trader attack leverages such plausible deniability afforded to the customer in the Payment Protocol.

The attack sequence diagram is shown in Figure 4. Assume a Customer wishes to buy some illicit goods from a 'Silkroad Trader'. The Customer receives a *Payment Request* from the Silkroad Trader (T) that includes the Silkroad Trader's Bitcoin address $\mathcal{H}(\mathrm{B_T})$.The Customer then finds an honest Merchant supporting the Payment Protocol and selling an item of similar (or possibly just greater) price. The customer then expresses their wish to buy the item and receives a *Payment Request* from the Merchant. The Customer then puts together a *Payment* message that includes the payment transaction τ_{C}, but crucially states the Silkroad Trader's address as the refund address, i.e. $\mathrm{R_T} = \mathcal{H}(\mathrm{B_T})$, and sends the *Payment* message to the Merchant. After finalising the payment and receiving the *Payment Acknowledgement* and before the Merchant ships the item, the Customer cancels the order and requests a refund. This will prompt the Merchant to prepare and broadcast a refund transaction τ_{M} that sends the funds to the Silkroad Trader's Bitcoin address. The Customer will then be able to detect this transaction and include it in a *Payment* message she composes and sends to the Silkroad Trader. The Silkroad Trader will then detect the transaction, send the *Payment Acknowledgement* to the Customer and ship the illicit goods.

Note that since the *Payment* message sent from the Customer to the Merchant is not digitally signed, the Merchant will not be able to subsequently prove that it was indeed the Customer that requested the funds to be refunded to the Silkroad Trader's address. Hence, the Customer will be able to pay for the illicit goods through the honest Merchant without leaving any trace of a direct payment to the Silkroad Trader.

4.2. Model Checking

We encoded the model described in Section 3 in the *AnBx* language [12], an extension of the *AnB* language supported by OFMC, allowing for macro definitions, functions type signature definition and stricter type-checking. Using the *AnBx compiler* [37], we translated the model in the *AnB* format which is suitable for verification with OFMC. The source code is available in the Appendix.

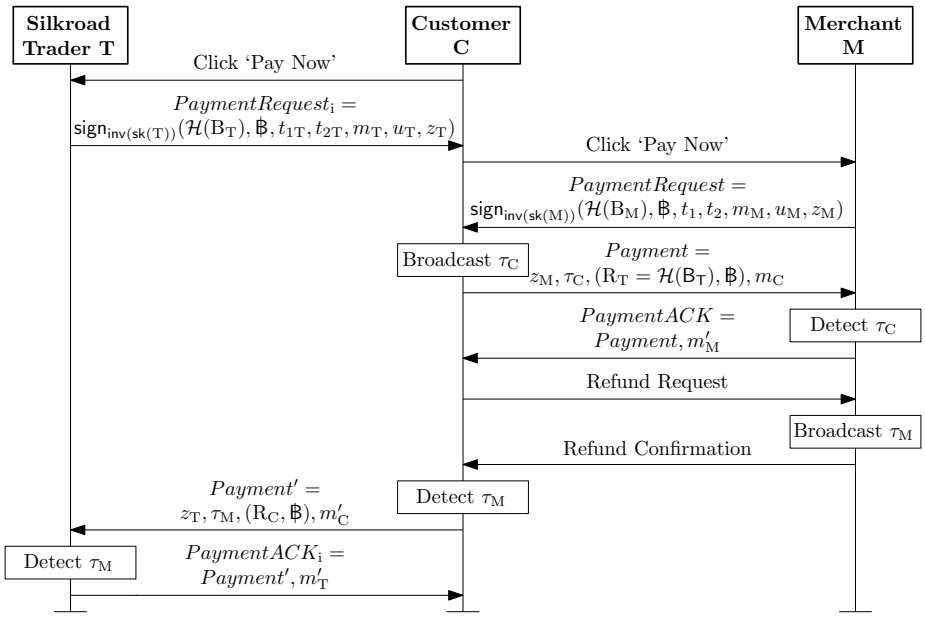

Figure 4: The *Silkroad Trader Attack* allows a customer to route bitcoins to an illicit trader via an honest merchant and then plausibly deny their involvement. This is achieved by requesting the refund to be issued to the illicit trader. Note that the customer uses the trader's address $\mathcal{H}(\mathrm{B_T})$ as the refund address within the *Payment* message sent to the merchant. All messages are sent over an HTTPS communication channel. The notation is specified in Section 3.1 and the detailed explanation of the attack can be found in Section 4.1.

We run our tests on a Windows 10 PC, with Intel Core i7 4700HQ 2.40 GHz CPU and 16 GB RAM and we verified the model for a single session in OFMC in the classic and typed mode. As a result, the model demonstrated that both authentication and secrecy goals were violated. The attack was found in 2.34 seconds.

The authentication goal M `weakly authenticates` C_i `on` $\mathrm{R}_{\mathrm{C}_i}, \mathbb{B}_{\mathrm{C}_i}$ states for all customers ($i = 1..n$), the merchant can have a guarantee of endorsment of the refund addresses and amounts.

In particular, the goal is violated because in not possible to verify the non-injective agreement [28] between the construction of $\mathrm{RA}_{\mathrm{C}_i} = (\mathrm{R}_{\mathrm{C}_i}, \mathbb{B}_{\mathrm{C}_i})$ done by C_i and the corresponding values received by M. This is possible because the customers are not required to endorse the value $(\mathrm{R}_{\mathrm{C}_i}, \mathbb{B}_{\mathrm{C}_i})$ using digital signatures. Therefore a compromised or dishonest client can easily manipulate the refund address and perform attacks like the one described in 4.1.

The secrecy goal $(\mathrm{R}_{\mathrm{C}_1}, \ldots, \mathrm{R}_{\mathrm{C}_n})$ `secret between` $\mathrm{M}, \mathrm{C}_1, \ldots, \mathrm{C}_n$ is also violated. The definition of secrecy used in our model implies that all members of the secrecy set know the secret values and agree on these. But in this case, due to lack of authentication, the customer who is communicating with the merchant

can convince other customers that the refund address she is using is different from the one sent to the merchant. For example, R_{C_2}, the refund address of the second customer, can be easily replaced with a different address by C_1 before being communicated to M.

It should be noted that, in general, with the automated verification it is not possible to validate a specific attack trace known a priori, and the analysis usually aims at assessing the absence or presence of at least an attack trace that leads at a violation of a security goal. In particular, in order to verify the protocol, the model-checker OFMC builds a state-transition system, and given the initial configuration, analyses the possible transitions in order to see if any attack state is reachable in presence of an active attacker. Therefore, the presence of a specific attack trace is not automatically confirmed, rather, such automated versification helps decide whether any attack trace is present or absent, where an attack trace is defined as a sequence of steps leading to a violation of a given security goal. However, the absence of any attack trace is the ultimate goal of verification, to state the security of the analysed protocol, and we will see how this can be achieved in Section 5.

4.3. Real-World Experiments

Our experiments, originally reported in [32], aimed to verify the practice of processing refunds by merchants, and assess the feasibility of the attacks. We purchased items from real-world merchants using a modified Bitcoin wallet before requesting for the order to be cancelled and a refund processed. The merchants used during these experiments are based in the UK and are supported by BitPay or Coinbase. The bitcoins used for the experiments are owned by the authors and no money is sent to any illicit trader. All experiments were ethically approved by the relevant research ethics committee.

4.3.1. Proof of Concept Wallet

We developed a wallet supporting the Payment Protocol and automating the *Silkroad Trader* attack. Our wallet works as follows:

1. The customer inserts the illicit trader's *Payment Request* URI into the wallet which stores both the request and Bitcoin address for later use.
2. The customer finds an item equal (or greater) in value as the 'illicit goods' and inserts the merchant's *Payment Request* URI into their wallet.
3. The wallet provides a list of refund addresses that can be chosen for the *Payment* message that is sent to the merchant and the customer can choose the illicit trader's Bitcoin address.
4. Assuming a refund has been authorised by the merchant, the wallet can detect the merchant's refund transaction on the network and include it in a *Payment* message that is sent to the illicit trader.
5. The wallet is notified by a *Payment Acknowledgement* message from the illicit trader that the payment has been received.

4.3.2. Simulation of the Attack

We discuss the results of carrying out a simulation of the Silkroad Trader attack against real-world merchants using arbitrary identities, e.g. names and e-mail addresses, created for the experiments only.

Cex refunded the bitcoins within 3 hours of cancelling the order and used the refund address from the Payment Protocol.

Pimoroni Ltd. refunded the bitcoins within a single business day and used the refund address from the Payment Protocol.

Scan refunded the bitcoins after 26 days and used the refund address from the Payment Protocol. The delay was due to Scan initially requesting us to provide a refund address over e-mail, but we insisted using the one specified in the original payment message.

Dell were unable to process the refund due to 'technical difficulties' and requested our bank details. We informed them that we did not own a bank account and Dell suggested sending the refund as a cheque. While not the experiment's aim, this potentially opens Dell as an exchange for laundering tainted bitcoins.

4.3.3. Payment Processors' Responses

We privately disclosed our attacks to the Payment Processors and received the following responses:

BitPay acknowledged "the researchers have done their homework" and that "refunds are definitely a significant money laundering attack vector". They are now actively monitoring for refund activity on behalf of their merchants.

Coinbase acknowledged the *Silkroad Trader* attack as a good example of an authentication vulnerability in the Payment Protocol.

Bitt acknowledged both attacks and believe the endorsement evidence may support Payment Processors to become more 'airtight' for future regulation.

In response to our disclosures, the payment processors have put in place temporary mitigation measures such as monitoring refunds. These measures only partially address the *Silkroad Trader* attack. To fully address the vulnerability, the BIP70 standard would need to be revised, as we have discuss in Section 5.

5. Verification of the Proposed Solutions

In our previous work (McCorry et al. [32]), we proposed a solution that requires the refund addresses in the *Payment* message to be endorsed by the customer. This provides the merchant with a proof of endorsement that can be used to demonstrate to a third party that the customer who authorised the payment also endorsed the refund addresses. We explain the solution first and then verify, by model-checking, that the solution meets the security goals set out in Section 3.1.2. We also present an alternative solution meeting the security goals and discuss the comparative merits of each solution.

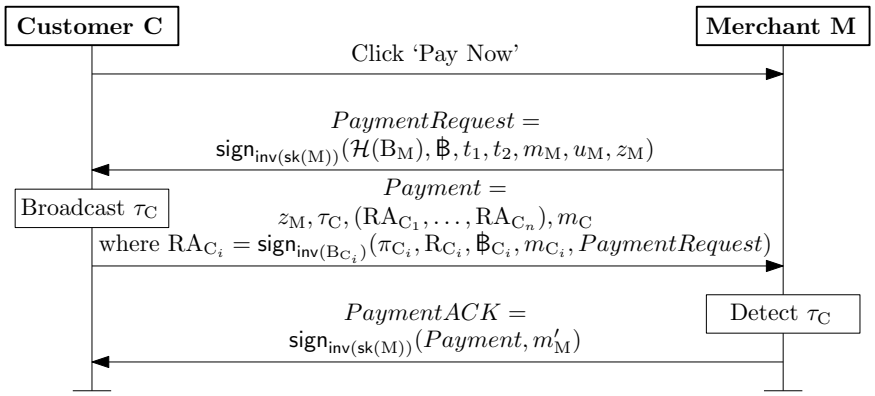

Figure 5: The Original Solution: expanded message contents for the amended Payment Protocol for Customer C and Merchant M. The Customer endorses the refund addresses by providing digital signatures $\mathrm{RA}_{\mathrm{C}_i}$. The endorsement can be used by the Merchant to prove that they acted as per the instructions of the Customer in case of a refund. The protocol is explained in detail in Section 5.1.

5.1. The Original Solution

Figure 5 shows the amended Payment Protocol as proposed in our earlier work [32]. The *Payment Request* is similar to before, except that this solution requires the memo m_M to include specific information to assure the customer that the message is in response to their 'click'. This is to mitigate the Marketplace Trader attack and does not change the formal model we propose in this work. The *Payment* message includes all the elements specified in the Payment Protocol plus a digital signature to endorse each refund address. More specifically, for each refund address $\mathrm{R}_{\mathrm{C}_i}$ and refund amount ฿_{C_i}, the customer provides a digital signature on $(R_{\mathrm{C}_i}, \text{฿}_{\mathrm{C}_i}, m_{\mathrm{C}_i}, PaymentRequest)$ as a proof of endorsement of the refund address and amount that binds these values to the corresponding payment transaction input and the specific *PaymentRequest*. The *Payment Acknowledgement* contains a copy of the *Payment* message and a memo as before, plus a digital signature by the merchant using the private key corresponding to their X.509 certificate public key.

More specifically, each refund address endorsement signature is in the form:

$$\mathrm{RA}_{\mathrm{C}_i} = \mathsf{sign}_{\mathsf{inv}(\mathrm{B}_{\mathrm{C}_i})}(\pi_{\mathrm{C}_i}, \mathrm{R}_{\mathrm{C}_i}, \text{฿}_{\mathrm{C}_i}, m_{\mathrm{C}_i}, PaymentRequest),$$

where $\mathsf{sign}()$ is the signature algorithm and $\mathsf{inv}(\mathrm{B}_{\mathrm{C}_i})$ is the private key that authorised the transaction input π_{C_i}. These parameters were chosen to clarify the correspondence between the transaction inputs and the endorsed refund addresses and to ensure the endorsement is only valid for a specific *Payment Request*. Moreover, the proposed solution suggests that the Merchant should digitally sign the *Payment Acknowledgement* as follows:

$$PaymentACK = \mathsf{sign}_{\mathsf{inv}(\mathsf{sk}(\mathrm{M}))}(Payment, m'_\mathrm{M}),$$

such that the customers can have evidence of the completion of the protocol.

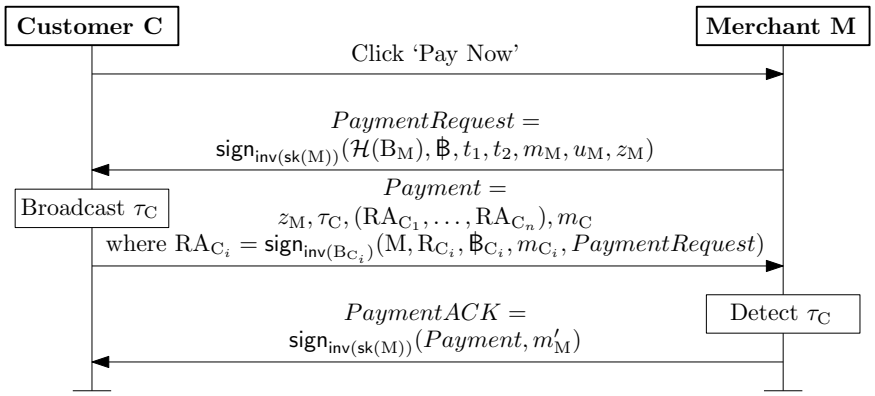

Figure 6: Alternative solution: expanded message contents for the amended Payment Protocol for Customer C and Merchant M. The Customer endorsement $\mathrm{RA}_{\mathrm{C}_i}$ does not require the transaction π_{C_i} and instead includes the identity of the Merchant M. The protocol is explained in detail in Section 5.2.

Verification. We updated the model described in Section 3 with the new specification, ran the tests and verified the model for a single session in OFMC in the classic and typed mode. With such amendments, we verified in 10.08 seconds that there are no attacks on the security goals for the one-session verification. We also tested the model for two parallel sessions, as long as the available RAM (32 GB) allowed for it. We did not find further attacks.

The fix works because now the refund address and the amount are digitally signed by the private key that authorised the transaction input π_{C_i}. It is now possible to prove the non-injective agreement, i.e. the weak authentication goal. Since the refund address is endorsed and the merchant receives evidence of endorsement of the refund addresses, it is now impossible for a compromised or dishonest customer to manipulate these values.

We furthermore tested the stronger authentication goal M `authenticates` C_i `on` $\mathrm{R}_{\mathrm{C}_i}, ฿_{\mathrm{C}_i}$ (for all $i = 1..n$) and we found it also holds. This in practice gives the recipients of the signed message evidence of freshness, provided the message contains sufficient information to distinguish between different transactions, which is usually the case.

The secrecy goal is also satisfied because now all the customers and the merchant agree on the refund addresses. It is not possible for an attacker to manipulate the refund addresses since the digital signatures would not verify in case of manipulation.

In conclusion, we verified that the proposed solution fixes the protocol in that it successfully prevents attacks on the authentication and secrecy goals.

5.2. An Alternative Solution

We also propose an alternative solution, depicted in Figure 6, where now the following definition is used:

$$\mathrm{RA}_{\mathrm{C}_i} = \mathsf{sign}_{\mathsf{inv}(\mathsf{sk}(\mathrm{B}_{\mathrm{C}_i}))}(\mathrm{M}, \mathrm{R}_{\mathrm{C}_i}, ฿_{\mathrm{C}i}, m_{\mathrm{C}_i}, \mathit{PaymentRequest}).$$

In this solution, instead of including π_{C_i}, which introduces a dependency on previous transactions, we simply add the identity of the merchant M, which would be immediately available, allowing τ_{C_i} to be computed later. Similarly to the previous solution, all goals, including strong authentication, are successfully verified, in this case significantly faster (5.30 sec).

It is important to note that the authentication goal is met as well because the identity M is included in the signed messages and this provides a clear evidence of the customers' intention to run the Payment Protocol with the explicitly identified merchant M. Specifically, from the theoretical point of view, this satisfies the definition of the notion of *'agreement'* [28] usually considered in formal methods. This notion is stronger than the kind of guarantee that can be achieved with a digital signature if the recipient identity is omitted in the payload (i.e. what is called *'proof of endorsement'*). In practice, binding the refund details to the merchant identity provides an extra guarantee (compared to [32]) that the endorsed refund details can not be reused with merchants other than M.

5.3. Discussion

The main idea behind the proposed solutions is to augment the protocol with a *proof of endorsement* of the refund addresses by the keys that authorise the original transaction. The merchant may store this proof of endorsement and demonstrate their appropriate conduct to a third party in case of a *Silkroad Trader* attack. The customer colluding with an illicit trader on the other hand will lose their ability to plausibly deny their involvement in such an attack. Note that the proof of endorsement only links pseudonymous Bitcoin addresses. This is the same as what the merchant would learn from the original protocol messages.

It should be pointed out that the proposed extension BIP75 [39], aims at providing the merchant with a publicly verifiable audit log of all transactions. The solutions considered in this paper also aim to provide the merchant with this audit log but without the need to know the customer's real-world identity. In our approach, customers do not need to maintain certificates to spend their bitcoins. Instead, we propose using the same keys that authorised the Bitcoin transaction to provide the merchant with publicly verifiable evidence. This evidence states that the same pseudonymous customer that authorised the payment has agreed to the terms of the purchase, and authenticates any new instructions provided by the pseudonymous customer. Every new payment authenticates a new pseudonymous customer and the merchant does not necessarily need to know their identity, but just cares that it is dealing with the correct pseudonymous customer for each payment.

5.4. Solution Performance

The computational overhead of the solution is quite low in general. On the customer side, the wallet already needs to carry out several cryptographic operations for participating in the original Payment Protocol, including verifying

Customer in the current protocol

1	Verify authenticity of merchant's certificate chain	0.83 ms
2	Verify merchant's signature on Payment Request	0.08 ms
3	Sign a single transaction input	0.08 ms
4a	Fetch list of pre-generated refund addresses $R_{C_1}, ..., R_{C_n}$	0.72 ms
4b	Generate new refund address R_C from wallet key pool	110.55 ms
5	Update wallet address book with refund address R_C	72.68 ms
	Total without 4b:	*74.39 ms*
	Total with 4b:	*184.94 ms*

Additional changes proposed for the customer

6	Compute endorsement signature RA_C	0.11 ms
	New Total without 4b:	*74.49 ms*
	New Total with 4b:	*185.04 ms*

Merchant in the current protocol

7	Verify customer's payment transaction	0.29 ms
	Total:	*0.29 ms*

Additional changes proposed for the merchant

8	Fetch referenced transaction output	0.01 ms
9	Verify endorsement signature RA_C	0.13 ms
	New Total:	*0.43 ms*

Table 3: Time performance for proposed changes to the Payment Protocol

the merchant's certificate chain and signature, signing the payment transaction, and generating refund keys. Hence, an extra signature (per input) introduces minimal overhead on the customer side. On the merchant side, the overhead is more substantial, however it is limited to the verification of an extra signature and the production of an extra signature for the *Payment Acknowledgement*.

We have implemented the modifications to the Payment Protocol and measured the performance of all the steps of both the original and the amended Payment Protocols. All tests were carried out on a MacBook Pro running OS X 10.9.1 with 2.3 GHz Intel Core i7 and 16 GB DDR3 RAM. The details of the measured timing performance can be found in Table 3. The reported measurements are for the Bitcoin Core Client while utilising 1 core. Furthermore, both signing operations in steps 3 and 8, and the verification operation in step 9, are performed using the Secp256k1 implementation which has replaced OpenSSL in Bitcoin Core [47]. Each step was executed 100 times and the reported times represent the average.

Steps 1–5 represent the customer's perspective in the current Payment Protocol's implementation. The wallet verifies the merchant's certificate authenticity using the chain of certificates that lead to a trusted root authority and verifies the merchant's signature on the *Payment Request* message before authorising at least one transaction input to authorise the payment. Then, the wallet fetches a list of pre-generated refund addresses and Step 4b only occurs if this list is empty

as a new refund address must be generated. This refund address is associated with the payment for future reference. These steps require 74.39 ms if the list of pre-generated refund addresses is not empty, otherwise 184.94 ms is required. Our proposed change in Step 6 takes 0.11 ms and requires the customer's wallet to sign an endorsement message for the refund address, obtaining the signature σ_{C}. In total, the time required for the customer is 185.04 ms with Step 4b, and 74.49 ms without Step 4b.

Step 7 represents the merchant's perspective in the current Payment Protocol's implementation and requires 0.29 ms to check if the payment transaction with a single input is valid. We propose in Steps 8–9 that the merchant fetches the transaction output referenced in the payment transaction's input to let the merchant check the number of bitcoins associated with each refund address. Then, the transaction input's public key C is used to verify the endorsement signature. These proposed changes require 0.14 ms, and in total the time required for the merchant is 0.43 ms.

6. Conclusion

In this work, we considered Bitcoin's Payment Protocol and its vulnerability to an empirically demonstrated attack that leverages the lack of refund address authentication in the protocol to allow malicious customers to route funds through honest merchants to illicit traders and be able to later deny doing so. We formally modelled the protocol, proved it is vulnerable and validated the solution proposed previously to fix the protocol. We also presented and verified a new alternative solution which is simpler and can have, in principle, a reduced computational impact than the previous one. In both cases, the solutions provide the merchant with evidence that the refund address received during the Payment Protocol has been digitally signed from the same pseudonymous customer who authorised the transaction.

To the best of our knowledge, our model of the Bitcoin Payment Protocol, which complements previous work focusing on underlying core aspects of blockchain and Bitcoin, is the first attempt to formally model and analyse the security of a protocol relaying on Bitcoin. It is worth noting that to complete the analysis we employed specific notions available in OFMC, such as pseudonyms channels and channel abstractions conveying security goals, that allowed us to specify a model that is tractable and can be analysed more efficiently.

Acknowledgements

The second and forth authors were supported by the European Research Council (ERC) Starting Grant (No. 306994).

References

[1] G. Andresen and M. Hearn. BIP 70: Payment Protocol. *Bitcoin Improvement Process*, July 2013. `https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki`.

[2] M. Arapinis, A. Gkaniatsou, D. Karakostas, and A. Kiayias. A formal treatment of hardware wallets. In Goldberg and Moore, editors, *Financial Cryptography and Data Security (FC 2019)*, volume 11598 of *LNCS*, pages 426–445. Springer, 2019.

[3] N. Atzei, M. Bartoletti, S. Lande, and R. Zunino. A formal model of bitcoin transactions. In Meiklejohn and Sako, editors, *Financial Cryptography and Data Security (FC 2018)*, volume 10957 of *LNCS*, pages 541–560. Springer, 2018.

[4] S. Avizheh, R. Safavi-Naini, and S. F. Shahandashti. A New Look at the Refund Mechanism in the Bitcoin Payment Protocol. In *Financial Cryptography*, volume 10957 of *Lecture Notes in Computer Science*, pages 369–387. Springer, 2018.

[5] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[6] K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 483–502. IEEE Computer Society, 2017.

[7] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Beguelin. Formal verification of smart contracts. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security PLAS16*, pages 91–96, 2016.

[8] Bitcoin Core Team. Bitcoin Core release notes 0.20.0, 2020. `https://github.com/bitcoin/bitcoin/blob/master/doc/release-notes/release-notes-0.20.0.md/`.

[9] BitMEX Research. Lightning network (part 6) - over 60,000 non-cooperative channel closures, 2020. `https://blog.bitmex.com/lightning-network-part-6-over-60000-non-cooperative-channel-closures`.

[10] BitPay. Avoiding and detecting a new virus affecting some bitcoin users, 2016. `https://bitpay.com/blog/avoiding-and-detecting-a-new-virus-affecting-some-bitcoin-users/`.

[11] BitPay. How payment protocol is eliminating costly bitcoin payment errors: Stats and results, 2018. `https://bitpay.com/blog/payment-protocol-results`.

[12] M. Bugliesi, S. Calzavara, S. Mödersheim, and P. Modesti. Security protocol specification and verification with AnBx. *Journal of Information Security and Applications*, 30:46–63, 2016.

[13] Chainalysis Team. Darknet market activity higher than ever in 2019 despite closures. how does law enforcement respond?, 2020. https://blog.chainalysis.com/reports/darknet-markets-cryptocurrency-2019.

[14] K. Chaudhary, A. Fehnker, J. van de Pol, and M. Stoelinga. Modeling and verification of the bitcoin protocol. In van Glabbeek, Groote, and Höfner, editors, *Proceedings Workshop on Models for Formal Analysis of Real Systems, MARS 2015*, volume 196 of *EPTCS*, pages 46–60, 2015.

[15] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)*, pages 1773–1788. ACM, 2017.

[16] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on information Theory*, 2(29), 1983.

[17] Z. Duan, H. Mao, Z. Chen, X. Bai, K. Hu, and J.-P. Talpin. Formal modeling and verification of blockchain system. In Meiklejohn and Sako, editors, *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, volume 10957 of *LNCS*, pages 231–235. Springer, 2018.

[18] V. Dwivedi, V. Deval, A. Dixit, and A. Norta. Formal-verification of smart-contract languages: A survey. In *International Conference on Advances in Computing and Data Sciences*, pages 738–747. Springer, 2019.

[19] C. Egger, P. Moreno-Sanchez, and M. Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *Computer and Communications Security (CCS)*, pages 801–815. ACM, 2019.

[20] L. Freitas, P. Modesti, and M. Emms. A Methodology for Protocol Verification applied to EMV1. In *Formal Methods: Foundations and Applications - 21th Brazilian Symposium, SBMF 2018, Proceedings*, volume 11254 of *LNCS*. Springer, 2018.

[21] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310, 2015.

[22] D. Harz and W. J. Knottenbelt. Towards safer smart contracts: A survey of languages and verification methods. *CoRR*, abs/1809.09805, 2018.

[23] M. Hearn. Re: [Bitcoin-development] BIP 70 refund field. *Bitcoin-Development*, Mar. 2014. http://sourceforge.net/p/bitcoin/mailman/message/32157661/.

[24] J. Herzog. A computational interpretation of dolev-yao adversaries. *Theor. Comput. Sci.*, 340(1):57–81, 2005.

[25] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. M. Moore, D. Park, Y. Zhang, A. Stefanescu, and G. Rosu. KEVM: A complete formal semantics of the ethereum virtual machine. In *31st IEEE Computer Security Foundations Symposium, CSF 2018*, pages 204–217. IEEE Computer Society, 2018.

[26] Keaton (BitPay Support). Which wallets work best for a bitpay payment? Which wallets are compatible?, June 2020. `https://support.bitpay.com/hc/en-us/articles/115005701523`.

[27] R. Klomp and A. Bracciali. On symbolic verification of bitcoin's script language. In García-Alfaro, Herrera-Joancomartí, Livraga, and Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Proceedings*, volume 11025 of *LNCS*, pages 38–56. Springer, 2018.

[28] G. Lowe. A hierarchy of authentication specifications. In *CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.

[29] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.

[30] P. McCorry, M. Möser, S. F. Shahandashti, and F. Hao. Towards bitcoin payment networks. In *ACISP (1)*, volume 9722 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2016.

[31] P. McCorry, S. F. Shahandashti, D. Clarke, and F. Hao. Authenticated Key Exchange over Bitcoin. In *Security Standardisation Research*, volume 9497 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.

[32] P. McCorry, S. F. Shahandashti, and F. Hao. Refund attacks on bitcoin‘s payment protocol. In *20th Financial Cryptography and Data Security conference*, 2016.

[33] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Financial Cryptography and Data Security*, pages 508–526. Springer, 2019.

[34] S. Mödersheim. Algebraic properties in Alice and Bob notation. In *International Conference on Availability, Reliability and Security (ARES 2009)*, pages 433–440, 2009.

[35] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In *Computer Security–ESORICS 2009: 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009, Proceedings*, page 337. Springer, 2009.

[36] S. Mödersheim and L. Viganò. Sufficient conditions for vertical composition of security protocols. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 435–446, New York, NY, USA, 2014. ACM.

[37] P. Modesti. AnBx: Automatic generation and verification of security protocols implementations. In *8th International Symposium on Foundations & Practice of Security*, volume 9482 of *LNCS*. Springer, 2015.

[38] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, November 2008. `https://bitcoin.org/bitcoin.pdf`.

[39] J. Newton, M. David, A. Voisine, and M. James. BIP 75: Out of Band Address Exchange using Payment Protocol Encryption. *Bitcoin Improvement Process*, Nov. 2015. `https://github.com/bitcoin/bips/blob/master/bip-0075.mediawiki`.

[40] R. O'Connor. Simplicity: A new language for blockchains. In *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2017*, pages 107–120. ACM, 2017.

[41] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. `https://lightning.network/lightningnetwork-paper.pdf`.

[42] N. Schneider and M. Corallo. BIP 21: URI Scheme. *Bitcoin Improvement Process*, 2012. `https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki`.

[43] K. Torpey. Is bitcoin's lightning network ready to replace altcoin use cases?, 2020. `https://cryptonews.com/exclusives/is-bitcoin-s-lightning-network-ready-to-replace-altcoin-use-6511.htm`.

[44] P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. T. Vechev. Securify: Practical security analysis of smart contracts. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 67–82. ACM, 2018.

[45] M. Turuani, T. Voegtlin, and M. Rusinowitch. Automated verification of electrum wallet. In Clark, Meiklejohn, Ryan, Wallach, Brenner, and Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC*, volume 9604 of *LNCS*, pages 27–42. Springer, 2016.

[46] W. Wei. Malicious chrome extension hijacks cryptocurrencies and wallets, 2014. `https://thehackernews.com/2014/04/malicious-chrome-extension-hijacks.html`.

[47] P. Wuille. Switch to libsecp256k1-based ECDSA validation. *Bitcoin Github Repository*, Nov. 2015. https://github.com/bitcoin/bitcoin/pull/6954.

Appendix

```
Protocol: BIP70 AnB

Types:

    Agent C1,C2,M;
    Number paynow,BTC,BTC1,BTC2,t1,t2,MM,UM,ZM,MM1,RC1,RC2;
    PublicKey BAM,BC1,BC2;
    Function [Agent -> PublicKey] sk;
    Function [Number,Number -> Number] tr;
    Function [Untyped -> Number] hash

Definitions:

    # Bitcoin transaction outputs (previous transaction)
    OutC1: BTC1,hash(BC1);
    OutC2: BTC2,hash(BC2);

    # we ignore inputs as they are not relevant for the next transaction
    TauC1: tr(OutC1);
    TauC2: tr(OutC2);

    # Bitcoin transaction inputs
    PiC1 : {hash(TauC1)}inv(BC1),BC1;
    PiC2 : {hash(TauC2)}inv(BC2),BC2;

    # Bitcoin transaction inputs
    Pi: PiC1,PiC2;

    # Payment request from merchant
    PaymentRequest:{hash(BAM),BTC,t1,t2,MM,UM,ZM}inv(sk(M));

    # SilkRoad attack
    # merchant does not have a guarantee that
    # refund addresses RCx are authorised by BCx

    # <-- UNCOMMENT THESE LINES TO REPRODUCE THE ATTACK
    # and COMMENT the other definition of RACx
    # Refund address authorization (original specification)
    # RAC1: RC1,BTC1;
    # RAC2: RC2,BTC2;

    # McCorry et al. solution: the payment request
    # is needed to link the refund request to the transaction
    # RAC1: {PiC1,RC1,BTC1,PaymentRequest}inv(BC1);
    # RAC2: {PiC2,RC2,BTC2,PaymentRequest}inv(BC2);

    # alternative solution
    RAC1: {M,RC1,BTC1,PaymentRequest}inv(BC1);
    RAC2: {M,RC2,BTC2,PaymentRequest}inv(BC2);
```

```
    TauC: Pi,(BTC,hash(BAM))

    # The fix includes a message digitally signed
    # with the session private key of C1 or C2 - inv(BCx)
    # The signed message includes the identities of M and the
    # Refund addresses RCx, the bitcoin amount BTCx, PiCx
    # Note that C1,M are required to achieve the weak authentication goal.
    # In practice C1, M may be an unique identifier on which
    # the parties agree, a public key for example
```

Knowledge:

```
    # initial knowledge of agents
    # M does not know C1,C2 - C1,C2 know each other - C2 does not know M
    C2: C2,C1,sk,hash,tr;
    C1: C1,M,C2,sk,hash,paynow,tr;
    M: M,sk,inv(sk(M)),hash,paynow,t1,t2,tr
```

Actions:

```
    # the previous transactions are exchanged between C1,C2
    # made public so the intruder can learn them
    # this is a fictional exchange as this information
    # is already public on the blockchain
    C1 -> C2: TauC1
    C2 -> C1: TauC2

    # starting payment protocol
    [C1] *->* M: paynow

    # payment request signed by the merchant
    M *->* [C1]: PaymentRequest

    # C1,C2 cooperate to construct a transaction
    # (not part of payment protocol)
    C1 *->* C2: RAC1,M,PaymentRequest,BC1
    C2 *->* C1: RAC2,PiC2

    # Payment
    [C1] *->* M: ZM,TauC,RAC1,RAC2

    # Payment Acknowledgement
    # original specification
    # M *->* [C1]: ZM,TauC,RAC1,RAC2,MM1

    # McCorry et al. and alternative solution
    M *->* [C1]: {ZM,TauC,RAC1,RAC2,MM1}inv(sk(M))

    # end of the payment protocol

    # agents names are revealed to M
    [C1] *->* M: C1,C2

    # TauC is made public (broadcasted) and learned by the intruder
    M -> C1: TauC
```

```
    # Public Keys are made available
    C1 -> M: BC1,BC2,TauC1,TauC2

    M -> C1: BAM
    C1 -> C2: BAM

Goals:

    # M has a guarantee that RCx,BTCx are provided by and linked to Cx
    M weakly authenticates C1 on RC1,BTC1
    M weakly authenticates C2 on RC2,BTC2

    # RCx are confidential
    RC1,RC2 secret between M,C1,C2

    # other goals
    M weakly authenticates [C1] on RC1,BTC1,RC2,BTC2
    M weakly authenticates [C1] on RC1,BTC1
    M weakly authenticates [C1] on BC1
```