Bolt: Anonymous Payment Channels for Decentralized Currencies

Matthew Green Ian Miers

Information Security Institute
The Johns Hopkins University
3400 N. Charles St.
Baltimore, MD 21218
{mgreen,imiers}@cs.jhu.edu

Abstract

Bitcoin owes its success to the fact that transactions are transparently recorded in the blockchain, a global public ledger that removes the need for trusted parties. Unfortunately, recording every transaction in the blockchain causes privacy, latency, and scalability issues. Building on recent proposals for "micropayment channels" — two party associations that use the ledger only for dispute resolution — we introduce techniques for constructing *anonymous* payment channels. Our proposals allow for secure, instantaneous and private payments that substantially reduce the storage burden on the payment network. Specifically, we introduce three channel proposals, including a technique that allows payments via untrusted intermediaries. We build a concrete implementation of our scheme and show that it can be deployed via a soft fork to existing anonymous currencies such as ZCash.

1 Introduction

Bitcoin has become increasingly popular as a decentralized electronic currency. In Bitcoin, each transaction is recorded in the *blockchain*, a public transaction ledger maintained by a set of decentralized peers. While this design has proven successful at low transaction volumes, the reliance on a globally-shared ledger has caused serious scaling issues. Since in Bitcoin 1MB blocks are added to the blockchain every ten minutes on average, the Bitcoin transaction rate is limited to fewer than ten new transactions per second across the entire Bitcoin user base [bit17].[1] Several proposals to increase blockchain bandwidth are being debated in the Bitcoin community today, but none are likely to produce a transaction rate that competes with centralized services such as payment card networks.

A promising approach to addressing the scaling problem is to move the bulk of Bitcoin transactions *off chain*, while preserving the system's decentralized structure and strong integrity guarantees. The leading proposal for off-chain payments is to use *payment channels*, exemplified by the Lightning Network [PD16] and Duplex Micropayment Channels [DW15]. Rather than posting individual payment transactions to the blockchain, channels employ the blockchain to first establish a shared deposit between two parties. The parties interact directly to make payments — adjusting the respective ownership shares of the deposit — and communicate with the blockchain only to agree on the final split of escrowed funds. In cases where no direct payment channel exists between two

[1] As of early May 2017, this has resulted in a backlog of nearly 165,000 transactions [Bun17].

parties, these proposals also allow participants to route transactions via intermediate peers [PD16]. The main benefit of the payment channel paradigm is that it dramatically reduces the transaction volume arriving at the blockchain, without adding new trusted and centralized parties.

While payment channels offer a solution to the scaling problem, they inherit many of the well-known privacy weaknesses of Bitcoin [MPJ+13, RS13]. Although payments are conducted off chain, any party may learn the pseudonymous identities and initial (resp. final) channel balances of the participants. More critically, payment channels provide few privacy protections against transaction counterparties. By establishing a channel to pay for *e.g.,* Tor bandwidth or web content, a user implicitly links each payment on a given channel to all of her other payments on this channel. This is particularly problematic in the likely event that payments are routed via a common intermediate peer — such as a currency exchange — since the intermediary must now be trusted to keep private your full payment history. Some proposals, such as the Lightning Network, have proposed to work around this problem by routing the payment via *multiple* intermediary nodes; however (as we discuss in §6) this approach substantially increases the complexity of establishing payment channels, and reveals payment information in the event that even a subset of the intermediaries collude.

Several academic works have recently proposed solutions that address the privacy problems of Bitcoin-type currencies [MGGR13, DFKP13, SCG+14, MMLN17]. Some of the resulting systems been publicly deployed, notably ZCash [zca17] (an implementation of the Zerocash protocol [SCG+14]) and Monero [mon17]. Unfortunately, the privacy mechanisms contained in these systems apply to the privacy of transactions *on the blockchain*, and do not address the setting of payment channels. Indeed privacy for payment channels seems fundamentally challenging due to channels' pairwise structure. Even when a channel is funded with anonymous currency, repeated payments within the same channel are inherently linkable. This is concerning, given that one of the main proposed applications of channels is for *web micropayments* — which are often described as a more private alternative to tracking and online behavioral advertising.

We stress that concerns about privacy are not theoretical. Several commercial ventures [Ell13, Blo14, Cha15] have been founded around the task of analyzing and tracing blockchain transactions. It is reasonable to expect that surveillance will be applied to payment channel systems if they become widely deployed.

Our Contribution. In this paper we propose Blind Off-chain Lightweight Transactions, or *Bolt*. Bolt consists of a set of techniques for constructing *privacy-preserving* payment channels for a decentralized currency. These techniques ensure that multiple payments on a single channel are unlinkable to each other and — if channels are funded with anonymized capital [2] — anonymous.

Our constructions enhance earlier work in privacy-preserving decentralized payments [MGGR13, DFKP13, SCG+14] while addressing the problem of providing fast and private off-chain transactions. Unlike earlier proposals [HBG16], which simply obfuscate participant identities from intermediaries, our proposals create anonymous direct channels even when a merchant does not know the identity of the paying party. Of more practical interest, we prototype one of our constructions and show that we can deploy Bolt as a soft fork of a Bitcoin like cryptocurrency. We provide constructions for three types of channels:

> **Unidirectional payment channels** allow a customer to pay a merchant fixed value coins repeatedly without linking the payments together. While our approach builds on the *compact e-cash* paradigm introduced by Camenisch *et al.* [CHL05], it requires a novel mechanism to achieve *succinct* channel closure, ensuring blockchain space usage is constant, regardless of the transaction volume.

[2]This requires either that the underlying cryptocurrency, is itself anonymous, e.g. as in ZCash [zca17], or that there exists some way of anonymizing or mixing that adds sufficient anonymity to non-anonymous cryptocurrency.

Bidirectional Channels allow two parties to exchange arbitrary valued payments in either direction, without linking the payments. The challenge here is preventing a malicious counterparty from using obsolete information to claim an earlier balance, while maintaining the scheme's unlinkability. While multiple payments on the same channel are unlinkable, to avoid linking an aborted payment to the payer's identity, our construction requires that the underlying payment channel be funded anonymously.

Indirect channels Finally, we extend our bidirectional payment channel construction to enable *third party payments*, where an untrusted intermediary acts as a "bridge" allowing two otherwise unconnected parties to exchange value. Critically, the intermediary learns neither the *identity* of the parties nor the *amount* transacted.

1.1 Background on Payment Channels

A payment channel is a relationship established between two participants in a privacy-preserving decentralized ledger-based currency network. While payments may flow in either direction on an established channel, the parties themselves are not symmetric: for a payment channel to work, at least one party must initiate the connection. For simplicity of exposition, we will refer to the initiating party as a *customer*, and the responding party as a *merchant*. We assume that the payment network includes a means to validate published transactions and to resolve disputes according to public rules. In principle these requirements can be satisfied by the scripting systems of consensus networks such as Monero or ZCash, using only minimal script extensions (which we discuss in §5.) We stress that our proposals in this work focus on the privacy of payment channels, and thus we assume the privacy of the underlying funding network.

When two parties wish to open a channel, the parties first agree on the respective balance shares of the channel, which we represent by non-negative integers B_0^{merch} and B_0^{cust}. The parties establish the channel by posting a payment to the network. Provided that these transactions are correctly structured, the network places the submitted funds in "escrow" until a subsequent closure transaction is received. The customer now conducts payments by interacting off-chain with the merchant. For some positive or negative integer payment amount ϵ_i, the i^{th} payment can be viewed as a request to update $B_i^{\mathsf{cust}} := B_{i-1}^{\mathsf{cust}} - \epsilon_i$ and $B_i^{\mathsf{merch}} := B_{i-1}^{\mathsf{merch}} + \epsilon_i$, with the sole restriction that $B_i^{\mathsf{merch}} \geq 0$ and $B_i^{\mathsf{cust}} \geq 0$. At any point, one or both parties may request to close the channel by posting a channel closure message to the ledger. If the closure messages indicate that the parties disagree about the current state of the channel, the ledger executes a dispute resolution algorithm to determine the final channel balances. After a delay sufficient to ensure each party has had an opportunity to contribute its closure message, the parties may recover their final shares of the channel balance using an on-chain payment transaction.

Any payment channel must meet two specific requirements, which we refer to as *universal arbitration* and *succinctness*:

1. **Universal arbitration.** In the event that two parties disagree about the state of a shared channel, the payment network can reliably arbitrate the dispute without requiring any private information.

2. **Succinctness.** To make payments scalable, all information posted to the ledger must be compact — *i.e.,* the size of this data should not grow linearly with the balance of the channel, the number of transactions or the amounts exchanged.

The latter property is an essential requirement for the setting of payment channels, since it rules out degenerate solutions that result in a posted transaction for every offline payment, or that post the full off-chain payment interaction to the ledger.

1.2 Customers, Merchants, and the Limits of Anonymity for Payment Channels

Informally our constructions for payment channels provide the following privacy guarantee:

> *Upon receiving a payment from some customer, the merchant learns no information beyond the fact that a valid payment (of some known positive or negative value) has occurred on a channel that is open with them. The network learns only that a channel of some balance has been opened or closed.*

Note, however, that the privacy protections against a channel participant are slightly weaker than those against third parties. This is an inherent limitation of the payment channel setting. Moreover, these limitations change depending on if a payment is made over a single direct channel or an indirect channel consisting of a series channels between the customer, one or more intermediaries, and a merchant. We explain these limitations further here.

Direct channels. The direct channel setting has three limitations. First, the privacy provided for direct channels is asymmetric: only the party initiating the payment is anonymous and unlinkable between payments, while the target of the payment is pseudonymous. This holds because at least one party must know which payment channel is being used.

Second, when receiving a payment on a channel, the recipient knows the payment came from someone with whom they have an open channel. This is also fundamental to the nature of channels, since they must be established with a counter-party before being used.

This final requirement suggests that recipients should use well-known channel parameters to group all channels, and thus maximize the anonymity set of its customers. If a recipient provides unique channel parameters to each potential payer (*i.e.,* behaves as though it was a different party to each payer), than the payer receives no privacy — as the set of channels open under that set of parameters has an anonymity set of a single person.

This setting maps well to a situations where the payment target is known, *e.g.,* where a single merchant or website accepts payments from many anonymous customers. Thus for the remainder of the paper we term this well-known target party the *merchant*, and refer to the paying party as the *customer*. We keep this terminology even when in settings where payment amounts are negative (resulting in a reverse payment), since one party must still be well known and this terminology maps to the case where, e.g., a merchant is refunding a customer for a previous purchase. We stress that to anyone not a party to the payment channel, privacy is absolute.

Indirect channels For indirect channels which involve one or more intermediate channels, the privacy guarantees may, surprisingly, be stronger than the direct case. First, this configuration facilitates a larger anonymity set, since it encompasses any party who has a channel open with the entry intermediary (for the initiator) as well as anyone who has a channel open with the exit intermediary (for the target). Additionally, when channels contain a single intermediary, they can be configured such that the merchant remains anonymous to the customer. Specifically, although the underlying pair-wise channels still offer asymmetric privacy (*i.e.,* one party is well-known), we can arrange the indirect channels so that the customer and merchant are both holding the private end of their channel and instead the intermediary is the only well-known party. We discuss this arrangement in §4.3.

The advantages of intermediaries do not fully generalize to chains containing more than a single intermediary. Specifically, we show that channels with a single intermediary can be configured to hide the payment amount from the intermediary. However, channels which involve more than one intermediary cannot hide the value of a payment from all intermediaries. Regardless of cryptographic underpinnings, at least one endpoint[3] of each channel must know the channel balance or else the channel cannot be closed. As a result, in any chain of channels with multiple intermediaries, at least one channel will have an intermediary party on both endpoints, and one of these parties will inevitably learn the value of the payments. This is not a limitation of our techniques but simply a consequence of the nature of payment channels.

1.3 Overview of our constructions

In this work we investigate two separate paradigms for constructing anonymous payment channels. Our first construction builds on the electronic cash, or e-cash, paradigm first introduced by Chaum [Cha83] and extended in many subsequent works, *e.g.,* [CFN90, Bra93, CHL05]. This unidirectional construction allows for succinct payments of fixed-value tokens from a customer to a merchant, while preserving the anonymity and functionality of a traditional payment channel. Our second construction extends these ideas to allow for variable-valued payments that traverse the channel in either direction (*i.e.,* each payment may have positive or negative value), at the cost of a more complex abort condition. Finally, we show how to extend our second construction to support path payments where users pay anonymously via a single untrusted intermediate party or a chain of intermediaries.

We now present the intuition behind our constructions.

Unidirectional payment channels from e-cash. An e-cash scheme is a specialized protocol in which a trusted party known as a *bank* issues one-time tokens (called *coins*) that customers can redeem exactly one time. "Offline" e-cash protocols seem like a natural candidate for implementing a one-way payment channel. For purposes of exposition, let us first consider a "strawman" proposal based on some ideal offline e-cash scheme that allows for the detection of doubly-spent coins. In this proposal, the merchant plays the role of the bank. After confirming that the customer has funded a channel, it issues a "wallet" of anonymous coins to the customer, who then spends them back to the merchant. To close the channel, the customer spends the remaining coins to herself and posts the evidence to the payment network. The merchant can dispute the customer's statement by providing evidence of a doubly-spent coin.

This strawman protocol suffers from several weaknesses. Most obviously, it is not *succinct*, since closure requires the customer to post all of her unspent coins. Secondly, there is an issue of timing: the merchant cannot issue a wallet to the customer until the customer's funds have been escrowed by the network, a process that can take from minutes to hours. At the same time, the customer must be assured that she can recover her funds in the event that the merchant fails to issue her a wallet, or aborts during wallet activation. Finally, to avoid customer "framing" attacks (in which a merchant issues coins to itself and then accuses the customer of double-spending) we require an e-cash scheme with a specific property called *exculpability*: namely, it is possible for any third party (in our case the payment network) to distinguish "true" double spends — made by a cheating customer — from false double-spends created by the merchant.

Intuition behind our unidirectional construction. To address the first concern, we begin with a *compact* e-cash scheme [CHL05]. Introduced by Camenisch *et al*, this is a form of e-cash in which

[3]It is possible that neither endpoint knows the balance in full and instead must cooperate to learn it. This does not alter the problem.

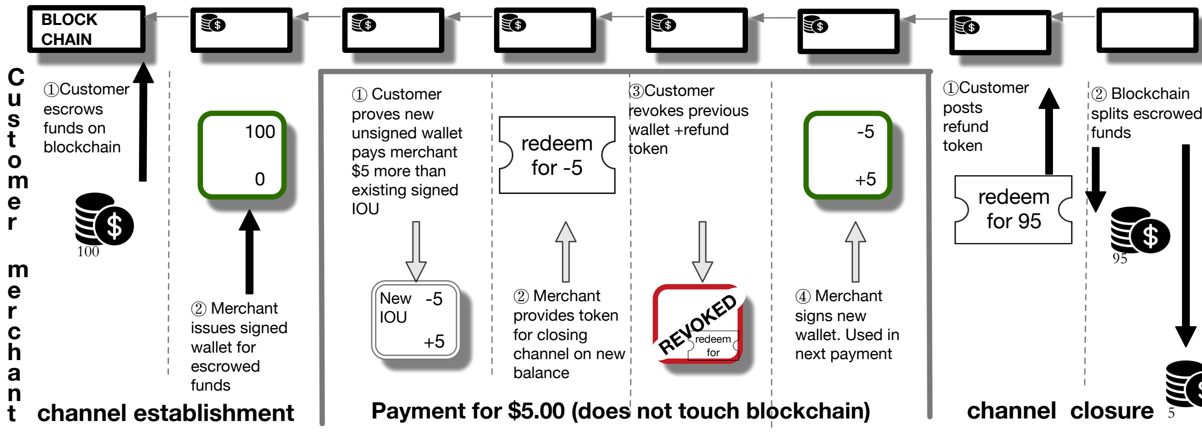

Figure 1: High level description of bidirectional channel protocol. The customer is the anonymous party. The merchant is a known identity. Only channel establishment and closure touch the blockchain.

B separate coins can be generated from a constant-sized wallet stored at the customer (here B is polynomial in the wallet size). While compact e-cash reduces the wallet storage cost, it does not immediately give rise to a succinct closure mechanism for our channels. The key innovation in our construction is a new mechanism that reduces channel closure to a single fixed-size message — at the cost of some increased (off-chain) interaction between the merchant and customer.

To create a payment channel in our construction, the customer first commits to a set of secrets used to formulate the wallet. These are embedded within a succinct *wallet commitment* that the customer transmits to the payment network along with the customer's escrow funds (and an ephemeral public signature verification key pk_c). The customer and merchant now engage in an interactive channel establishment protocol that operates as follows. The customer first generates B coin spend transactions, and attaches to each a non-interactive zero knowledge proof that each coin is tied to the wallet commitment. She then individually encrypts each of the resulting transactions using a symmetric encryption scheme such that each ciphertext C_i embeds a single spend transaction, along with the decryption key for ciphertext C_{i+1}. After individually signing each of the resulting ciphertexts using her secret key, the customer transmits the signed results to the merchant for safekeeping. A critical aspect of this scheme is that from the merchant's perspective these ciphertexts are opaque: the customer does not need to prove to the merchant that any ciphertext is well-formed.

When the customer wishes to close an active channel with remaining balance N (for $0 < N \leq B$), she computes $j = (B - N) + 1$ and posts a signed message $(\mathsf{channel\ ID}, j, k_j)$ to the network, with k_j being the decryption key for the j^{th} ciphertext. The merchant can use this tuple to decrypt each of the ciphertexts $C_j, \ldots, C_B$ and thus detect further spending on the channel. If the customer cheats by revealing an invalid decryption key, or if any ciphertext decrypts to an invalid coin, or if the resulting transactions indicate that she has double-spent any coin, the merchant can post indisputable evidence of this cheating to the network — which, to punish the customer, will grant the full channel balance to the merchant.

Bidirectional payment channels. A restriction on the previous construction is that it is *unidirectional*: all payments must flow from the customer to the merchant. While this is sufficient for many useful applications — such as micropayments for web browsing — some applications of payment channels require payments to flow from the merchant to the customer. As we further discuss below, a notable example of such an application is *third party payments*, where two parties send funds via an intermediary, who must increase the value of one channel while decreasing the other.

For these applications, we propose a second construction that combines techniques from existing (non-anonymous) payment channels with blind signatures and efficient zero-knowledge proofs. As in the existing payment channel systems [PD16, DW15], the customer and merchant first on agree on an initial channel state, with the customer holding B_0^{cust} escrowed funds, and the merchant provides a signature on this balance. When the customer wishes to pay the merchant an arbitrary positive or negative amount ϵ, she conducts an interactive protocol to (1) prove knowledge of the previous signature on the current balance B_{i-1}^{cust}, and (2) demonstrate that she possesses sufficient balance to complete the payment. She then (3) blindly extracts a new signed *refund token* from the merchant containing the updated balance $B_i^{\mathsf{cust}} = B_{i-1}^{\mathsf{cust}} - \epsilon$. At any point, the customer may post her most recent refund token to the blockchain to redeem her available funds. See figure 1.

The main challenge in this approach is to prevent a dishonest customer from retaining and using earlier versions of her refund token on channel closure. To prevent this, during each payment, the customer interacts with the merchant to present a *revocation token* for the previous state. As long as the customer behaves honestly, this revocation token can never be linked to the channel or to any previous transactions. However, if the customer misbehaves by posting an obsolete refund token, the merchant can instantly detect this condition and present the revocation token to the network as proof of the customer's malfeasance – in which case, the network awards the balance of the channel to the merchant. Unlike the e-cash approach, this proposal suffers from the possibility that one of the parties will *abort* the protocol early; we address this by using the network to enforce fairness.

From direct to third-party payments. As the concluding element of our work, we show how a bidirectional payment channel can be used to construct *third-party* payments, in which a first party **A** pays a second party **B** via a common, untrusted intermediary **I** to which both parties have previously established a channel. In practice, this capability eliminates the need for parties to maintain channels with all of their peers. The key advantage of our proposal is that the intermediary **I** cannot link transactions to individual users, nor — surprisingly — can they learn the amount being paid in a given transaction. Similarly, even if **I** is compromised, it cannot claim any transactions passing through it. This technique makes anonymous payment channels usable in practice, provided there exists a highly-available (untrusted) intermediary to route the connections. We provide the full details of our construction and how to extend it to support multiple intermediaries in §4.3.

Aborts. Our unidirectional protocol provides privacy guarantees that are similar to the underlying e-cash protocol, with the obvious (and necessary) limitation that final channel balances are revealed on closure. Payments between a customer and merchant are non-interactive and completely anonymous. The bidirectional payment construction, on the other hand, provides a slightly weaker guarantee: by aborting during protocol execution, the merchant can place the customer in a state where she is unable to conduct future transactions. This does not prevent the customer from resorting to the network to close the channel, but it does raise concerns for anonymity in two ways:

1. The merchant can arbitrarily reduce the anonymity set by (even temporarily) evicting other users through induced aborts.

2. The merchant may link a user to a repeating sequence of transactions by aborting the user in the middle of the sequence.

For many traditional commerce settings, the consequences of such aborts may be minimal: no matter the payment mechanism, the merchant can fail to deliver the promised goods and the customer will almost certainly abort. For other settings, such as micropayments, these possibilities should be considered. In such settings customers should scan the network for premature closures and abort the channel if the number of open channels with a merchant falls below their minimal anonymity set.

1.4 Comparison to related work

In concurrent work, Heilman *et al.* proposed an elegant mixing system called Tumblebit [HAB$^+$17]. Tumblebit is compatible with classical Bitcoin and operates in two modes. The first allows users to anonymize (aka mix or "launder") their own coins. The second mode allows for payment channels between distinct users. Overcoming the limited choice of cryptographic primitives to get Bitcoin compatibility is a serious achievement, but for Tumblebit it comes at the cost of far more limited features, performance, and privacy in comparison to Bolt's payment channels.

Most significantly, in a payment from Alice to Bob, "Bob and the Tumbler can collude to learn the true identity of Alice" [HAB$^+$17] since Alice identifies herself to the Tumbler when making a payment because she must pay the Tumbler with traceable Bitcoins.[4] In contrast both our schemes provide provable privacy for Alice even in the face of corrupted and colluding parties. Second, Tumblebit does not hide payment values.

On the functionality side: Tumblebit payments are of a single fixed value and payment channels are unidirectional. In contrast we provide for bidirectional payment channels with variable valued payments. Tumblebit payments are also not succinct: a channel allowing n payment needs either $O(n)$ state on the blockchain or $O(n^2)$ invocations of their protocol.

On the performance side: at $387ms$ per channel payment, Tumblebit is 5 times slower than our prototype implementation of Bolt's bidirectional channels. We stress that this is not due to a design flaw in Tumblebit: working within the confines of Bitcoin compatibility is extremely challenging and comes at a high cost.

Finally, like Tumblebit, our unidirectional protocol provides full protections from aborts. Our bidirectional protocol does not and requires an underlying anonymous currency for safety (see §1.3). Variable payments seem to require multiple rounds of interaction, thus risking aborts terminating in invalid intermediate states.

1.5 Outline of this paper

The remainder of this paper proceeds as follows. In §2 we present definitions for anonymous payment channels. In §3 we present the building blocks of our scheme. In §4 we describe the protocols for our payment channel constructions, and in §5 we present concrete instantiations of these protocols. Finally, in §6 we discuss the related work.

2 Definitions

Notation: Let λ be a security parameter. We write $P(\mathcal{A}(a), \mathcal{B}(b)) \to (c, d)$ to indicate a protocol P run between parties $\mathcal{A}$ and $\mathcal{B}$, where a is $\mathcal{A}$'s input, c is $\mathcal{A}$'s output, b is $\mathcal{B}$'s input and d is $\mathcal{B}$'s output. We will define $\nu(\cdot)$ as a negligible function. We will use $\mathsf{val}_{\mathsf{max}}$ to denote the maximum balance of a payment channel, and denote by the set of integers $\{\epsilon_{\mathsf{min}}, \ldots, \epsilon_{\mathsf{max}}\}$ the range of valid payment amounts.

[4] This is likely fundamental. See Section 7.c of [HAB$^+$17] for further discussion. Although Heilman et al. provide some mitigations for these attacks, as they acknowledge the Tumbler and Bob can still correlate Alice's interactions. Thus they cannot offer Alice provable privacy from Bob.

2.1 Anonymous Payment Channels

An Anonymous Payment Channel (APC) is a construct established between two parties that interact via a payment network. In this section we first describe the properties of an *anonymous payment channel scheme*, which is a collection of algorithms and protocols used to establish these channels. We then explain how these schemes can be used to construct channels in a payment network. We now provide a formal definition of an APC scheme.

Definition 2.1 (APC scheme)**.** An anonymous payment channel scheme consists of a tuple of possibly probabilistic algorithms ($\mathsf{KeyGen}, \mathsf{Init}_{\mathcal{C}}, \mathsf{Init}_{\mathcal{M}}, \mathsf{Refund}, \mathsf{Refute}, \mathsf{Resolve}$) and two interactive protocols ($\mathsf{Establish}, \mathsf{Pay}$). These are defined in Figure 2. For completeness we also define an optional function $\mathsf{Setup}(1^\lambda)$ to be run by a trusted party for generating the parameters pp, *e.g.,* a Common Reference String. In some instantiations the CRS is not required. In this case, we set $\mathsf{pp} := 1^\lambda$.[5]

Key generation and channel initialization algorithms:

$\mathsf{KeyGen}(\mathsf{pp})$. This algorithm generates a keypair (pk, sk) for use by each customer or merchant.

$\mathsf{Init}_{\mathsf{P}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk, sk)$. For $\mathcal{P} \in \{\mathcal{C}, \mathcal{M}\}$ this algorithm is run by each party prior to opening a channel. On input the initial channel balances, public parameters and the party's keypair, the $\mathsf{Init}_{\mathcal{C}}$ algorithm outputs the party's channel token $\mathsf{T}_{\mathcal{P}}$ and a corresponding secret $csk_{\mathcal{P}}$.

Two-party protocols run between a customer $\mathcal{C}$ and a merchant $\mathcal{M}$:

$\mathsf{Establish}(\{\mathcal{C}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}})\}, \{\mathcal{M}((\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, csk_{\mathcal{M}})\}$. On input public parameters and each of the initial channel tokens, the $\mathsf{Establish}$ protocol activates a channel between two parties who have previously escrowed funds. If successful, the merchant receives $\mathsf{established}$ and the customer receives a wallet w. Either party may receive the distinguished failure symbol $\perp$.

$\mathsf{Pay}(\{\mathcal{C}(\mathsf{pp}, \epsilon, w_{\mathsf{old}})\}, \{\mathcal{M}(\mathsf{pp}, \epsilon, \mathbf{S}_{\mathsf{old}})\})$. On input parameters, a payment amount ϵ, and a wallet w_{old} from a customer, and the merchant's current state $\mathbf{S}_{\mathsf{old}}$ (initially $\emptyset$) from the merchant: the customer receives a payment success bit $R_{\mathcal{C}}$ and new wallet w_{new} if the interaction succeeded. The merchant receives a payment success bit $R_{\mathcal{M}}$ and an updated state $\mathbf{S}_{\mathsf{new}}$ if the interaction succeeded.

Channel closure and dispute algorithms, run by the customer and merchant respectively:

$\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. On input a wallet w, outputs a customer channel closure message $\mathsf{rc}_{\mathcal{C}}$.

$\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. On input the merchant's current state $\mathbf{S}_{\mathsf{old}}$ and a customer channel closure message, outputs a merchant channel closure message $\mathsf{rc}_{\mathcal{M}}$ and an updated merchant state $\mathbf{S}_{\mathsf{new}}$.

Dispute resolution algorithm, run by the network:

$\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$. On input the customer and merchant's channel tokens $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$, along with closure messages $\mathsf{rc}_{\mathcal{C}}$, $\mathsf{rc}_{\mathcal{M}}$ (where either message may be null), this algorithm outputs the final channel balance $B_{\mathsf{final}}^{\mathsf{merch}}, B_{\mathsf{final}}^{\mathsf{cust}}$.

Figure 2: Definition of an Anonymous Payment Channel scheme.

Using Anonymous Payment Channels. An anonymous payment channel scheme must be used in combination with a payment network capable of conditionally escrowing funds and binding these escrow transactions funds to some data. Such payment networks can be constructed using blockchain-based systems, although they can be built from other technology as well. In this work we assume only the existence of a payment network with these capabilities, and leave the details of the

[5] Looking forward to our recommended instantiations in §5, we propose to use a CRS based on public randomness.

payment network's implementation (*e.g.,* modeling a blockchain) as a separate problem. We now describe how these algorithms and protocols are used to establish a channel on a payment network.

To instantiate an anonymous payment channel, the merchant $\mathcal{M}$ first generates a long-lived keypair $(pk_{\mathcal{M}}, sk_{\mathcal{M}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ that will identify it to all customers. The merchant initializes its state $\mathbf{S} \leftarrow \emptyset$. A customer $\mathcal{C}$ generates an ephemeral keypair $(pk_{\mathcal{C}}, sk_{\mathcal{C}})$ for use on a single channel. The customer and merchant agree on their respective initial channel balances $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}$. They now perform the following steps:

1. Each party executes the $\mathsf{Init}_{\mathcal{C}}$ algorithm on the agreed initial channel balances, in order to derive the channel tokens $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$.

2. The two parties transmit these tokens to the payment network along with a transaction to escrow the appropriate funds.

3. Once the funds have been verifiably escrowed, the two parties run the Establish protocol to activate the payment channel. If the parties disagree about the initial channel balances, this protocol returns $\perp$ and the parties may close the channel.

4. If channel establishment succeeds, the customer initiates the Pay protocol as many times as desired, until one or both parties close the channel.

5. If the customer wishes to close the channel, she runs Refund and transmits $\mathsf{rc}_{\mathcal{C}}$ along with the channel identifier to the payment network.[6]

6. The merchant runs Refute on the customer's closure token to obtain the merchant closure token $\mathsf{rc}_{\mathcal{M}}$.

At the conclusion of this process, the network runs the Resolve algorithm to determine the final channel balance and allows each party to collect the determined share of the escrowed funds.

2.2 Correctness and Security

We now described the correctness and security of an anonymous payment channel scheme. Here we provide intuition, and present formal definitions in Appendix B.

Correctness. Informally, an APC scheme is correct if for all correctly-generated parameters pp and opening balances $B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}} \in \{0, \ldots, \mathsf{val}_{\mathsf{max}}\}$, every correct (and honest) interaction following the paradigm described above always produces a correct outcome. Namely, each valid execution of the Pay protocol produces success, and the final outcome of Refute correctly reflects the final channel balance.

Security. The security of an Anonymous Payment Channel scheme is defined in terms of two games, which we refer to as *payment anonymity* and *balance.* We now provide an informal description of each property, and refer the reader to Appendix B for the formal definitions.

Payment anonymity. Intuitively, we require that the merchant, even in collaboration with a set of malicious customers, learns nothing about a customer's spending pattern *beyond* the information that is available outside of the protocol. In our anonymity definition, which extends a definition of Camenisch *et al.* [CHL05], the merchant interacts with either (1) a series of oracles implementing

[6] Here we assume that channel closure is initiated by the customer. In cases where the merchant wishes to initiate channel closure, it may transmit a special message to the network requesting that the customer close the channel.

the real world protocols for customers $\mathcal{C}_1, \ldots, \mathcal{C}_N$, or (2) with a simulator $\mathcal{S}$ that performs the customer's part of the Pay protocol. In the latter experiment, we assume a simulator that has access to side information not normally available to participants in the real protocol, *e.g.,* a simulation trapdoor or control of a random oracle. We require that the simulator has the ability to simulate any customer without access to the customer's wallet, and without knowing the identity of the customer being simulated. Our definition holds if no adversary can determine whether she is in world (1) or (2). We stress that this definition implies anonymity because the simulator has no information about which party it is simulating.

Balance. The balance property consists of two separate games, one for the merchant and one for the customer. In both cases, assuming honest execution of the $\mathsf{Resolve}$ protocol, this property ensures that no colluding set of adversarial counterparties can extract more value from a channel than justified by (1) the party's initial channel funding, combined with (2) the set of legitimate payments made to (or by) the adversary. Because the merchant and customer have different interfaces, we define this property in terms of two slightly different games. In each game, the adversarial customer (resp. merchant) is given access to oracles that play the role of the merchant (resp. customer), and allows the parties to establish an arbitrary number of channels with chosen initial balances. The adversary may then initiate (resp. cause the other party to initiate) the Pay protocol repeatedly on adversarially-chosen payment amounts ϵ. Finally, the adversary can initiate channel closure with the counterparty to obtain channel closure messages $\mathsf{rc}_{\mathcal{C}}$, $\mathsf{rc}_{\mathcal{M}}$. The adversary *wins* if the output of the $\mathsf{Resolve}$ protocol is inconsistent with the total value funded and paid.

3 Technical Preliminaries

In this section we recall some basic building blocks that we will use in our constructions.

Commitment schemes. Let $\Pi_{\mathsf{commit}} = (\mathsf{CSetup}, \mathsf{Commit}, \mathsf{Decommit})$ be a commitment scheme where CSetup generates public parameters; on input parameters, a message M, and random coins r, Commit outputs a commitment $\mathcal{C}$; and $\mathsf{Decommit}$ on input parameters and a tuple (C, m, r) outputs 1 if $\mathcal{C}$ is a valid commitment to the message, or 0 otherwise. In our instantiations, we recommend using the Pedersen commitment scheme [Ped92] based on the discrete logarithm assumption in a cyclic group.

Symmetric encryption schemes. Our constructions require an efficient symmetric encryption scheme as well as a one-time symmetric encryption scheme. We define a symmetric encryption scheme $\Pi_{\mathsf{symenc}} = (\mathsf{SymKeyGen}, \mathsf{SymEnc}, \mathsf{SymDec})$ where $\mathsf{SymKeyGen}$ outputs an ℓ-bit key. We also make use of a one-time encryption scheme $\Pi_{\mathsf{otenc}} = (\mathsf{OTKeyGen}, \mathsf{OTEnc}, \mathsf{OTDec})$. In practice, the encryption scheme can be implemented by encoding the plaintext as an element in a cyclic group $\mathbb{G}$ and multiplying by a random group element. In either case, our constructions require that the schemes provide IND-CPA security.

Pseudorandom Functions. Our unidirectional construction requires a pseudorandom function (PRF) F that supports efficient proofs of knowledge. For our purposes it is sufficient that the PRF be secure for a poly-size input space. In addition to the standard pseudorandomness property, ourprotocols require that the PRF should also possess a property we refer to as *strong pre-image resistance*. This property holds that, given access to an oracle implementing the function $F_s(\cdot)$ for a random seed s, no adversary can find an input point x and a pair (s', x') in the domain of the function such that $F_s(x) = F_{s'}(x')$ except with negligible probability. We propose to instantiate F

using the Dodis-Yampolskiy PRF [DY05], the public parameters are a group $\mathbb{G}$ of prime order q with generator g. The seed is a random value $s \in \mathbb{Z}_q$ and the function is computed as $f_s(x) = g^{1/(s+x)}$ for x in a polynomially-sized set. We show in Appendix E that the Dodis-Yampolskiy PRF satisfies the strong pre-image resistance property.

Signatures with Efficient Protocols. Our schemes make use of a signature scheme $\Pi_{\mathsf{sig}} = (\mathsf{SigKeygen}, \mathsf{Sign}, \mathsf{Verify})$ with efficient protocols, as proposed by Camenisch and Lysyanskaya [CL02]. These schemes feature: (1) a protocol for a user to obtain a signature on the value(s) in a commitment without the signer learning anything about the message(s), and (2) a protocol for (non-interactively) proving knowledge of a signature. Several instantiations of these signatures have been proposed in the literature, including constructions based on the Strong RSA assumption [CL02] and various assumptions in bilinear groups [BCKL08, CL04]. For security, we assume that all signatures satisfy the property of *existential unforgeability under chosen message attack* (EU-CMA).

Non-Interactive Zero-Knowledge Proofs. We use several standard results for non-interactively proving statements about committed values, such as (1) a proof of knowledge of a committed value, and (2) a proof that a committed value is in a range. When referring to the proofs above, we will use the notation of Camenisch and Stadler [CS97]. For instance, $PoK\{(x, r) : y = g^x h^r \ \wedge \ (1 \leq x \leq n)\}$ denotes a zero-knowledge proof of knowledge of integers x and r such that $y = g^x h^r$ holds and $1 \leq x \leq n$. All values not in enclosed in ()'s are assumed to be known to the verifier. Our protocols require a proof system that provides *simulation extractability*, which implies that there exists an efficient proof extractor that (under specific circumstances, such as the use of a simulation CRS) can extract the witness used by an adversary to construct a proof, even when the adversary is also supplied with simulated proofs. In practice we can conduct these proofs non-interactively using a variety of efficient proof techniques [BCKL08, Sch91, CDS94, Bra97, CNS07, GS, CC$^+$08, Bou00, Gro06].

4 Protocols

In this section we present our main contribution, which consists of three protocols for implementing anonymous payment channels. Our first protocol in §4.1 is a unidirectional payment channel based on e-cash techniques. Our second construction in §4.2 allows for bidirectional payments, with a more complex protocol for handling aborts. Finally, in §4.3 we propose an approach for third-party payments, in which two parties transmit payment via an *intermediary*.

4.1 Unidirectional payment channels

Our first construction modifies the compact e-cash construction of Camenisch *et al.* [CHL05] to achieve efficient and *succinct* unidirectional payment channels. We now provide a brief overview of this construction.

Compact e-cash. In a compact e-cash scheme, a customer withdraws a fixed-size wallet capable of generating B coins. The customer's wallet is based on a tuple (k, sk, B): k is an (interactively generated) seed for a pseudorandom function F, sk is the customer's private key, and B is the number of coins in the wallet. Once signed by the merchant, this wallet can be used to generate up to B coins as follows: the i^{th} coin consists of a tuple (s, T, π) where s is a "serial number" computed as $s = F_k(i)$; T is a "double spend tag" computed such that, if the same coin is spent twice, the double spend tags can be combined to reveal the customer's key pk (or sk); and π is a non-interactive zero-knowledge proof of the following statements:

1. $0 < i \leq B$
2. The prover knows sk.
3. The prover has a signature on the wallet (k, sk, B).
4. The pair (s, T) is correctly structured with respect to the signed wallet.

This construction ensures that double spending is immediately detected by a verifier, since both transactions will share the serial number s.[7] The verifier can then recover the spender's public key by combining the double-spend tags. At the same time, the individual coin spends cannot be linked to each other or to the user. Camenisch *et al.* [CHL05] show how to construct the proof π efficiently using signatures and proof techniques secure under the Strong RSA or bilinear assumptions in the random oracle model. Subsequent work presents efficient proofs in the standard model [BCKL08, BCKL09].

Achieving succinct closure. Let us recall our intuition for using compact e-cash in a unidirectional payment channel (see §1.3). In this proposal, the merchant plays the role of the bank and issues the customer a wallet of B coins, which she can then (anonymously) spend back to the merchant. To close a channel, the customer simply spends any unused coins "to herself", thus proving to the merchant that she retains no spending capability on the channel (since any subsequent attempt to spend those coins would be recognized by the merchant as a double spend). Unfortunately while compact e-cash provides a succinct wallet, this does not immediately lead to a succinct protocol for closing the channel — as the customer cannot simply reveal the wallet secrets without compromising the anonymity of previous coins spent on the channel. We require a mechanism to succinctly reveal only a fraction of the coins in a wallet, without revealing them all. At the same time, we wish to avoid complex proofs (*e.g.,* a proving cost that scales with $O(B)$).[8]

Our approach is to use the merchant to store the necessary information to verify channel closure. This requires a number of changes to the compact e-cash scheme of Camenisch *et al.* [CHL05] (requiring a fresh analysis of the scheme, which we provide in §4.1.1). First, we design the customer's $\mathsf{Init}_{\mathcal{C}}$ algorithm so that the PRF seed k is generated solely by the customer, rather than interactively by the customer and the bank (merchant) as in [CHL05]. The customer now commits to the wallet secrets, producing wCom, and embeds this into the customer's channel token $\mathsf{T}_{\mathcal{C}} := (\mathsf{wCom}, pk_c)$ where pk_c is a signature verification key. During the $\mathsf{Establish}$ protocol to obtaining the merchant's signature on wCom, the customer provides the merchant with a series of signed ciphertexts $(C_1, \ldots, C_B)$, each of which contains a coin spend tuple of the form (s, T, π') where π' is identical to the normal compact e-cash proof, but simply proves that s, T are correct with respect to wCom (which is not yet signed by the merchant). These ciphertexts are structured so that a key revealed for the j^{th} ciphertext will also open each subsequent ciphertext.

The key feature of this approach is that the merchant *does not need to know if these ciphertexts truly contain valid proofs* at the time the channel is opened. To reveal the remaining j coins in a channel, the customer reveals a key for the j^{th} ciphertext, which allows the merchant to "unlock" all of the remaining coin spends and verify them with respect to the commitment wCom embedded in the customer's channel token. If any ciphertext fails to open, or if the enclosed proof is not valid,

[7] In the original compact e-cash construction [CHL05], the key k was generated using an interactive protocol between the customer and bank, such that honest behavior by one party ensured that k was uniformly random. In our revised protocol below, k will be chosen only by the customer. This does not enable double-spending, provided that the PRF is deterministic and the proof system is sound.

[8] Indeed, an alternative proposal is to construct the coin serial numbers using a chained construction, where each s_i is computed as a one-way hash of the key used in the previous transaction. This would allow the customer to revoke the channel by posting a secret from one transaction. Unfortunately, proving the correctness of s_i using standard zero-knowledge techniques would then require $O(B)$ proving cost, and moreover, does not seem easy to accomplish using the efficient zero knowledge proof techniques we recommend in this work.

the merchant can easily prove malfeasance by the customer and obtain the balance of the channel. This requires only symmetric encryption and a means to "chain" symmetric encryption keys – both of which can easily be constructed from standard building blocks.[9] Our schemes additionally require a one-time encryption algorithm OTEnc where the keyspace of the algorithm is also the range of the pseudorandom function F.

We now present the full scheme:

$\mathsf{Setup}(1^\lambda)$. On input λ, optionally generate CRS parameters for (1) a secure commitment scheme and (2) a non-interactive zero knowledge proof system. Output these as pp.

$\mathsf{KeyGen}(\mathsf{pp})$. Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1^\lambda)$.[10]

$\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. On input a keypair (pk_c, sk_c), uniformly sample two distinct PRF seeds k_1, k_2 and random coins r for the commitment scheme. Compute $\mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2, B_0^{\mathsf{cust}}; r)$. For $i = 1$ to B, sample $ck_i \leftarrow \mathsf{SymKeyGen}(1^\lambda)$ to form the vector $\vec{ck}$. Output $\mathsf{T}_{\mathcal{C}} = (\mathsf{wCom}, pk_c)$ and $csk_{\mathcal{C}} = (sk_c, k_1, k_2, r, B_0^{\mathsf{cust}}, \vec{ck})$.

$\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.

$\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. Parse w (generated by the $\mathsf{Establish}$ and Pay protocols) to obtain $\vec{ck}$ and the current coin index i. Compute $\sigma \leftarrow \mathsf{Sign}(sk_c, \mathsf{refund}\|\mathsf{cID}\|i\|ck_i)$ (where cID uniquely identifies the channel being closed) and output $\mathsf{rc}_{\mathcal{C}} := (\mathsf{cID}, i, ck_i, \sigma)$.

$\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. Parse the customer's channel closure message $\mathsf{rc}_{\mathcal{C}}$ as $(\mathsf{cID}, i, ck_i, \sigma)$ and verify cID and the signature σ. If the signature verifies, then obtain the ciphertexts $C_i, \ldots, C_B$ stored after the $\mathsf{Establish}$ protocol. For $j = i$ to B, compute $(j\|s_j\|u_j\|\pi_j^r\|ck_j\|\hat{\sigma}_j) \leftarrow \mathsf{SymDec}(ck_j, C_j)$ and verify the signature $\hat{\sigma}_j$ and the proof π_j^r. If (1) the signature $\hat{\sigma}_j$ or the proof π_j^r fail to verify, (2) any ciphertext fails to decrypt correctly, or (3) any of the decrypted values (s_j, u_j) match a valid spend containing (s_j, t_j) in $\mathbf{S}$ where $\mathsf{OTDec}(u_j, t_j) = pk_c$: record the invalid result into $\mathsf{rc}_{\mathcal{M}}$ along with cID and sign the result using sk_m so that it can be verified by the network. Otherwise set $\mathsf{rc}_{\mathcal{M}} = (\mathsf{accept})$ and sign with sk_m. Finally for each valid C_j, set $\mathbf{S} \leftarrow \mathbf{S} \cup (s_j, t_b, \pi)$ and output $\mathbf{S}$ as the new merchant state.

$\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$. Parse the customer and merchant closure messages and verify all signatures. If any fail to verify, grant the balance of the channel to the opposing party. If $\mathsf{rc}_{\mathcal{C}} = (N, sk_N, \sigma)$ and $\mathsf{rc}_{\mathcal{M}} = \mathsf{accept}$ then set $B_{\mathsf{final}}^{\mathsf{cust}}$ to $(B_0^{\mathsf{cust}} - N) + 1$. Otherwise, evaluate the merchant closure message to determine whether the customer misbehaved. If so, assign the merchant the full balance of the channel.

We present the $\mathsf{Establish}$ and Pay protocols in Figure 3.

4.1.1 Security Analysis

Theorem 4.1. *The unidirectional channel scheme satisfies the properties of* anonymity *and* balance *under the assumption that* (1) *F is pseudorandom and provides strong pre-image resistance,* (2) *the commitment scheme is secure,* (3) *the zero-knowledge system is sound and zero-knowledge,* (4) *the signature scheme is existentially unforgeable under chosen message attack and signature extraction is blind, and* (5) *the symmetric encryption and one-time encryption scheme are each IND-CPA secure.*

Customer($\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}$) — **Establish** — Merchant($\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, csk_{\mathcal{M}}$)

1. Parse $csk_{\mathcal{C}}$ as $(pk_c, sk_c, k_1, k_2, r, B)$. Sample $sk_0 \in \{0,1\}^{\ell}$.
2. Generate a proof π_1 that
$$PK\{(sk_c, k_1, k_2, r) : \mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2; r) \wedge (pk_c, sk_c) \in \mathsf{KeyGen}(1^{\lambda})\}$$
3. For $j = 1$ to B:
   (a) Compute $s_j \leftarrow F_{k_1}(j), u_j \leftarrow F_{k_2}(j), \pi_j^r$ where
$$\pi_j^r = PK\{(sk_c, k_1, k_2, r) : \; s = F_{k_1}(j) \; \wedge u = F_{k_2}(j) \wedge \mathsf{wCom} = \mathsf{Commit}(sk_c, k_1, k_2; r) \wedge (pk_c, sk_c) \in \mathsf{KeyGen}(1^{\lambda})\}$$
   (b) Compute an internal signature $\hat{\sigma}_j = \mathsf{Sign}(sk_c, \mathsf{spend}\|j\|s_j\|u_j\|\pi_j^r\|ck_{j+1})$.
   (c) Compute $C_j = \mathsf{SymEnc}(ck_j, j\|s_j\|u_j\|\pi_j^r\|\hat{\sigma}_j\|ck_{j+1})$ and an external signature $\sigma_j = \mathsf{Sign}(sk_c, \mathsf{coin}\|j\|C_j)$.

$\xrightarrow[(C_1,\sigma_1,\ldots,C_B,\sigma_B)]{\mathsf{wCom}, \pi}$

$\xleftrightarrow{\sigma_w}$

Verify the signature on $\mathsf{T}_{\mathcal{C}}$, and check that $B_0^{\mathsf{cust}} = B$. Verify π_1 and for $i = 1$ to B, verify the signature σ_j on C_j. If any check fails, abort and output $\perp$. Otherwise, interact with the customer to provide a blind signature σ_w on the contents of wCom.

Return $w = (sk_0, sk_c, k_1, k_2, r, B, \sigma_w, 1)$. — Return $\mathsf{established}$.

Customer($\mathsf{pp}, \epsilon, w_{\mathsf{old}}$) — **Pay** — Merchant($\mathsf{pp}, \epsilon, \mathbf{S}_{\mathsf{old}}$)

1. Parse w_{old} as $(sk_0, sk_c, k_1, k_2, r, B, \sigma_w, i)$. Abort if $i \geq B$.
2. Compute $s \leftarrow F_{k_1}(i)$, $t \leftarrow \mathsf{OTEnc}(F_{k_2}(i), pk_c)$, and a proof:
$$\pi = PK\{(pk_c, sk_c, k_1, k_2, r, i, \sigma_w) : \; s = F_{k_1}(i) \wedge 0 < i \leq B \wedge t = \mathsf{OTEnc}(F_{k_2}(i), pk_c) \wedge \mathsf{Verify}(pk_m, (k_1, k_2, sk_c), \sigma_w) \wedge (pk_c, sk_c) \in \mathsf{KeyGen}(\mathsf{pp}) \; \}$$

$\xrightarrow{(s, t, \pi)}$

Verify π and that $(s, \cdot, \cdot) \notin \mathbf{S}$. If so, set $\mathbf{S} \leftarrow \mathbf{S} \cup (s, t, \pi)$ and set $R_{\mathcal{M}} \leftarrow 1$, else set $R_{\mathcal{M}} \leftarrow \perp$.

Return $w_{\mathsf{new}} := (sk_0, sk_c, k_1, k_2, r, B, \sigma_w, i + 1)$. — Return $R_{\mathcal{M}}$.

Figure 3: Establishment and Payment protocols for the Unidirectional Payment Channel scheme.

Customer($\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}$) | Establish | Merchant($\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, csk_{\mathcal{M}}$)

1. Parse $csk_{\mathcal{C}}$ to obtain $(\mathsf{cID}, \mathsf{wCom}, wpk, wsk, r, B_0^{\mathsf{cust}})$.
2. Generate a proof π_1 of the following statement:

$$\pi_1 = PK\{(wpk, wsk, r) : \ \mathsf{wCom} = \mathsf{Commit}(\mathsf{cID}, wpk, B_0^{\mathsf{cust}}; r) \\ \wedge\ (wpk, wsk) \in \mathsf{KeyGen}(\mathsf{pp})\}$$

$\xrightarrow{\pi_1}$

$\xleftrightarrow{\sigma_w}$

Parse $\mathsf{T}_{\mathcal{C}}$ to obtain $B_0^{\mathsf{cust}}, \mathsf{wCom}$. Verify that the proof π_1 is valid. If not, output $\perp$, If the proof is valid: interact with the customer to provide a blind signature σ_w under sk_m on the contents of wCom.

Return $w := (B_0^{\mathsf{cust}}, wpk, wsk, r, \sigma_w)$. | Return $\mathsf{established}$.

Customer($\mathsf{pp}, \epsilon, w_{\mathsf{old}}$) | Pay | Merchant($\mathsf{pp}, \epsilon, \mathbf{S}_{\mathsf{old}}$)

1. Parse w_{old} as $(\mathsf{cID}, B, wpk, wsk, r, \sigma_w)$.
2. Sample $(wpk', wsk') \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and sample random coins r'.
3. Generate $\mathsf{wCom}' \leftarrow \mathsf{Commit}(\mathsf{cID}, wpk', B - \epsilon; r')$ and formulate the proof:

$$\pi_2 = PK\{(wpk', B, r', \sigma_w) : \ \mathsf{wCom}' = \mathsf{Commit}(\mathsf{cID}, wpk', B - \epsilon; r') \\ \wedge\ \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\ \wedge\ 0 \leq (B - \epsilon) \leq \mathsf{val}_{\mathsf{max}}\ \}$$

$\xrightarrow{\epsilon, \mathsf{wCom}', wpk, \pi_2}$

$\xleftrightarrow{rt_{w'}}$

Verify π_2, ensure that $(wpk, \cdot) \notin \mathbf{S}$ and $\epsilon_{\mathsf{min}} \leq \epsilon \leq \epsilon_{\mathsf{max}}$. If these conditions do not hold, abort and output $\perp$. Otherwise set $\mathbf{S}_{\mathsf{new}} := \mathbf{S}_{\mathsf{old}}\ \cup\ \{(wpk, \perp)\}$. If $\epsilon < 0$, $R_{\mathcal{M}} \leftarrow 1$ otherwise $R_{\mathcal{M}} \leftarrow \perp$. Interact with the customer to provide a partially blind signature $rt_{w'}$ under sk_m on the message $(\mathsf{refund} \| wpk' \| B - \epsilon)$, where wpk' and $B - \epsilon$ are the contents of wCom'.

Compute $\mathsf{Verify}(pk_m, rt_{w'}, \mathsf{refund} \| wpk' \| B - \epsilon)$. If verification fails, or if this message does not arrive, abort and output $rt_{w'}$. Else compute $\sigma_{rev} = \mathsf{Sign}(wsk, \mathsf{revoke} \| \mathsf{cID} \| wpk)$.

$\xrightarrow{\sigma_{rev}}$

$\xleftrightarrow{\sigma_{w'}}$

Ensure $\mathsf{Verify}(wpk, \mathsf{revoke} \| wpk, \sigma_{rev}) = 1$. If so, set $\mathbf{S}_{\mathsf{new}} := \mathbf{S}_{\mathsf{old}} \cup \{(wpk, \sigma_{rev})\}$ and $R_{\mathcal{M}} \leftarrow 1$. Generate a blind signature $\sigma_{w'}$ on the contents of wCom' using sk_m. If this completes, set $R_{\mathcal{M}} \leftarrow 2$

return $w_{\mathsf{new}} := (B - \epsilon, wpk', wsk', r', \sigma_{w'})$ | return $\mathbf{S}_{\mathsf{new}}, R_{\mathcal{M}}$

Figure 4: Establishment and Payment protocols for the Bidirectional Payment Channel scheme

We present a proof of Theorem 4.1 in Appendix C.

4.2 Bidirectional payment channels

The key limitation of the above construction is that it is *unidirectional*, and only supports payments from a customer to a merchant. Additionally, it supports only fixed-value coins. In this section we describe a construction that enables bidirectional payment channels which feature compact closure, compact wallets, and allow a single run of the Pay protocol to transfer arbitrary values (constrained by a maximum payment amount).

In this construction the customer's wallet is structured similarly to the previous construction: it consists of B_0^{cust}, and a random wallet public signature key wpk. The customer first commits to these values and sends the resulting commitment to the payment network. The wallet is activated when the customer and merchant interact to provide the customer with a blind signature on its contents.

The key difference from the first protocol is that, instead of conducting the payment ϵ using a series of individual coins, each payment has the customer (1) prove that it has a valid signed wallet with balance $B^{\mathsf{cust}} \geq \epsilon$ of currency in it, and (2) request a blind signature on a new wallet for the amount $B^{\mathsf{cust}} - \epsilon$ (and embedding a fresh wallet public key wpk_{new}). Notice that in this construction the value ϵ can be positive or negative. The customer uses a zero knowledge proof and signatures with efficient protocols to prove that the contents of the new requested wallet are constructed properly, that the balances of the new wallet differs from the original balance by ϵ, and that $(B^{\mathsf{cust}} - \epsilon) \geq 0$. At the conclusion of the transaction, the customer reveals wpk_{old} to assure the merchant that this wallet cannot be spent a second time, and the old wallet is invalidated by the customer signing a "revoked" message with wsk the corresponding private key. Closing the channel consists of the customer posting a valid wallet signed by the merchant to the blockchain.[11] The challenge in this construction is to simultaneously invalidate the existing wallet and sign the new one. If the merchant signs the new wallet before the old wallet is invalidated, then the customer can retain funds in the old wallet while continuing to use the new one. On the other hand, if the merchant can invalidate the old wallet before signing the new one, the customer has no way to close the channel if the merchant refuses to sign the new wallet.

To solve this, we separate the wallet — used in interactive payments — from the value that is posted to perform channel closure and use a two phase protocol to obtain each of these values. Instead of revealing the most recent wallet w, $\mathcal{C}$ closes the channel using a *refund token* rt which specifies B^{cust}, the current wallet's public key, and a signature by the merchant. In phase one of Pay, the customer first obtains a signature on the refund token blindly from M. In the second phase, the customer invalidates the old wallet, and then the merchant signs the new wallet. If the merchant refuses to sign the new wallet, the customer can safely close the channel using rt.

We now describe the revised scheme. The protocols $\mathsf{Establish}$ and Pay are presented in Figure 4. The Setup and $\mathsf{Init}_{\mathcal{M}}$ algorithms are identical to the previous construction.

- $\mathsf{KeyGen}(\mathsf{pp})$. Compute $(pk, sk) \leftarrow \Pi_{\mathsf{sig}}.\mathsf{SigKeygen}(1^\lambda)$.
- $\mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, \mathsf{cID}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_c, sk_c)$. The customer generates the wallet commitment by sampling random coins r, computing an ephemeral keypair $(wpk, wsk) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and produc-

[9] For example, the necessary properties can be achieved using a secure commitment scheme and any secure symmetric encryption mechanism.

[10] For simplicity of exposition, we assume that pk can be derived from sk

[11] In the special case where the customer has not obtained a signature on the wallet from the merchant (*e.g.,* because the merchant never accepted the channel opening), it can simply post an opening of the wallet commitment.

ing a commitment $\mathsf{wCom} = \mathsf{Commit}(\mathsf{cID}, wpk, B_0^{\mathsf{cust}}; r)$. It outputs the token $\mathsf{T}_{\mathcal{C}} = (pk_c, \mathsf{wCom})$ and retains the secrets $csk_{\mathcal{C}} = (\mathsf{wCom}, sk_c, \mathsf{cID}, wpk, wsk, r, B_0^{\mathsf{cust}})$.

- $\mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B_0^{\mathsf{cust}}, B_0^{\mathsf{merch}}, pk_m, sk_m)$. Output $\mathsf{T}_{\mathcal{M}} = pk_m$, $csk_{\mathcal{M}} = (sk_m, B_0^{\mathsf{cust}})$.

- $\mathsf{Refund}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk_{\mathcal{C}}, w)$. If the customer has not yet invoked the Pay protocol, it sets $m := (\mathsf{refundUnsigned}, (\mathsf{cID}, wpk, B), r)$. Otherwise set $m := (\mathsf{refundToken}, (\mathsf{cID}, wpk, B), rt_w)$. Compute $\sigma = \mathsf{Sign}(sk_c, m)$. Output $\mathsf{rc}_{\mathcal{C}} = (m, \sigma)$.

- $\mathsf{Refute}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathbf{S}, \mathsf{rc}_{\mathcal{C}})$. If a merchant sees a channel closure message, it first parses $\mathsf{T}_{\mathcal{C}}$ to obtain pk_c. It parses $\mathsf{rc}_{\mathcal{C}}$ as (m, σ) and verifies the signature σ using pk_c. If this signature verifies, it parses m to obtain (cID, wpk, B) and verifies that cID matches the channel. Finally, if it has previously stored (wpk, σ_{rev}) in its state $\mathbf{S}$ then it outputs $\mathsf{rc}_{\mathcal{M}} = ((\mathsf{revoked}, \sigma_{rev}), \sigma)$ where σ is a valid signature on the message $(\mathsf{revoked}, \sigma_{rev})$ under sk_m. Otherwise it adds the new key wpk to its state $\mathbf{S}$.

- $\mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$.

  Verify that both $\mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}}$ are correctly signed by the customer and merchant keys pk_c and pk_m respectively. Verify that both tokens contain the same cID and this matches the channel identifier from $\mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}$. If either of the tokens fails this test, replace it with $\perp$. Let $B_{\mathsf{total}} = B_0^{\mathsf{cust}} + B_0^{\mathsf{merch}}$. If $\mathsf{rc}_{\mathcal{C}}$ is $\perp$, simply output all funds to the merchant.

1. Parse $\mathsf{T}_{\mathcal{C}}$ to obtain (pk_c, wCom).
2. Parse m as $(\mathsf{type}, (\mathsf{cID}, wpk, B), \mathsf{Token})$.
3. Parse m as $(\mathsf{revoked}, wpk, \sigma_{rev})$. Check that $\mathsf{Verify}(wpk, (\mathsf{revoke} \| \mathsf{cID} \| wpk)\sigma) = 1$. If any check fails, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = B_{\mathsf{total}}$ and $B_{\mathsf{final}}^{\mathsf{merch}} = 0$.
4. Perform the following checks:
   (a) Check the refund's validity: If type is $\mathsf{refundUnsigned}$, check that $\mathsf{wCom} = \mathsf{Commit}(\mathsf{cID}, wpk, B, \mathsf{Token})$. If the merchant's token contains σ_{rev}
   Otherwise type is $\mathsf{refundToken}$, so check that Token is a valid refund token on (cID, wpk, B). If either check fails, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = 0$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}}$.
   (b) Check the refutation's validity: and check $\mathsf{Verify}(wpk, \mathsf{revoke} \| wpk, \sigma_{rev}) = 1$. If so, terminate and output $B_{\mathsf{final}}^{\mathsf{cust}} = 0$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}}$. Otherwise return $B_{\mathsf{final}}^{\mathsf{cust}} = B$ and $B_{\mathsf{final}}^{\mathsf{merch}} = B_{\mathsf{total}} - B$ (i.e. pay the claimed B to $\mathcal{C}$ and the remainder to $\mathcal{M}$).

4.2.1 Security Analysis

As we noted in §1.3, the main limitation of the bidirectional protocol is the possibility that a malicious merchant may abort the protocol. The nature of the protocol ensures that a customer is not at risk of losing funds due to such an abort, since she may simply provide her refund token $rt_{w'}$ to the payment network in order to recover her balance. The main limitation therefore is to the customer's anonymity. A malicious merchant can place a customer into a situation where she cannot continue to spend, and must close her channel. This implicitly links the payment to the channel – a matter that is of only limited concern, if the channel is funded with anonymous currency.

Of more concern is the possibility that a malicious merchant will use aborts to reduce the anonymity set of the system, by causing several channels to enter a non-functional state. In practice, this attack will produce a visible signal at the payment network, allowing customers to halt use of the channel. However, within the context of our security proof we address this in a simpler way, by simply preventing the adversarial merchant from aborting during the Pay protocol.

Theorem 4.2. *The bidirectional channel scheme satisfies the properties of* anonymity *and* balance *under the restriction that the adversary does not abort during the* Pay *protocol, and the assumption that* (1) *the commitment scheme is secure,* (2) *the zero-knowledge system is simulation extractable and zero-knowledge,* (3) *the blind signature scheme is existentially unforgeable under chosen message attack, and* (4) *the one time signature scheme is existentially unforgeable under one time chosen message attack.*

We include a proof of Theorem 4.2 in Appendix D.

4.3 Bidirectional Third Party Payments

One of the main applications of the bidirectional construction above is to enable *third party payments*. In these payments, a first party **A** makes a payment of some positive value to a second party **B** via some intermediary **I** with whom both **A** and **B** have open channels. In this case, we assume that both **A** and **B** act as the customer for channel establishment, and **I** plays the role of the merchant. Our goal is that **I** does not learn the identities of the participants, or the amount being transferred (outside of side information she can learn from her channel state), nor should she trusted to safeguard the participants' funds. This construction stands in contrast to existing non-anonymous payment channel schemes [PD16, DW15] where given the chain $\mathbf{A} \rightarrow \mathbf{I} \rightarrow \mathbf{B}$, the intermediary always learns both the amount and the participants.

The challenge in chaining payment channels is to make the payments *atomic*. That is, the payer **A** only wants to pay the intermediary **I** once **I** has paid the recipient **B**. But of course this places the intermediary at risk if **A** fails to complete the payment. Similarly, the payer risks losing her funds to a dishonest intermediary. There is no purely cryptographic solution to this problem, since it's in essence fair exchange — a problem that has been studied extensively in multi-party protocols. However, there are known techniques for using blockchains to mediate aborts [BK14, ADMM14]. This is our approach as well.

Recall from §4.2 that the Pay protocol occurs in two phases. The first portion is an exchange of *refund tokens* that can be used to reclaim escrowed funds. The second phase generates an anonymous wallet for subsequent payments. For a chained transaction from $\mathbf{A} \rightarrow \mathbf{I} \rightarrow \mathbf{B}$ to be secure, we need only ensure that the first phase of both legs completes or fails atomically.

We accomplish this by adding conditions to the refund tokens. These conditions are simple statements for the network to evaluate on examining a token during the Resolve protocol. Specifically, to prevent the recipient **B** from claiming funds from **I** if the payer **A** has not delivered a corresponding payment to **I**, we introduce the following conditions into **B**'s refund token:

1. **B** must produce a revocation message (*i.e.* a signature using **A**'s wsk) on **A**'s previous wallet.
2. **A** has not posted a revocation token containing wsk to the ledger.

By condition (1), once **B** forces a payment on $\mathbf{I} \rightarrow \mathbf{B}$, $\mathbf{A} \rightarrow \mathbf{I}$ cannot be reversed since **I** has the revocation token. By condition (2) if $\mathbf{A} \rightarrow \mathbf{I}$ has been already been reversed, **B** cannot force the payment $\mathbf{I} \rightarrow \mathbf{B}$ since wpk is already on the ledger.

Hiding the payment amount. Our third-party payment construction also provides an additional useful feature. Since **I** acts only as a passive participant in the transaction and does not maintain state for either channel, there is no need for for **I** to learn the amount being paid. Provided that both **A** and **B** agree on an amount ϵ (*i.e.,* both parties have sufficient funds in each of their channels), neither party need reveal ϵ to **I**: **I** need merely be assured that the balance of funds is conserved.

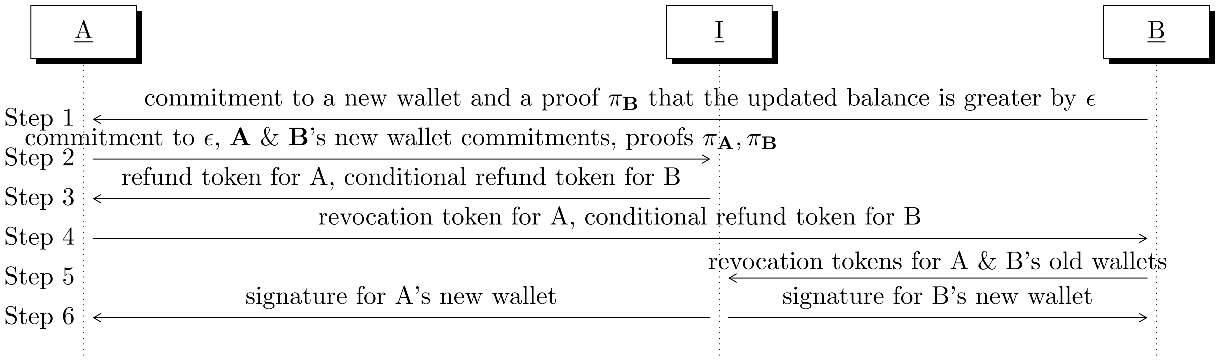

Figure 5: Outline of our third-party payments protocol. In practice, **A** can route all messages from **B** to **I**.

To hide the payment amount, we must modify the proof statement used to construct π_2 from the Pay protocol of Figure 4. Rather than revealing ϵ to the merchant, the customer **A** now commits to ϵ and uses this value as a secret input in computing the payment. Simultaneously, in the payment protocol conducted to adjust **B**'s wallet, **B** now proves that his wallet has been adjusted by $-\epsilon$.

To do this, we change the proof in the pay protocol to one that binds ϵ to a commitment but does not reveal it:

$$\begin{aligned}\pi_2 = PK\{(wpk', &B, r', \sigma_w, \epsilon, r_\epsilon) : \\ \mathsf{wCom}' = \ &\mathsf{Commit}(wpk', B - \epsilon; r') \\ &\wedge \mathsf{Verify}(pk_m, (wpk, B), \sigma_w) = 1 \\ &\wedge \mathsf{vCom} = \mathsf{Commit}(\epsilon, r_\epsilon) \\ &\wedge 0 \leq (B - \epsilon) \leq \mathsf{val}_{\mathsf{max}} \ \}\end{aligned}$$

A can then prove to **I** that the two payments cancel or (if fee is non-zero), leave B with fee extra funds via a proof:

$$\begin{aligned}\pi_\epsilon = PK\{(\epsilon_{\mathbf{A}}, \epsilon_{\mathbf{B}}, r_{\epsilon_{\mathbf{A}}}, r_{\epsilon_{\mathbf{B}}}) : &\mathsf{vCom}_{\epsilon_A} = \mathsf{Commit}(\epsilon_{\mathbf{A}}; r_{\epsilon_{\mathbf{A}}}) \\ &\wedge \mathsf{vCom}_{\epsilon_{\mathbf{B}}} = \mathsf{Commit}(\epsilon_{\mathbf{B}}; r_{\epsilon_{\mathbf{B}}}) \\ &\wedge \epsilon_{\mathbf{A}} < \epsilon_{\mathsf{max}} \wedge -\epsilon_{\mathbf{B}} < \epsilon_{\mathsf{max}} \\ &\wedge \epsilon_{\mathbf{A}} + \epsilon_{\mathbf{B}} = fee\end{aligned}$$

The protocol. We now combine the process of updating both **A** and **B**'s wallet into a single protocol flow, which we outline in Figure 5. In detail, the steps are as follows:

1. **B** commits to ϵ and conducts the first move of the variable payment Pay protocol (Figure 4) (with the modified balance-hiding proof described above) and sends a commitment to its new wallet state wCom'_b, proof of correctness for the wallet, $\pi_{\mathbf{B}}$, and commitment randomness to **A**.

2. **A** completes it's own first move, generating $\mathsf{wCom}'_a, \pi_{\mathbf{A}}$ and additionally computes $\pi_{\mathbf{A}}$ attesting to the correct state of its original wallet and new wallet commitment. It sends these and **B**'s new wallet commitment and $\pi_{\mathbf{A}}$ to **I**.

3. **I**, after validating the proofs, issues **A** a refund token for its new wallet $rt_{w'_a}$ and **B** a conditional refund token $crt_{w'_b}^{\sigma_{rev}^{w_a}}$ as its new wallet. This token embeds the condition that **B** must produce a revocation token for **A**'s old wallet and that **A** must not have closed the channel already.

4. **A** completes its second move in the variable payment Pay protocol to generate $\sigma_{rev}^{w_a}$ the revocation token for its old wallet. It sends that and the $crt_{w'_b}^{\sigma_{rev}^{w_a}}$ to **B**.

5. **B** completes its second move to generate $\sigma_{rev}^{w_b}$ the revocation token for its old wallet. After validating that it now has a valid refund token by verifying $\sigma_{rev}^{w_a}$, it sends $\sigma_{rev}^{w_a}, \sigma_{rev}^{w_b}$ to **I**.

6. **I** completes the remaining moves of the variable payment Pay protocol with **A** and **B** individually, giving each a blind signature on their new wallets.

Security and abort conditions. We provide a proof sketch for balance preservation in Appendix F.

In terms of anonymity, the execution of this protocol is no different in terms of the information revealed than two in parallel payments from $\mathbf{A} \rightarrow \mathbf{I}$ and $\mathbf{I} \rightarrow \mathbf{B}$. Our payment anonymity definition already allows this type of attack even for the two party case.

The main challenge in realizing this construction is the possibility that a malicious **I** can selectively abort the protocol during a transaction. This does not allow **I** to steal funds, but it does force **A** and **B** to transmit messages to the network in order to recover their funds. This potentially links the payment attempt to **A** and **B**'s channels. Unfortunately, this seems fundamentally difficult to avoid in an interactive protocol.

We note that the anonymity threat is limited in practice by the fact that the channels themselves can be funded with an anonymous currency (*e.g.,* [MGGR13, DFKP13, SCG+14]), so linking two separate channels does not reveal the participant identifiers. More importantly, since the intermediary can use this abort technique only one time during the lifetime of a channel, there is no possibility for the merchant to link *separate* payments on the same channel. Finally, an intermediary who performs this abort technique will produce public evidence on the network, which allows participants to avoid the malicious intermediary.

4.4 From Third Party Payments to Payment Networks

It should be possible to extend the above protocol to allow payments of the form $\mathbf{A} \rightarrow \mathbf{I}_1 \rightarrow \ldots \rightarrow \mathbf{I}_n \rightarrow \mathbf{B}$ via techniques similar to those used in non-anonymous payment channel networks [PD16]. As discussed in §1.2, it is not possible to hide channel balances in such a setting.

The general approach is as follows: we use "hash locks" to enforce that either all refund and revocation tokens are valid or none are. Specifically, we attach to both the refund and revocation tokens a condition that they can only be used if one party reveals x such that $y = \mathcal{H}(x)$, where x is picked by **A**. Because if one party releases x, all parties may close their channels, this forces the entire sequence of payments to either go through or not. As with Lightning, the timeouts for each channel must be carefully chosen.

An immediate objection is that x identifies all parties. However, there exist proposals[Tow15] to make chains unlinkable. In essence each channel is locked not under a single global preimage but under a unique one encrypted under the neighbor's preimages. Thus one preimage being posted enables a cascade of decryptions and closures.

We note that while this approach allows payments over multiple hops, if all but one party is dishonest, the anonymity set is only that of the link involving the honest client. We leave the exact details of this approach to future work.

	Customer		Merchant		
primitive	Establish(ms)	Pay(ms)	Setup(ms)	Establish(ms)	Pay(ms)
Bilinear CL-Sigs[CL04]	8.07 ± 0.13	100.13 ± 1.60	1433.51 ± 23.69	15.87 ± 0.27	82.32 ± 1.37
Algebraic MACs[CMZ14]	6.90 ± 0.17	37.61 ± 0.93	826.78 ± 19.26	11.97 ± 0.31	34.39 ± 0.88

Figure 6: Performance comparison of different implementations of BOLT bidirectional payment protocol. 1000 iterations on a single core of a Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.10GHz. Customer setup is included in Establish.

4.5 Hiding Channel Balances

Each of the constructions presented above has a privacy limitation: the balance of each payment channel is revealed when a channel is closed. While individuals can protect their identities and initial channel balances by using an anonymous currency mechanism to fund channels, the information about channel balances leaks useful information to the network. We note, however, that in the case of *non-disputed* channel closure, even this information can be hidden from the public as follows. On channel closure, the customer posts a commitment to the final channel balance, along with a zero-knowledge proof that she possesses a valid channel closure token (i.e., a signature on the channel balance in our bidirectional construction). In systems such as Zerocash [SCG+14], the final payment redeeming coins to the merchant and customer can be modified to include an additional statement: *the amounts paid in this transaction are consistent with the commitment, and do not exceed the initial funding transaction that created the channel.* We leave the precise details of such a construction to future work.

5 Implementation of the Bidirectional scheme

We now detail the integration of Bolt into a cryptocurrency and performance and cryptographic details of a concrete instantiation.

5.1 Integration with a Currency

In this section we consider the problem of integrating the bidirectional Bolt protocol into a Bitcoin like cryptocurrency in a *soft fork*: a protocol change which does not break backward compatibility with existing nodes. Recall that the bidirectional scheme requires that the channels be funded anonymously in order to protect against aborts linking the aborted payment to the channel opening (this does not hold if one wishes merely to prevent multiple payments on the same channel from being linked together). In these conditions, the anonymity of the payment channel is no better than the anonymity of the underlying cryptocurrency. Of the Bitcoin derived currencies, Zerocash and ZCash [SCG+14, zca17] provide a strong underlying anonymity layer. Anonymity tools for Bitcoin, such as Coinjoin [Max13], may also be sufficient in some circumstances and future improvements to Bitcoin may increase the achievable anonymity. The mechanism for deployment is compatible with either currency.

In Bitcoin and ZCash each transaction[12] consists of a set of inputs and a set of outputs. Inputs reference a previous transaction output and contain a `ScriptSig` authorizing use of the funds. Outputs specify the amount of the output and a `ScriptPubKey` specifying when the output can be spent. To evaluate a transaction the `ScriptPubKey` from the previous transaction and `ScriptSig` from the current transaction are combined and evaluated using a stack-based scripting language.

[12]We refer here to the "unshielded" transactions in ZCash. Shielded transactions function differently.

In the simplest case, `ScriptPubKey` requires a signature under a specified public key to spend the funds and `ScriptSig` contains such a signature. However, more complex scripts are allowed including control flow such as if statements, time locks that enforce that a given number of blocks has elapsed since the transaction was created, and threshold signatures. As long as the combined script evaluates to True, spending is authorized.

Our soft fork approach involves adding a single opcode, `OP_BOLT` to the scripting language. This opcode has the power to 1. validate the commitment opening and blind signature on the commitment in a refund token, and 2. inspect the output of the transaction and enforce restrictions on it.

Most opcodes do not inspect transaction outputs. The notable exception to this rule are signature opcodes that may hash the entire transaction, including both inputs and outputs, in order to verify the signature. However, it is entirely possible to modify the ZCash and Bitcoin codebase to enable opcodes that do have access to transaction outputs in general.[13] Specifically, our new opcode will enforce two constraints on the outputs of a transaction closing the channel:

1. Verifying that there are two outputs: one paying the merchant his balance and the other paying the customer hers.
2. Verifying that the customer's payout is time locked such that it can be claimed by the merchant if the refund token has been invalidated (i.e. the customer tried to close on an old channel state).

To accomplish this, we implement the Pay protocol so that the revocation token is the private key corresponding to a Bitcoin (resp. transparent ZCash) address. When a channel is closed, the merchant's fraction of the channel balance is paid in a transaction that is spendable immediately. However, the customer's funds are not immediately spendable by the customer since we need to allow the merchant to dispute the closure. The merchant does this by signing a transaction with both his key and the revocation token key. If the merchant does not do so, the customer can claim the funds after some elapsed time. The script is given below:

```
OP_IF # If merchant
    OP_2 <rev-pubkey><merchant-pubkey> OP_2
    OP_CHECKMULTISIG #2 of 2 multi-sig check
OP_ELSE #If customer wait
    <delay> # delay to wait
    OP_CSV # Timelock enforces delay
    OP_DROP
    <customer-pubkey> # key for customers funds
    OP_CHECKSIG
OP_ENDIF
```

This approach greatly simplifies the implementation. Channel opening consists of posting a transaction with (previously anonymized) inputs containing the needed funds for escrow and a single output. This outputs `ScriptPubKey` contains the new opcode `OP_BOLT` and the channel parameters as one argument. To close the channel, the customer posts the appropriate transaction spending that output with a `ScriptSig` that contains what is required to satisfy `OP_BOLT`: the refund token and two outputs. One output for the merchant's share of the channel which is spendable

[13] In systems derived from Bitcoin core's source code, this requires some modification as the current architecture abstraction in script evaluation provides a callback to get the transaction hash, not direct access to the transaction itself. However this is not a protocol limitation and indeed past versions of the codebase did expose direct access. Exposing direct access does not affect consensus rules in and of itself.

immediately and one with the customer's spendable under the above time locked script. Finally, each party can post the appropriate transaction claiming their output. The merchant can dispute the channel closure and claim the funds immediately with `OP_FALSE <sig revocation-pubkey> <sig merchant-pubkey>`. On the other hand, the customer must wait and then claim with `OP_TRUE <sig customer-pubkey>`.

Simplified resolution At the cost of an extra round trip in the Pay protocol, we can eliminate the need to validate "blind" signatures in the resolution phase, instead opting to verify a simple commitment opening and a standard (e.g. ECDSA) signature on that commitment. We do this by having the refund token consist of a standard signature on the wallet commitment. Because it is a standard signature, refund token issuance can be linked to usage. This is not a problem if the token is used to handle an abort in the same protocol run as its issuance—our security model assumes the attacker can link a single transaction to channel open/closure. However, it cannot be safely used after that, i.e. in the first step of the next pay protocol, because the unblinded signature will link the current execution of the protocol to the previous one. To solve this, at the start of every run of the pay protocol, we request a fresh refund token on the current wallet before revealing anything. Because the contents of the refund token commitment are deterministically and provably generated from the existing wallet, issuing multiple of them has no other effect. However, it eliminates the need to ever use the old refund token. The only consequence is an additional round trip as we must wait for the refreshed refund token before we can publish.

5.2 Implementation

We provide two constructions of Bolt, one using signatures with efficient protocols [CL04] and the other using using Algebraic MACs [CMZ14]. We defer further discussion of primitive selection and usage to Appendix A and move directly to presenting performance numbers in Figure 6.

6 Related Work

Anonymity and scaling for Bitcoin. A number of works have proposed additional privacy protections for Bitcoin. Zerocoin, Zerocash and similar works [MGGR13, SCG$^+$14] provide strong anonymity through the use of complex zero knowledge proofs. A separate line of works seek to increase anonymity by Bitcoin by mixing transactions (e.g. CoinJoin [Max13], CoinShuffle, CoinSwap). Like Bitcoin, each of these constructions require that all transactions are stored on the blockchain. Finally, recent work has proposed *probabilistic payments* as an alternative payment mechanism [Ps15].

Privacy in payment channels As discussed in detail in the introduction, Heilman *et al.* [HBG16] construct off-chain payments with 3$^{\text{rd}}$ party privacy.

Lightning anonymity limitations. The Lightning Network [PD16] does not provide payment anonymity between pairs of channel participants – *i.e.,* a merchant can see the channel identity of every customer that initiates a payment. However, the protocol includes some limited anonymity protections for *path payments.* These operate on a principle similar to an onion routing network, by using multiple non-colluding intermediaries to obscure the origin and destination of a path. Unfortunately this proposal suffers from collusion problems: given the chain $A \to I_1 \to I_2 \to I_3 \to B$, only I_1 and I_3 must collude to recover the identities of A and B, since all transactions on the path share the same Hash Timelock Contract ID. Moreover, this security mechanism assumes there exist a network with sufficient path diversity for these protections to be viable. The practical viability of path routing in the Lightning payment network is a subject of some debate given the large amount of funds that would be tied up in maintaining open channels [Rat16, Pac15]. It seems more likely that deployed channels will rely on a star topology where clients and merchants interact via a one of a handful of highly-available parties, which is the situation we address in our constructions.

7 Acknowledgments

This research was supported in part by the National Science Foundation under awards CNS-1010928, CNS-1228443, and EFMA-1441209; The Office of Naval Research under contract N00014-14-1-0333; and the Mozilla Foundation.

References

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on Bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, 2014.

[AG17] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`, 2017.

[AGM+13] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC 2008*, 2008.

[BCKL09] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact E-Cash and Simulatable VRFs Revisited. In *Pairing-Based Cryptography '09*, 2009.

[BDJR97] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science '97*, pages 394–403, 1997.

[bit17] Bitcoin Wiki: Maximum transaction rate. `https://en.bitcoin.it/wiki/Maximum_transaction_rate`, 2017.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use Bitcoin to design fair protocols. In *Advances in Cryptology–CRYPTO 2014*. 2014.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1087–1098. ACM, 2013.

[Blo14] Block Chain Analysis. Block chain analysis. `http://www.block-chain-analysis.com/`, 2014.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology EUROCRYPT 2000*, 2000.

[Bra93] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology CRYPTO 93*, 1993.

[Bra97] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT '97*, 1997.

[Bun17] JP Buntinx. Bitcoin Network Backlog Grows To Over 165,000 Unconfirmed Transactions. 2017.

[CC+08] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 234–252. Springer, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, 1994.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, 1990.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology*, 1983.

[Cha15] Chainalysis. Chainalysis inc. https://chainalysis.com/, 2015.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Advances in Cryptology–EUROCRYPT 2005*. 2005.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in communication networks*. 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology–CRYPTO 2004*, 2004.

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1205–1216. ACM, 2014.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT '07*, 2007.

[CS97] Jan Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Annual International Cryptology Conference*, pages 126–144. Springer, 2003.

[Dan] George Danezis. petlib: A python library that implements a number of Privacy Enhancing Technolgies. https://github.com/gdanezis/petlib.

[DFKP13] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: Building Zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies*, PETShop '13, 2013.

[DW15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with Bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, 2015.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC '05*, pages 416–431, 2005.

[Ell13] Elliptic. Elliptic enterprises limited. https://www.elliptic.co/, 2013.

[Gro06] Jens Groth. *Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures*, pages 444–459. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[GS] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In *NDSS 2017*, 2017.

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions. In *BITCOIN '16*, 2016.

[Lyn15] Ben Lynn. PBC:The Pairing-Based Cryptography Library. https://crypto.stanford.edu/pbc/, 2015.

[Max13] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. Available at https://bitcointalk.org/index.php?topic=279249.0, August 2013.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, 2013.

[MMLN17] Andrew Miller, Malte Moeser, Kevin Lee, and Arvind Narayanan. An empirical analysis of linkability in the monero blockchain. Available at https://arxiv.org/abs/1704.04299, 2017.

[mon17] The Monero Currency. Available at https://getmonero.org/, 2017.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, 2013.

[Pac15] Chris Pacia. Lightning Network skepticism. https://chrispacia.wordpress.com/2015/12/23/lightning-network-skepticism, December 2015.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, January 2016.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, 1992.

[Ps15] Rafael Pass and abhi shelat. Micropayments for decentralized currencies. In *ACM CCS '15*, pages 207–218, New York, NY, USA, 2015. ACM.

[Rat16] John Ratcliff. The Lightning Network is so great that it has all kinds of problems. `http://codesuppository.blogspot.com/2016/02/the-lightning-network-is-so-great-that.html`, February 2016.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography '13*, 2013.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Security and Privacy*, 2014.

[Sch91] Claus-Peter Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 1991.

[Tow15] Anthony Towns. Better privacy with SNARKs. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2015-November/000309.html`, November 2015.

[zca17] The zcash currency. Available at `https://z.cash/`, 2017.

A Choice of cryptographic primitives

We now describe in depth our choice of cryptographic primitives:

A.1 Possible building blocks

Signatures with efficient protocols are the core building block of anonymous credentials and are a well studied primitive with many solutions offering various performance, security, and feature trade offs. One of the most efficient schemes that offers a full set of features and provable security is the bilinear variant of CL-signatures due to Camenisch and Lysyanskaya [CL04]. An implementation exists in Charm [AGM+13].

We are aware of two other candidate signature schemes with available implementations from petLib [Dan] that are aimed at providing increased performance with reduced functionality. The first is used in the construction of Lightweight Anonymous Credentials [BL13]. Here signatures can only be shown anonymously once. Second, Algebraic MACs are used in [CMZ14], to build a limited form of anonymous credential. Because it uses a MAC not a signature, only the issuer can verify "signed" messages. This requires some modification to our protocol since closure of a channel currently requires public verification of the refund token.

A.2 Selecting the signature scheme

The scheme from Anonymous Credentials Light [BL13] is the fastest for issuing and showing, with most operations taking less than $0.01ms$. However, a registration phase must be completed for the set of messages that can be signed. This must be repeated every time the set changes and takes $100ms$. Because the refund token rt is selected at random and changes on every instance, this process must be done on every payment. Moreover, even if this were made far faster, the registration process reveals the message set. It may be possible to patch Bolt to accommodate this or modify the credential scheme to remove the restriction, but the $100ms$ cost is too high to pay per payment. The remaining two schemes are more promising with most operations taking less than $30ms$ each.

A.3 Implementation

We build two completely distinct implementations of the bidirectional payment protocol. One using bilinear CL-Sigs and the other using Algebraic MACs. Our approach mirrors the construction of a credential scheme: we present a commitment to the wallet and a proof that it is signed and then use Schnorr proofs [Sch91] to prove the balance of the new wallet commitment is correct with respect to the old wallet. We then blindly obtain a signature on the new wallet. For the range proof, we use a technique reminiscent of [CC+08]: we decompose the balance into bytes and prove we have a signature issued on each byte. This allows us to reuse the code and primitives from the signature scheme rather than using a separate range proof which would introduce more cryptographic assumptions, more code, and more dependencies.

The results are given in Table 6. The implementation based on Algebraic MACs is approximately twice as fast as the CL-Sig approach. It should be noted, however, that there is far more room for optimization in the CL-signature library. While both are implemented in Python, the implementation of Algebraic MACs use only elliptic curve operations via openSSL. As such, the principle overhead is from calling native code from Python. On the other hand, the CL-signature implementation uses symmetric bilinear pairings with an implementation from the PBC library[Lyn15]. Use of asymmetric pairings and a faster pairing library such as RELIC [AG17] would give a marked improvement.

A.4 Adapting channel closure to avoid public verification of credentials

Closing a disputed channel currently requires the blockchain to verify that the refund token is signed. For our faster construction, this is impossible since the key remains secret and the "signature" is actually a MAC. There are two solutions to this: 1) we can, as outlined in paragraph 5.1 opt to have the blockchain validate conventional signatures at the cost of an extra round trip in pay. 2) We can allow the merchant to prove that a purported MAC is invalid.

The MAC itself consists of $u, u' = u^{\mathcal{H}_x(m)}$ where $\mathcal{H}_x(m)$ is a keyed and deterministic hash function. Unfortunately, u is chosen at random so the MAC is not unique and it is not sufficient to reveal the correct MAC on the message and prove its correctness.[14] Instead, we must prove that $log_u(u') \neq log_v(v')$ (i.e. that $u_x^{\mathcal{H}}(m) \neq u'$) and that the revealed MAC $v, v')$ is correct. Camenisch and Shoup give an extremely efficient proof for discrete log inequality [CS03] where only one discrete log is known to the prover and none known to the verifier. We implement this full proof of invalid MAC combining the prove of MAC validity and discrete log inequality. It takes approximately $14ms$ to generate and verify. We note that as this proof includes the actual valid MAC on the forged refund token, it is necessary for the blockchain to blacklist this MAC and not accept it. However, since the refund token can never be used in payments, we need not add extra steps to the pay protocol.

B Security Definitions

In this section we provide formal security definitions for an anonymous payment channel scheme.

B.1 Payment anonymity

Let $\mathcal{A}$ be an adversary playing the role of merchant. We consider an experiment involving P "customers", each interacting with the merchant as follows. First, $\mathcal{A}$ is given pp, then outputs $\mathsf{T}_{\mathcal{M}}$. Next $\mathcal{A}$ issues the following queries in any order:

[14] Counter-intuitively, despite being built on a MAC, Keyed-Verification Anonymous Credentials include an efficient zk-proof of validity of a MAC that effectively transforms the MAC into a (non blind) signature. Since this proof is somewhat expensive, it is only used to verify the correctness of issued credentials.

Initialize channel for $\mathcal{C}_i$. When $\mathcal{A}$ makes this query on input $B^{\mathsf{cust}}, B^{\mathsf{merch}}$, it obtains the commitment $\mathsf{T}^i_{\mathcal{C}}$, generated as $(\mathsf{T}^i_{\mathcal{C}}, csk^i_{\mathcal{C}}) \xleftarrow{R} \mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B^{\mathsf{cust}}, B^{\mathsf{merch}})$.

Establish channel with $\mathcal{C}_i$. In this query, $\mathcal{A}$ executes the $\mathsf{Establish}$ protocol with $\mathcal{C}_i$ as:

$$\mathsf{Establish}(\{\mathcal{C}(\mathsf{pp}, \mathsf{T}_{\mathcal{M}}, csk^i_{\mathcal{C}})\}, \{\mathcal{A}(state)\}$$

Where *state* is the adversary's state. Let us denote the customer's output as w_i, where w_i may be $\perp$.

Payment from $\mathcal{C}_i$. In this query, if $w_i \neq \perp$, then $\mathcal{A}$ executes the Pay protocol for an amount ϵ with $\mathcal{C}_i$ as:

$$\mathsf{Pay}(\{\mathcal{C}(\mathsf{pp}, \epsilon, w_i)\}, \{\mathcal{A}(state)\})$$

Where *state* is the adversary's state. We denote the customer's output as w_i, where w_i may be $\perp$.

Finalize with $\mathcal{C}_i$. When $\mathcal{A}$ makes this query, it obtains the closure message $\mathsf{rc}^i_{\mathcal{C}}$, computed as $\mathsf{rc}_{\mathcal{C}} \xleftarrow{R} \mathsf{Refund}(\mathsf{pp}, w_i)$.

We say that $\mathcal{A}$ is *legal* if $\mathcal{A}$ never asks to spend from a wallet where $w_i = \perp$ or where w_i is undefined, and where $\mathcal{A}$ never asks $\mathcal{C}_i$ to spend unless the customer has sufficient balance to complete the spend.

Let *auxparams* be an auxiliary trapdoor not available to the participants of the real protocol. We require the existence of a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the following two experiments with non-negligible advantage:

Real experiment. In this experiment, all responses are computed as described above.

Ideal experiment. In this experiment, the Commitment, Establishment and Finalize queries are handled using the procedure described abvove. However, in the Payment query, $\mathcal{A}$ does not interact with $\mathcal{C}_i$ but instead interacts with $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, auxparams, \cdot)$.

As in [CHL05] we note that this definition preserves anonymity because the simulator $\mathcal{S}$ does not know the identity of the user i for which he is spending the coin.

B.2 Payment Balance

$\mathcal{A}$ interacts with a collection of P honest customers $\mathcal{C}_1, \ldots, \mathcal{C}_P$ and Q honest merchants $\mathcal{M}_1, \ldots, \mathcal{M}_Q$. Initialize the counters $\mathsf{bal}_{\mathcal{A}} \leftarrow 0, \mathsf{claimed}_{\mathcal{A}} \leftarrow 0$. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. For each merchant $i \in [1, Q]$, at the start of the game let $(pk_{\mathcal{M}_i}, sk_{\mathcal{M}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Give pp and $(pk_{\mathcal{M}_1}, \ldots, pk_{\mathcal{M}_Q})$ to $\mathcal{A}$. Now $\mathcal{A}$ may issue the following queries in any order:

Initialize channel for $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) When $\mathcal{A}$ makes this query on input $(\mathcal{P}_i, B^{\mathsf{cust}}_0, B^{\mathsf{merch}}_0)$, it obtains the commitment $\mathsf{T}_{\mathcal{C}_i}$ (resp. $\mathsf{T}_{\mathcal{M}_i}$) computed as follows:

- If the party $\mathcal{P}_i$ is a customer: First compute $(pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then $(\mathsf{T}_{\mathcal{C}_i}, csk^i_{\mathcal{C}}) \xleftarrow{R} \mathsf{Init}_{\mathcal{C}}(\mathsf{pp}, B^{\mathsf{cust}}_0, B^{\mathsf{merch}}_0, pk_{\mathcal{C}_i}, sk_{\mathcal{C}_i})$. Set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + B^{\mathsf{merch}}_0$.
- If the party $\mathcal{P}_i$ is a merchant: Compute $(\mathsf{T}_{\mathcal{M}_i}, csk_{\mathcal{M}_i}) \xleftarrow{R} \mathsf{Init}_{\mathcal{M}}(\mathsf{pp}, B^{\mathsf{cust}}_0, B^{\mathsf{merch}}_0, pk_{\mathcal{M}_i}, csk^i_{\mathcal{M}})$. Set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + B^{\mathsf{cust}}_0$.

Establish channel with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$). When $\mathcal{A}$ specifies $(\mathcal{P}_i, \mathsf{T}_{\mathcal{A}})$, and $\mathcal{A}$ has previously initialized a channel with party $\mathcal{P}_i$, execute the Establish protocol with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) using the following input:

- If $\mathcal{P}_i$ is a customer: $\mathsf{Establish}(\{\mathcal{C}_i(\mathsf{pp}, \mathsf{T}_{\mathcal{A}}, csk^i_{\mathcal{C}})\}, \{\mathcal{A}(state)\} \rightarrow w_i$ (or $\perp$).
- If $\mathcal{P}_i$ is a merchant: $\mathsf{Establish}(\{\mathcal{A}(state)\}, \{\mathcal{M}(\mathsf{pp}, \mathsf{T}_{\mathcal{A}}, csk^i_{\mathcal{M}})\} \rightarrow \mathsf{established}$ (or $\perp$).

Where *state* is the adversary's state.

Payment from $\mathcal{C}_i$ (resp. to $\mathcal{M}_i$). In this query, $\mathcal{A}$ specifies $(\mathcal{P}_i, \epsilon)$ where ϵ may be positive or negative. If $\mathcal{A}$ has previously conducted the Establish protocol with this party and the party's output was not $\perp$, then execute the Pay protocol with $\mathcal{A}$ as:

- If $\mathcal{P}_i$ is a customer: $\mathsf{Pay}(\{\mathcal{C}_i(\mathsf{pp}, \epsilon, w_i)\}, \{\mathcal{A}(state)\}) \rightarrow w_i$ (or $\perp$). If the customer's output is not $\perp$, set $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} + \epsilon$.
- If $\mathcal{P}_i$ is a merchant: $\mathsf{Pay}(\{\mathcal{A}(state)\}, \{\mathcal{M}_i(\mathsf{pp}, \epsilon, \mathbf{S}_i)\}) \rightarrow \mathbf{S}_i$ (or $\perp$). If the merchant's output is not $\perp$, $\mathsf{bal}_{\mathcal{A}} \leftarrow \mathsf{bal}_{\mathcal{A}} - \epsilon$.

Where *state* is the adversary's state.

Finalize with $\mathcal{C}_i$ (resp. $\mathcal{M}_i$) When $\mathcal{A}$ makes this query on input $\mathcal{P}_i$ and optional input $\mathsf{rc}_{\mathcal{C}}$, if it has previously established a channel with $\mathcal{P}_i$, it obtains a closure message as:

- If $\mathcal{P}_i$ is a customer: if $\mathcal{A}$ has previously established a channel with $\mathcal{P}_i$ and has not previously Finalized on this party, compute $\mathsf{rc}_{\mathcal{C}} \xleftarrow{R} \mathsf{Refund}(\mathsf{pp}, w_i)$, store $\mathsf{rc}_{\mathcal{C}}$, and return $\mathsf{rc}_{\mathcal{C}}$ to $\mathcal{A}$.
- If $\mathcal{P}_i$ is a merchant: if $\mathcal{A}$ has previously established a channel with $\mathcal{P}_i$ and has not previously Finalized on this party, compute $\mathsf{rc}_{\mathcal{M}} \xleftarrow{R} \mathsf{Refute}(\mathsf{pp}, \mathbf{S}_i, \mathsf{rc}_{\mathcal{C}})$.

If the adversary provided $\mathsf{rc}_{\mathcal{M}}$ and $\mathsf{rc}_{\mathcal{C}}$ is stored, compute $(B^{\mathsf{cust}}_{\mathsf{final}}, B^{\mathsf{merch}}_{\mathsf{final}}) \leftarrow \mathsf{Resolve}(\mathsf{pp}, \mathsf{T}_{\mathcal{C}}, \mathsf{T}_{\mathcal{M}}, \mathsf{rc}_{\mathcal{C}}, \mathsf{rc}_{\mathcal{M}})$ and update $\mathsf{claimed}_{\mathcal{A}} \leftarrow \mathsf{claimed}_{\mathcal{A}} + B^{\mathsf{merch}\text{ (resp. cust)}}_{\mathsf{final}}$.

We say that $\mathcal{A}$ is *legal* if $\mathcal{A}$ never asks to spend from a wallet where $w_i = \perp$ or where w_i is undefined, and where $\mathcal{A}$ never asks $\mathcal{C}_i$ to spend unless the customer has sufficient balance to complete the spend. We say that $\mathcal{A}$ wins the game if at the conclusion of $\mathcal{A}$'s queries, we have $\mathsf{claimed}_{\mathcal{A}} > \mathsf{bal}_{\mathcal{A}}$.

C Proof of Security for Unidirectional Scheme

Proof of Theorem 4.1

Proof. The proof of Theorem 4.1 requires two separate arguments: (1) that the scheme satisfies the *anonymity* property and (2) that the scheme satisfies the *balance* property. We begin by addressing anonymity.

C.1 Anonymity

To prove that the scheme satisfies the anonymity property, we must describe a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, \mathit{auxparams}, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the Real experiment from the Ideal experiment with non-negligible advantage. Recall that in the Ideal experiment (as in the Real experiment), when the adversary $\mathcal{A}$ queries on channel initialization, establishment or closure, the customer answers these queries by honestly running the appropriate algorithms. When the adversary triggers a customer to initiate the Pay protocol, in the Real experiment the adversary runs the protocol honestly. In the Ideal experiment, the customer's side of the protocol is conducted by $\mathcal{S}$.

For all allowed adversaries $\mathcal{A}$, the simulator $\mathcal{S}$ operates as follows. First, if required by the zero-knowledge proof system, we generate a *simulation* CRS for the zero-knowledge proof system, and embed this in pp.[15] When $\mathcal{A}$ calls the simulator on a legal transaction, the simulator emulates the customer's side of the Pay protocol, but with the following changes. First, for $j = 1$ to B, the simulator $\mathcal{S}$ employs the ZK simulation algorithm to simulate each of the zero knowledge proofs π. It generates s_i by sampling a random element in the range of F. Finally, it samples a random key k' for the one-time encryption scheme, samples a random public key pk' by running the KeyGen algorithm, and sets $t := \mathsf{OTEnc}(k', pk')$.

To prove that the Real and Ideal experiments are indistinguishable, we will begin with Real experiment, and modify elements via a series of games until we arrive at the Ideal experiment conducted using our simulator $\mathcal{S}$. Let $\nu_1, \ldots, \nu_3$ be negligible functions. For notational convenience, let $\mathbf{Adv}\,[\,\mathbf{Game\ i}\,]$ be $\mathcal{A}$'s advantage in distinguishing the output of **Game i** from **Game 0**, *i.e.*, the Real distribution.

Game 0. This is the real experiment.

Game 1. In this game, each NIZK π issued during the Pay protocol is simulated. If the proof system is zero-knowledge, then $\mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_1(\lambda)$.

Game 2. In this game, each serial number s presented to $\mathcal{A}$ in the Pay protocol, and each encryption key k used to construct the value t, is replaced with a random value in the range of the pseudorandom function F. In Lemma C.1 we show that if the F is a PRF, the commitment scheme is hiding, and the committing encryption is IND-CPA, then $\mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_2(\lambda)$.

Game 3. In this game, each value t presented to $\mathcal{A}$ in the Pay protocol is constructed by sampling a random $(pk'_c, sk'_c) \leftarrow \mathsf{KeyGen}(1^\lambda)$, then encrypting pk'_c. Under the assumption that OTEnc is IND-CPA for a unique, random key k, then $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] \leq \nu_3(\lambda)$.

By summation over the individual hybrids, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,]$ is negligible in the security parameter. Since the distribution of **Game 3** is identical to the Ideal experiment conducted with our simulator $\mathcal{S}$, this concludes the main proof. We now sketch proofs of the remaining Lemmas.

Lemma C.1 (Replacement of the s, k_t values.)**.** For all p.p.t. distinguishers $\mathcal{A}$ the distribution of **Game 1** (in which each value s, t is generated as in the real protocol) is computationally indistinguishable from the distribution of **Game 2** (in which each s and the key k_t used to encrypt t is a random element) if (1) F is a PRF, (2) the wallet commitment scheme is hiding, and (3) the committing symmetric encryption scheme $(\mathsf{SymEnc}, \mathsf{SymDec})$ is IND-CPA secure.

[15] This is necessary for certain proof systems such as [GS].

Proof sketch. Let $\mathcal{A}$ be an allowed adversary that outputs 1 with non-negligibly different probability when playing **Game 2** and **Game 1**. We use $\mathcal{A}$ to construct three separate distinguishers $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ where at least one of the following is true: (1) $\mathcal{B}_1$ distinguishes the PRF F from a random function with non-negligible advantage, (2) $\mathcal{B}_2$ succeeds against the IND-CPA security of the committing symmetric encryption scheme $(\mathsf{SymEnc}, \mathsf{SymDec})$ with non-negligible advantage, or (3) $\mathcal{B}_3$ succeeds against the hiding property of the commitment scheme with non-negligible advantage.

Let us define a series of intermediate hybrids $H_0 = \textbf{Game 1}, \ldots, H_P = \textbf{Game 2}$, and in each Hybrid $i = 1$ to P, the output of the Pay protocol for a single customer $\mathcal{C}_i$ is modified in the manner of **Game 2**. Given an allowed adversary $\mathcal{A}$ that distinguishes **Game 1** from **Game 2** with non-negligible probability, there must exist an adversary $\mathcal{A}'$ that for some $i \in \{1, \ldots, P\}$, distinguishes one pair of hybrids H_i and H_{i-1} with non-negligible probability. Given such an adversary we now define several more hybrids, and argue that for each of these hybrids the adversary $\mathcal{A}$ must distinguish each from the previous hybrid with at most negligible probabilty.

I.1 Replace the proof π_1 issued by $\mathcal{C}_i$ during the $\mathsf{Establish}$ protocol with a simulated proof. If the proof system is zero knowledge, then $\mathcal{A}$'s advantage in distinguishing this hybrid from the previous hybrid is negligible in λ.

I.2 Replace wCom with a commitment to random values k_1', k_2'. If an adversary distinguishes this hybrid from the previous with non-negligible advantage, then this implies a distinguisher $\mathcal{B}_3$ that succeeds with non-negligible advantage against the hiding property of the commitment scheme. Since we assume the commitment scheme is secure, this bounds the difference between the hybrids to be negligible in λ. (Note that the NIZK proof π_1 is simulated and thus independent of wCom and k_1', k_2'.)

I.3 Replace each s, k_t in the Pay protocol with a value computed using k_1', k_2'. If an adversary distinguishes this hybrid from the previous hybrid with non-negligible advantage, then by Lemma C.2 this implies an attacker against $\mathcal{B}_2$ that wins the $\mathsf{IND-CPA}$ game with non-negligible advantage against $(\mathsf{SymEnc}, \mathsf{SymDec})$. Since we assume the encryption scheme to be $\mathsf{IND-CPA}$ secure, this bound the difference between the hybrids to be negligible in λ. (Recall that the NIZK proof generated in the Pay protocol is simulated.)

I.4 Replace each s, k_t in the Pay protocol with a random element in the range of F. If an adversary distinguishes this hybrid from the previous hybrid with non-negligible advantage, then this implies the existence of $\mathcal{B}_1$ that distinguishes F from a random function, hence under the assumption that F is a PRF, this bounds the difference between the hybrids to be negligible in λ.

I.5 Replace the commitment wCom and proof π_1 with the original distribution from Game 2. Under the assumption that the proof system is zero-knowledge, and the commitment scheme is hiding, the difference between this hybrid and the previous is negligible in λ.

Note that the final hybrid is identical to **Game 2**. Under the assumptions that the proof system is zero knowledge, that F is a PRF, the committing encryption scheme is IND-CPA secure, and the commitment scheme is hiding, the difference between $\mathcal{A}$'s probability of outputting 1 in **Game 2** and **Game 1** is negligible in λ. $\square$

Lemma C.2 (Replacement of the wallet secrets.)**.** For all p.p.t. adversaries $\mathcal{A}$ no adversary can distinguish the intermediate hybrid **I.2** from hybrid **I.3** with non-negligible probability if $(\mathsf{SymEnc}, \mathsf{SymDec})$ is IND-CPA secure.

Proof sketch. Let $\mathcal{A}$ be an allowed adversary that outputs 1 with non-negligibly different probability in hybrid **I.2** from hybrid **I.3**. We show that $\mathcal{A}$ implies an adversary $\mathcal{B}_2$ such that $\mathcal{B}_2$ succeeds in the IND-CPA game against the encryption scheme ($\mathsf{SymEnc}, \mathsf{SymDec}$) with non-negligible advantage. We now describe this adversary.

If $\mathcal{A}'$ distinguishes **I.2** from hybrid **I.3** with non-negligible advantage, we construct $\mathcal{B}_2$ that succeeds with non-negligible advantage against the LOR-CPA security of the symmetric encryption scheme (a definition that is equivalent to IND-CPA security [BDJR97]). $\mathcal{B}_2$ begins with the distribution of **I.2** and first picks a random integer $J \in \{0, \ldots, B\}$ and for $d = 1$ to J: queries the LOR encryption oracle on the pair $(s_d \| u_d \| \pi^r_d, s'_d \| u'_d \| \pi'^r_d)$ where the left input is structured as in **I.2** and the right input is structured as in **I.3**. Given the resulting ciphertexts $C_1, \ldots, C_J$, $\mathcal{A}_1$ now generates the remaining ciphertexts $C'_{J+1}, \ldots, C'_B$ by querying the LOR oracle such that both inputs are constructed as in hybrid H_j. It then constructs the ciphertext vector for customer i as $(C_1, \ldots, C_J, C'_{j+1}, \ldots, C'_B)$ and gives this to $\mathcal{A}'$ as $\mathcal{C}_i$'s output in the $\mathsf{Establish}$ protocol. Note that if the LOR oracle chooses the left input, the distribution of this vector is as in **I.2**, and if it chooses the right input, the distribution is as in **I.3**. When $\mathcal{A}'$ finalizes the channel with $\mathcal{C}_i$, check that the final balance of the customer is $B - J$, and if not, abort. Otherwise, finalize the channel and output $\mathcal{A}'$'s guess as the guess for the LOR-CPA oracle. Note that the abort probability is at most $1/P$, for P polynomial in λ. □

C.2 Balance

We now sketch a proof that the scheme satisfies the Balance definition if the zero-knowledge proof system is simulation extractable, the commitment scheme is binding, and the signature schemes are EU-CMA secure. The primary observation in our proof is that if $\mathcal{A}$, acting as a customer, is able to succeed in obtaining $\mathsf{claimed}_{\mathcal{A}} > \mathsf{bal}_{\mathcal{A}}$, this implies that one of the following conditions is true: (1) $\mathcal{A}$ has successfully paid more than B_0^{cust} coins on a given channel, (2) during the Finalization process, $\mathcal{A}$ has successfully claimed more than the remaining number of coins on a given channel and the honest merchant is not able to produce evidence of fraud. Similarly, a legal adversary $\mathcal{A}$ that wins the game must succeed in either (3) producing evidence (as a merchant) of a doubly-spent coin, even when the customer has behaved honestly, or (4) producing evidence of an invalid ciphertext opening.

Let us first describe a simulated experiment, which is identical to the real protocol interaction but with the following differences. First, if necessary we configure the proof system to allow for the extraction of witnesses, and embed any resulting CRS into pp. Whenever $\mathcal{A}$ initiates the Pay protocol (acting as a customer) to send a successful (accepted) payment, we then extract the witness used to construct the proof π and abort the experiment if the extractor does not produce a valid witness. In addition, we abort if $\mathcal{A}$ is able to submit more than B coins for any given channel (identified by the witness), or if the attacker is able to submit a signature forgery (*i.e.,* submit a signature that was not granted through the $\mathsf{Establish}$ protocol). Finally, if the attacker Finalizes the channel, extracting more than the remaining number of coins available on a given channel, we abort (this implies that $\mathcal{A}$ has produced an additional spend value with respect to the commitment wCom). We simulate all proofs issued during the Pay and $\mathsf{Establish}$ protocols. Whenever the adversary, acting as a merchant, posts a channel closure message $\mathsf{rc}_{\mathcal{M}}$ such that $\mathsf{Resolve}$ executed on the customer and merchant inputs outputs $B_{\mathsf{final}}^{\mathsf{merch}} > 0$, where the final balance is inconsistent with the actual remaining balance, we abort the protocol. We note that if there exists an adversary $\mathcal{A}$ who succeeds in winning the Balance game with non-negligible advantage, then this implies an attacker with the ability to distinguish the real experiment from the simulated experiment with non-negligible advantage. We show that such an attacker represents a contradiction, assuming that the proof system is sound and the signature scheme is EU-CMA. Consider the following hybrids:

Game 0. This is the real experiment.

Game 1. This game is identical to **Game 0** except that we extract on every valid proof π_1 in the $\mathsf{Establish}$ protocol, every proof π in the Pay protocol, and every proof π_r^j that the customer reveals as a result of a channle Finalization. We abort if the extractor ever fails to produce a valid witness. Under the assumption that the proof system is sound, the abort probability is negligible. Thus $\mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that we abort if the customer ever presents a collision in wCom (*e.g.,* in the witness to any proof of knowledge). Assuming that the commitment scheme is binding, $\mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_2(\lambda)$.

Game 3. This game is identical to **Game 2** except that we abort if $\mathcal{A}$ is able to successfully submit $B' > B$ coins on a given channel. Note that the serial number s is computed as a function of the secret key k_1 and the coin index $0 \leq i < B$. Thus, there are at most B distinct values of s for any given (signed) PRF seed k_1. Thus, for this abort to occur, it must be the case that $\mathcal{A}$ has forged a signature σ_w that was not issued during the $\mathsf{Establish}$ protocol. If this occurs, we obtain an adversary $\mathcal{B}$ that succeeds against the EU-CMA security of the signature. Since we assume that the signature scheme is EU-CMA secure, then we obtain $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] \leq \nu_3(\lambda)$.

Game 4. This game is identical to **Game 3**, except that we abort if $\mathcal{A}$ ever produces a ciphertext C_j that contains a witness to the proof statement with an opening of the commitment wCom that does not match with the corresponding values used in the Pay protocol. If this occurs, this implies an attacker that violates the binding property of the commitment scheme. Since we assume that the commitment scheme is binding, then we obtain $\mathbf{Adv}\,[\,\mathbf{Game\ 4}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] \leq \nu_4(\lambda)$.

Game 5. This game is identical to **Game 4** except that whenever $\mathcal{A}$ presents signed evidence that the customer has supplied an invalid ciphertext (that does not decrypt with key ck_j), we abort. Since no customer ever outputs invalid ciphertexts or keys, this implies that the adversary has constructed a forged signature using the signature scheme. This implies that we can use $\mathcal{A}$ to win the EU-CMA game against the signature scheme. Thus, under the assumption that the signature scheme is EU-CMA secure, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 5}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 4}\,] \leq \nu_5(\lambda)$.

Game 6. This game is identical to **Game 5** except that we simulate each zero knowledge proof issued in the Pay and $\mathsf{Establish}$ protocols. Since the proof system is zero knowledge, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 6}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 5}\,] \leq \nu_6(\lambda)$.

Game 7. This game is identical to **Game 6** except that whenever $\mathcal{A}$, acting as a merchant, presents signed evidence of a doubly-spent coin that is accepted by the $\mathsf{Resolve}$ algorithm, we abort. We argue that intuitively, such an adversary $\mathcal{A}$ can be used to break the EU-CMA property of the signature scheme or the IND-CPA property of the symmetric encryption scheme as follows. On input a public key in the EU-CMA game, embed this key as pk_c. Now guess an index J at which the payment channel will be closed. We further replace each of the first $J-1$ ciphertexts created during the $\mathsf{Establish}$ protocol with the encryption of a random element. Now, if the adversary outputs a new proof, note that we can extract a witness to the (new) proof, which is distinct from any of the previous proofs and therefore embeds a valid secret key sk_c for the customer. This provides us with the signing key for the signature scheme and allows us to forge a signature on any message. This proof requires that $\mathcal{A}$ cannot distinguish the encryption of random messages from the encryption of valid proofs; this can be shown using the IND-CPA property of the signature scheme. Completing this proof requires a hybrid argument in which the above process is repeated for each customer. Thus, under the assumption that the scheme is IND-CPA secure and the signature scheme is EU-CMA secure, we have $\mathbf{Adv}\,[\,\mathbf{Game\ 7}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 6}\,] \leq \nu_7(\lambda)$.

By summation over the individual hybrids, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 7}\,]$ is negligible in the security parameter. We note that the distribution of **Game 7** is computationally indistinguishable from the real experiment. Thus the simulation satisfies the property of Balance. □

D Proof of Security for Bidirectional Scheme

Proof sketch. As in the previous proofs, the proof of Theorem 4.2 requires two separate arguments: (1) that the scheme satisfies the *anonymity* property and (2) that the scheme satisfies the *balance* property. We begin by addressing anonymity. Note that for this scheme we make the simplifying assumption that the legal adversary does not abort the Pay protocol.

D.1 Anonymity

To prove that the scheme satisfies the anonymity property, we must describe a simulator $\mathcal{S}^{X-Y(\cdot)}(\mathsf{pp}, \mathit{auxparams}, \cdot)$ such that for all $\mathsf{T}_{\mathcal{M}}$, no allowed adversary $\mathcal{A}$ can distinguish the Real experiment from the Ideal experiment with non-negligible advantage. Recall that in the Ideal experiment (as in the Real experiment), when the adversary $\mathcal{A}$ queries on channel initialization, establishment or closure, the customer answers these queries by honestly running the appropriate algorithms. When the adversary triggers a customer to initiate the Pay protocol, in the Real experiment the adversary runs the protocol honestly. In the Ideal experiment, the customer's side of the protocol is conducted by $\mathcal{S}$.

For all allowed adversaries $\mathcal{A}$, the simulator $\mathcal{S}$ operates as follows. First, if required by the zero-knowledge proof system, we generate a *simulation* CRS for the zero-knowledge proof system, and embed this in pp.[16] When $\mathcal{A}$ calls the simulator on a legal transaction, the simulator $\mathcal{S}$ emulates the customer's side of the Pay protocol, but with three differences: (1) the commitment wCom' is replaced with a commitment to a random message, (2) the simulator $\mathcal{S}$ generates a random public key wpk when it runs the protocol, and (3), the simulator employs the ZK simulation algorithm to simulate each of the zero-knowledge proofs. In all other ways it behaves as in the normal protocol, generating wpk and σ_{rev} as usual.

To prove that the Real and Ideal experiments are indistinguishable, we will begin with Real experiment, and modify elements via a series of games until we arrive at the Ideal experiment conducted using our simulator $\mathcal{S}$. Let ν_1, ν_2 be negligible functions. For notational convenience, let $\mathbf{Adv}\,[\,\mathbf{Game\ i}\,]$ be $\mathcal{A}$'s advantage in distinguishing the output of **Game i** from the Real distribution.

Game 0. This is the real experiment.

[16] This is necessary for certain proof systems such as [GS].

Game 1. This game is identical to **Game 0** except that each NIZK generated by a customer at any stage of the Pay protocol interaction is replaced with a simulated proof. Note that we require all legal adversaries to refuse to proceed subsequent to the failure of any Pay protocol interaction, and we provide this information to $\mathcal{S}$. Thus, If the proof system is zero-knowledge, then $\mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that the commitment wCom' is replaced with a commitment to a random message. If the commitment scheme is (computationally) hiding, then $\mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_1(\lambda)$.

Game 3. This game is identical to **Game 2** except that the value wpk is replaced with a random key generated using the KeyGen algorithm. Note that the distribution of the replacement wpk value is identical to the distribution of the original value, hence $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] = 0$.

By summation over the hybrids, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,]$ is negligible in the security parameter. Since **Game 3** is identical to the Ideal experiment, the bidirectional scheme is anonymous.

D.2 Balance

To win the Balance game, a malicious adversary $\mathcal{A}$ must claim more money than actually available, as measured by her expenditures and channel openings. We proceed by describing a simulated experiment in which $\mathcal{A}$ wins the Balance game with probability 0, and proceed to show that the real protocol interaction is computationally indistinguishable from this simulation, under the assumptions that (1) the ZK proof system is simulation-extractable, (2) the signature scheme is EU-CMA secure, (3) the commitment scheme is secure. To complete this argument, let us first define the following hybrids.

Game 0. This is the real experiment.

Game 1. This game is identical to **Game 0** except that we extract on every proof π_1, π_2 in the $\mathsf{Establish}$ and Pay protocols and abort if the extractor fails. By the soundness of the proof system, $\mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_1(\lambda)$.

Game 2. This game is identical to **Game 1** except that we abort if $\mathcal{A}$ ever presents a collision in wCom (*e.g.,* in the witness to any proof of knowledge). Assuming that the commitment scheme is binding, $\mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 1}\,] \leq \nu_2(\lambda)$.

Game 3. This game is identical to **Game 1** except that we abort if the extracted signature on wCom is not on a message signed by the merchant (as indicated by the witnesses extracted in the first game). Under the assumption that the signature scheme is EU-CMA, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 2}\,] \leq \nu_3(\lambda)$.

Game 4. This game is identical to **Game 2**, except we abort if σ_w in the refund transaction was not one produced by the merchant. Under the assumption that the signature scheme is EU-CMA, we have that $\mathbf{Adv}\,[\,\mathbf{Game\ 4}\,] - \mathbf{Adv}\,[\,\mathbf{Game\ 3}\,] \leq \nu_4(\lambda)$.

In the following we will argue that no alllowed adversary can succeed in the Balance game against **Game 4**. By summation over the hybrids we have that **Game 4** s indistinguishable from **Game 0**, and this implies that all allowed adversaries will succeed with at most negligible advantage against the real protocol.

Let $\mathcal{A}$ be a p.p.t. adversary that succeeds with non-negligible advantage in the Balance game. We argue that this implies one of the following events has occurred:

1. The adversarial customer has presented a signature σ_w (as a witness) that was not issued by the merchant. This cannot occur in **Game 4** as it would imply an abort due to a signature forgery.
2. The adversarial customer has forged a zero-knowledge proof. This cannot occur in **Game 4** as all proofs produce valid witnesses.
3. The adversarial customer has identified a collision in the commitment scheme. This cannot occur in **Game 4** as it would cause an abort.
4. The adversarial merchant has produced a refund token σ_{rev} that the honest customer did not produce. This cannot occur in **Game 4** as it would imply an abort due to a signature forgery.

Since these events do not occur in **Game 4**, the advantage of an adversary succeeding in this game is 0. This concludes the sketch $\square$

E Additional assumptions for the PRF

In this section we briefly sketch a proof that the Dodis-Yampolskiy pseudorandom function [DY05] provides strong pre-image resistance if the q-DBDHI assumption holds in $\mathbb{G}$.

The Dodis-Yampolskiy PRF. Let p be a prime and let $\mathcal{I} \subset \mathbb{Z}_p \setminus \{0\}$ be a polynomially-sized input space. The public parameters for the Dodis-Yampolskiy PRF are a group $\mathbb{G}$ of prime order p with generator g. The seed is a random element $s \in \mathbb{Z}_p$ and the pseudorandom function is computed as $f_s(x) = g^{1/(s+x)}$. Security for the PRF over input space $\mathcal{I}$ with $|\mathcal{I}| = q$ is shown to hold under the q-DBDHI assumption in [DY05].

The Dodis-Yampolskiy PRF provides strong pre-image resistance. We now sketch a proof that the Dodis-Yampolskiy PRF provides strong pre-image resistance for a polynomially-sized domain under the q-DBDHI assumption.

Our proof proceeds as follows. Let $\mathcal{A}$ be a p.p.t. adversary that, given access to an oracle F_s^{DY} implementing the Dodis-Yampolskiy PRF with an honestly-generated seed s (with the restriction that $\mathcal{A}$ can query only on elements in $\mathcal{I}$) such that with non-negligible probability $\mathcal{A}$ outputs (x, s', x') with $x, x' \in \mathcal{I}$ and $F_s^{DY}(x) = F_{s'}^{DY}(x')$. We show that $\mathcal{A}$'s output can be used to recover the seed for any PRF instance, thus violating the pseudorandomness property of the PRF.

To show this last step, we construct a distinguisher $\mathcal{B}$ against the pseudorandomness of the Dodis-Yampolskiy scheme. $\mathcal{B}$ runs $\mathcal{A}$ internally and interacts with an oracle that implements either the PRF or a random function. Each time $\mathcal{A}$ queries on some value x_i, $\mathcal{B}$ queries its oracle on the same value and returns the response to $\mathcal{A}$. When $\mathcal{A}$ outputs (x, s', x') such that $F_s^{DY}(x) = F_{s'}^{DY}(x')$, $\mathcal{B}$ computes a candidate guess for the PRF seed as $\bar{s} = s' + x' - x$, and tests to see whether two or more distinct outputs it receives from its oracle are consistent with $\bar{s}$. If so, $\mathcal{B}$ outputs 1.

If $\mathcal{B}$ is interacting with an instance of the PRF, then $\mathcal{A}$ will succeed with non-negligible probability. In this instance, the value $\bar{s}$ will be equal to the PRF seed, because if $F_s^{DY}(x) = F_{s'}^{DY}(x')$ then this implies the relation $g^{1/s+x} = g^{1/s'+x'}$ and thus $s + x = s' + x'$, yielding $s = s' + x' - x$. If $\mathcal{B}$ is interacting with a random function, then there is no seed to recover, and the probability that multiple oracle outputs are consistent with a recovered candidate seed is negligible. Thus $\mathcal{B}$ succeeds with non-negligible probability. Since the pseudorandomness of the Dodis-Yampolskiy PRF is shown to hold under the q-DBDHI assumption, this implies that the strong pre-image resistance must also hold if q-DBDHI holds in $\mathbb{G}$.

Other PRFs. While we recommend using the Dodis-Yampolskiy PRF for our constructions, the strong pre-image resistance property holds for other PRFs. For example, hash-based PRFs such as HMAC provide this property under the assumption that the underlying hash function is collision-resistant, since the equality of two distinct outputs implies a collision in the hash function.

F Proof sketch for indirect payment protocol

We now argue security of the indirect payment protocol against an extension of our current definition where the attacker can also initiate third party payments to any party including himself, via any third party including himself.

Recall that the indirect payment protocol consists of six messages. Messages 1 is identical to message 1 in the bidirectional protocol, just with a different recipient. As such, any attack here directly reduces to an attack on the bidirectional protocol. The same holds for message 2 since it is the same, up to the omission of the payment value, as the corresponding message in the bidirectional protocol plus the contents of message 1. Extracting the payment value from the corresponding proofs closes this gap.

Lastly, we note that message 6 reduces to the corresponding last message in bidirectional protocol for $\mathbf{A} \rightarrow \mathbf{I}$ or $\mathbf{I} \rightarrow \mathbf{B}$. An attack here again reduces to an attack on the same message in the underlying bidirectional channel. Note that it is possible that a dishonest adversary sends only half of message 6, leaving the other party without it. However, aborts at this phase are allowed in the bidirectional protocol as well.

We must now consider the remaining messages and what happens when various parties are dishonest.

Consider the case where A is honest but I and B are not. Messages 1 and 6 are covered by the analysis above. This leaves the question of message 3. If message 3 is omitted or invalid, then the attack is either in the portion intended for B (in which case it does not effect A) or identical to an attack on the bare $\mathbf{A} \rightarrow \mathbf{I}$ channel. Thus the security in this case reduces to the underlying channel security.

We now consider the case where **I** is honest, but **A** and **B** are dishonest. Then message 5 may be omitted or invalid. In which case either:

1. **A** closes on their new channel balance.

2. **A** closes on their old channel balance.

Case 1 is an allowed outcome since **I** is paid. In case 2, by the security of the $\mathbf{A} \rightarrow \mathbf{I}$ channel, **A** legitimately closed on their new channel state. The conditional refund token for **B** prevents **B** from closing on their new channel state. As such, any attack where I is forced to pay **B** results form an attack on the underlying $\mathbf{I} \rightarrow \mathbf{B}$ channel.

We now consider the case where **B** is honest but **I** and **A** are not. Messages 1 and 6 are handled by the initial analysis. The concerning question is message 4 which consists of a conditional refund token and a revocation token. There are several possibilities:

1. the refund token is omitted

2. the refund token is invalid

3. the revocation token is omitted.

4. A has already closed their channel.

Cases 1 and 2 reduce to the security of the underlying $\mathbf{I} \rightarrow \mathbf{B}$channel. Cases 3 and 4 are distinct because they represent the two states where **B** cannot meet the conditions on the refund token. In case 3, **B** simply aborts the protocol and closes the channel under the old state. We note that there is no race condition here as there must be some timeout between when **A** posts a closure message to and when the channel is closed to allow for disputes. In case 4, **B** can also just abort and close their channel. Thus in these two cases, security reduces to the security of the underlying $\mathbf{I} \rightarrow \mathbf{B}$ for aborts.

Finally, we must consider the case where **A** and **B** are honest, but **I** is not. For A, this security reduces to the malicious $\mathbf{I}, \mathbf{B}$ case and vice versa for **B**. There is no guarantee that in the face of a malicious **I**, the parties can complete a payment, only that they don't loose money or anonymity.