TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub

Ethan Heilman*, Leen AlShenibr*, Foteini Baldimtsi†, Alessandra Scafuro‡ and Sharon Goldberg*
*Boston University {heilman, leenshe}@bu.edu, goldbe@cs.bu.edu
†George Mason University foteini@gmu.edu
‡North Carolina State University ascafur@ncsu.edu

***Abstract*—This paper presents *TumbleBit*, a new unidirectional unlinkable payment hub that is fully compatible with today's Bitcoin protocol. TumbleBit allows parties to make fast, anonymous, off-blockchain payments through an untrusted intermediary called the Tumbler. TumbleBit's anonymity properties are similar to classic Chaumian eCash: no one, not even the Tumbler, can link a payment from its payer to its payee. Every payment made via TumbleBit is backed by bitcoins, and comes with a guarantee that Tumbler can neither violate anonymity, nor steal bitcoins, nor "print money" by issuing payments to itself. We prove the security of TumbleBit using the real/ideal world paradigm and the random oracle model. Security follows from the standard RSA assumption and ECDSA unforgeability. We implement TumbleBit, mix payments from 800 users and show that TumbleBit's off-blockchain payments can complete in seconds.**

I. Introduction

One reason for Bitcoin's initial popularity was the perception of anonymity. Today, however, the sheen of anonymity has all but worn off, dulled by a stream of academic papers [31], [42], and a blockchain surveillance industry [26], [22], that have demonstrated weaknesses in Bitcoin's anonymity properties. As a result, a new market of anonymity-enhancing services has emerged [35], [17], [1]; for instance, 1 million USD in bitcoins are funneled through JoinMarket each month [35]. These services promise to mix bitcoins from a set of *payers* (*aka,* input Bitcoin addresses $\mathcal{A}$) to a set of *payees* (*aka,* output bitcoin addresses $\mathcal{B}$) in a manner that makes it difficult to determine which payer transferred bitcoins to which payee.

To deliver on this promise, anonymity must also be provided in the face of the anonymity-enhancing service itself—if the service knows exactly which payer is paying which payee, then a compromise of the service leads to a total loss of anonymity. Compromise of anonymity-enhancing technologies is not unknown. In 2016, for example, researchers found more than 100 Tor nodes snooping on their users [37]. Moreover, users of mix services must also contend with the potential risk of "exit scams", where an established business takes in new payments but stops providing services. Exit scams have been known to occur in the Bitcoin world. In 2015, a Darknet Marketplace stole 11.7M dollars worth of escrowed customer bitcoins [44], while btcmixers.com mentions eight different scam mix services. Thus, it is crucial that anonymity-enhancing services be designed in a manner that prevents bitcoin theft.

TumbleBit: An unlinkable payment hub. We present TumbleBit, a *unidirectional unlinkable payment hub* that uses an *untrusted* intermediary, the *Tumbler* $\mathcal{T}$, to enhance anonymity. Every payment made via TumbleBit is backed by bitcoins. We use cryptographic techniques to guarantee Tumbler $\mathcal{T}$ can neither violate anonymity, nor steal bitcoins, nor "print money" by issuing payments to itself. TumbleBit allows a payer Alice $\mathcal{A}$ to send fast off-blockchain payments (of denomination one bitcoin) to a set of payees $(\mathcal{B}_1, ..., \mathcal{B}_Q)$ of her choice. Because payments are performed off the blockchain, TumbleBit also serves to scale the volume and velocity of bitcoin-backed payments. Today, on-blockchain bitcoin transactions suffer a latency of $\approx$ 10 minutes. Meanwhile, TumbleBit payments are sent off-blockchain, via the Tumbler $\mathcal{T}$, and complete in seconds. (Our implementation[1] completed a payment in 1.2 seconds, on average, when $\mathcal{T}$ was in New York and $\mathcal{A}$ and $\mathcal{B}$ were in Boston.)

TumbleBit Overview. TumbleBit replaces on-blockchain payments with off-blockchain puzzle solving, where Alice $\mathcal{A}$ pays Bob $\mathcal{B}$ by providing $\mathcal{B}$ with the solution to a puzzle. The puzzle z is generated through interaction between $\mathcal{B}$ and $\mathcal{T}$, and solved through an interaction between $\mathcal{A}$ and $\mathcal{T}$. Each time a puzzle is solved, 1 bitcoin is transferred from Alice $\mathcal{A}$ to the Tumbler $\mathcal{T}$ and finally on to Bob $\mathcal{B}$.

The protocol proceeds in three phases; see Figure 1. In the on-blockchain *Escrow Phase*, each payer Alice

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
Copyright 2017 Internet Society, ISBN 1-891562-46-0
http://dx.doi.org/10.14722/ndss.2017.23086

[1] https://github.com/BUSEC/TumbleBit/

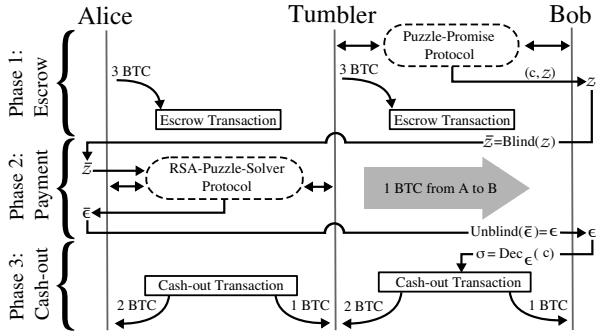

Fig. 1. Overview of the TumbleBit protocol.

$\mathcal{A}$ opens a payment channel with the Tumbler $\mathcal{T}$ by escrowing Q bitcoins on the blockchain. Each payee Bob $\mathcal{B}$ also opens a channel with $\mathcal{T}$. This involves (1) $\mathcal{T}$ escrowing Q bitcoins on the blockchain, and (2) $\mathcal{B}$ and $\mathcal{T}$ engaging in a *puzzle-promise* protocol that generates up to Q puzzles for $\mathcal{B}$. During the off-blockchain *Payment Phase*, each payer $\mathcal{A}$ makes up to Q off-blockchain payments to any set of payees. To make a payment, $\mathcal{A}$ interacts with $\mathcal{T}$ to learn the solution to a puzzle $\mathcal{B}$ provided. Finally, the *Cash-Out Phase* closes all payment channels. Each payee $\mathcal{B}$ uses his Q' solved puzzles (*aka,* TumbleBit payments) to create an on-blockchain transaction that claims Q' bitcoins from $\mathcal{T}$'s escrow. Each payer $\mathcal{A}$ also closes her escrow with $\mathcal{T}$, recovering bitcoins not used in a payment.

Anonymity properties. TumbleBit provides *unlinkability*: Given the set of escrow transactions and the set of cash-out transactions, we define a valid configuration as a set of payments that explains the transfer of funds from Escrow to Cash-Out. Unlinkability ensures that if the Tumbler $\mathcal{T}$ does not collude with other TumbleBit users, then $\mathcal{T}$ cannot distinguish the true configuration (*i.e.,* the set of payments actually sent during the Payment Phase) from any other valid configuration.

TumbleBit is therefore similar to classic Chaumian eCash [12]. With Chaumian eCash, a payee $\mathcal{A}$ first withdraws an eCash coin in exchange for money (*e.g.,* USD) at an intermediary Bank, then uses the coin to pay a payee $\mathcal{B}$. Finally $\mathcal{B}$ redeems the eCash coin to the Bank in exchange for money. Unlinkability ensures that the Bank cannot link the withdrawal of an eCash coin to the redemption of it. TumbleBit provides unlinkability, with Tumbler $\mathcal{T}$ playing the role of the Chaumian Bank. However, while Tumbler $\mathcal{T}$ need not be trusted, the Chaumian Bank is trusted to not (1) "print money" (*i.e.,* issue eCash coins to itself) or (2) steal money (*i.e.,* refuse to exchange coins for money).

TumbleBit: As a classic tumbler. TumbleBit can also be used as a classic Bitcoin tumbler, mixing together the transfer of one bitcoin from $\aleph$ distinct *payers* (Alice $\mathcal{A}$) to $\aleph$ distinct payees (Bob $\mathcal{B}$). In this mode, TumbleBit is run as in Figure 1 with the payment phase shrunk to 30 seconds, so the protocol runs in *epochs* that require two blocks added to the blockchain. As a classic tumbler, TumbleBit provides k-*anonymity within an epoch*—no one, not even the Tumbler $\mathcal{T}$, can link one of the k transfers that were successfully completed during the epoch to a specific pair of payer and payee $(\mathcal{A}, \mathcal{B})$.

RSA-puzzle solving. At the core of TumbleBit is our new "RSA puzzle solver" protocol that may be of independent interest. This protocol allows Alice $\mathcal{A}$ to pay one bitcoin to $\mathcal{T}$ in *fair exchange*[2] for an RSA exponentiation of a "puzzle" value z under $\mathcal{T}$'s secret key. Fair exchange prevents a cheating $\mathcal{T}$ from claiming $\mathcal{A}$'s bitcoin without solving the puzzle. Our protocol is interesting because it is fast—solving 2048-bit RSA puzzles faster than [30]'s fair-exchange protocol for solving 16x16 Sudoku puzzles (Section VIII))—and because it supports RSA. The use of RSA means that blinding can be used to break the link between the user providing the puzzle (*i.e.,* Bob $\mathcal{B}$) and the user requesting its solution (*e.g.,* payer Alice $\mathcal{A}$).

Cryptographic protocols. TumbleBit is realized by interleaving the RSA-puzzle-solver protocol with another fair-exchange *puzzle-promise* protocol. We formally prove that each protocol is a fair exchange. Our proofs use the real/ideal paradigm in the random oracle model (ROM) and security relies on the standard RSA assumption and the unforgeability of ECDSA signatures. Our proofs are in the full version [20].

A. TumbleBit Features

Bitcoin compatibility. TumbleBit is fully compatible with today's Bitcoin protocol. We developed (off-blockchain) cryptographic protocols that work with the very limited set of (on-blockchain) instructions provided by today's Bitcoin scripts. Bitcoin scripts can only be used to perform two cryptographic operations: (1) validate the preimage of a hash, or (2) validate an ECDSA signature on a Bitcoin transaction. The limited functionality of Bitcoin scripts is likely here to stay; indeed, the recent "DAO" theft [39] has highlighted the security risks of complex scripting functionalities.

No coordination. In contrast to earlier work [28], [43], if Alice $\mathcal{A}$ wants to pay Bob $\mathcal{B}$, she need not interact with any other TumbleBit users. Instead, $\mathcal{A}$ and $\mathcal{B}$ need only interact with the Tumbler and each other. This lack of coordination between TumbleBit users makes it possible to scale our system.

Performance. We have implemented our TumbleBit system in C++ and python, using LibreSSL as our cryptographic library. We have tumbled payments from 800 payers to 800 payees; the relevant transactions are visible on the blockchain. Our protocol requires 327 KB

[2]True fair exchange is impossible in the standard model [38] and thus alternatives have been proposed, such as gradual release mechanisms, optimistic models, or use of a trusted third party. We follow prior works that use Bitcoin for fair exchange [4], [24], [25] and treat the blockchain as a trusted public ledger. Other works use the term Contingent Payment or Atomic Swaps [27], [5].

Scheme	Prevents Theft	Anonymous	Resists DoS	Resists Sybils	Minimum Mixing Time	Bitcoin Compatible	No Coordination?
Coinjoin [28]	✓	small set	×	×	1 block	✓	×
Coinshuffle [43], [34]	✓	small set	×	×	1 block	✓	× (p2p network)
Coinparty [49]	$2/3$ users honest	✓	some[1]	✓ (fees)	2 blocks	✓	×
XIM [9]	✓	✓	✓	✓ (fees)	hours	✓	× (uses blockchain)
Mixcoin [11]	TTP accountable	× (TTP)	✓	✓ (fees)	2 blocks	✓	✓
Blindcoin [48]	TTP accountable	✓	✓	✓ (fees)	2 blocks	✓	✓
CoinSwap [29]	✓	× (TTP)[2]	✓	✓ (fees)	2 blocks	✓	✓
BSC [21]	✓	✓	✓	✓ (fees)	3 blocks	×	✓
TumbleBit	✓	✓	✓	✓ (fees)	2 blocks	✓	✓

TABLE I. A COMPARISON OF BITCOIN TUMBLER SERVICES. TTP STANDS FOR TRUSTED THIRD PARTY. WE COUNT MINIMUM MIXING TIME BY THE MINIMUM NUMBER OF BITCOIN BLOCKS. ANY MIXING SERVICE INHERENTLY REQUIRES AT LEAST ONE BLOCK.
[1] COINPARTY COULD ACHIEVE SOME DOS RESISTANCE BY FORCING PARTIES TO SOLVE PUZZLES BEFORE PARTICIPATING.

of data on the wire, and 0.6 seconds of computation on a single CPU. Thus, performance in classic tumbler mode is limited only by the time it takes for two blocks to be confirmed on the blockchain and the time it takes for transactions to be confirmed; currently, this takes ≈ 20 minutes. Meanwhile, off-blockchain payments can complete in seconds (Section VIII).

B. Related Work

TumbleBit is related to work proposing new anonymous cryptocurrencies (*e.g.,* Zerocash [33], [7], Monero [2] or Mimblewimble [23]). While these are very promising, they have yet to be as widely adopted as Bitcoin. On the other hand, TumbleBit is an anonymity service for Bitcoin's *existing* user base.

Off-blockchain payments. When used as an unlinkable payment hub, TumbleBit is related to micropayment channel networks, notably Duplex Micropayment Channels [13] and the Lightning Network [40]. These systems also allow for Bitcoin-backed fast off-blockchain payments. Payments are sent via paths of intermediaries with pre-established on-blockchain pairwise escrow transactions. TumbleBit (conceptually) does the same. However, while the intermediaries in micropayment channel network can link payments from $\mathcal{A}$ to $\mathcal{B}$, TumbleBit's intermediary $\mathcal{T}$ cannot. Our earlier workshop paper [21] proposed a protocol that adds anonymity to micropayment channel networks. TumbleBit is also related to concurrent work proposing Bolt [18], an off-blockchain unlinkable payment channel. However, while TumbleBit is both implemented and Bitcoin comptabile, Bolt [18] and [21] are not. Our full version [20] has more discussion on [18], [21].

Bitcoin Tumblers. Prior work on classic Bitcoin Tumblers is summarized in Table I-A. CoinShuffle(++) [43], [34] both perform a mix in a single transaction. Bitcoin's maximum transaction size (100KB) limits CoinShuffle(++) to 538 users per mix. These systems are also particularly vulnerable to DoS attacks, where a user joins the mix and then aborts, disrupting the protocol for all other users. Decentralization also requires mix users to interact via a peer-to-peer network in order to identify each other and mix payments. This coordination between users causes communication to grow quadratically [9], [10], limiting scalability; neither [43] nor [34] performs a mix with more than 50 users. Decentralization also makes it easy for an attacker to create many Sybils and trick Alice $\mathcal{A}$ into mixing with them in order to deanonymize her payments [10], [47]. TumbleBit sidesteps these scalability limitations by not requiring coordination between mix users. The full version [20] discusses the other tumblers in Table I-A.

After this paper was first posted, Dorier and Ficsor began an independent TumbleBit implementation.[3]

II. BITCOIN SCRIPTS AND SMART CONTRACTS

In designing TumbleBit, our key challenge was ensuring compatibility with today's Bitcoin protocol. We therefore start by reviewing Bitcoin transactions and Bitcoin's non-Turing-complete language *Script*.

Transactions. A Bitcoin user Alice $\mathcal{A}$ is identified by her bitcoin address (which is a public ECDSA key), and her bitcoins are "stored" in *transactions*. A single transaction can have multiple *outputs* and multiple *inputs*. Bitcoins are transferred by sending the bitcoins held in the output of one transaction to the input of a different transaction. The blockchain exists to provide a public record of all valid transfers. The bitcoins held in a transaction output can only be transferred to a single transaction input. A transaction input T_3 *double-spends* a transaction input T_2 when both T_2 and T_3 *point to* (*i.e.,* attempt to transfer bitcoins from) the same transaction output T_1. The security of the Bitcoin protocol implies that double-spending transactions will not be confirmed on the blockchain. Transactions also include a *transaction fee* that is paid to the Bitcoin miner that confirms the transaction on the blockchain. Higher fees are paid for larger transactions. Indeed, fees for confirming transactions on the blockchain are typically expressed as "Satoshi-per-byte" of the transaction.

Scripts. Each transaction uses Script to determine the conditions under which the bitcoins held in that transaction can be moved to another transaction. We build "smart contracts" from the following transactions:
- T_{offer}: One party $\mathcal{A}$ offers to pay bitcoins to any party that can sign a transaction that meets some condition $\mathcal{C}$. The T_{offer} transaction is signed by $\mathcal{A}$.
- T_{fulfill}: This transaction points to T_{offer}, meets the condition $\mathcal{C}$ stipulated in T_{offer}, and contains the public key of the party $\mathcal{B}$ receiving the bitcoins.

[3] https://github.com/NTumbleBit/NTumbleBit

T_{offer} is posted to the blockchain first. When T_{fulfill} is confirmed by the blockchain, the bitcoins in T_{fulfill} flow from the party signing transaction T_{offer} to the party signing T_{fulfill}. Bitcoin scripts support two types of conditions that involve cryptographic operations:

Hashing condition: The condition $\mathcal{C}$ stipulated in T_{offer} is: "T_{fulfill} must contain the preimage of value y computed under the hash function H." Then, T_{fulfill} collects the offered bitcoin by including a value x such that $H(x) = y$. (We use the `OP_RIPEMD160` opcode so that H is the RIPEMD-160 hash function.)

Signing condition: The condition $\mathcal{C}$ stipulated in T_{offer} is: "T_{fulfill} must be digitally signed by a signature that verifies under public key PK." Then, T_{fulfill} fulfills this condition if it is validly signed under PK. The signing condition is highly restrictive: (1) today's Bitcoin protocol requires the signature to be ECDSA over the Secp256k1 elliptic curve [41]—no other elliptic curves or types of signatures are supported, and (2) the condition specifically requires T_{fulfill} *itself* to be signed. Thus, one could not use the signing condition to build a contract whose condition requires an *arbitrary message* m to be signed by PK.[4] (TumbleBit uses the `OP_CHECKSIG` opcode, which requires verification of a single signature, and the "2-of-2 multisignature" template '`OP_2 key1 key2 OP_2 OP_CHECKMULTISIG`' which requires verification of a signature under `key1` AND a signature under `key2`.)[5]

Script supports composing conditions under "IF" and "ELSE". Script also supports *timelocking* (`OP_CHECKLOCKTIMEVERIFY` opcode [46]), where T_{offer} also stipulates that T_{fulfill} is timelocked to time window tw. (Note that tw is an absolute block height.) This allows the party that posted T_{fulfill} to reclaim their bitcoin if T_{fulfill} is unspent and the block height is higher than tw. Section VIII-A details the scripts used in our implementation.

2-of-2 escrow. TumbleBit relies heavily on the commonly-used *2-of-2 escrow* smart contract. Suppose that Alice $\mathcal{A}$ wants to put Q bitcoin in escrow to be redeemed under the condition $\mathcal{C}_{2of2}$: "the fulfilling transaction includes two signatures: one under public key PK_1 AND one under PK_2."

To do so, $\mathcal{A}$ first creates a multisig address $PK_{(1,2)}$ for the keys PK_1 and PK_2 using the Bitcoin `createmultisig` command. Then, $\mathcal{A}$ posts an *escrow transaction* T_{escr} on the blockchain that sends Q bitcoin to this new multisig address $PK_{(1,2)}$. The T_{escr} transaction is essentially a T_{offer} transaction that requires the fulfilling transaction to meet condition $\mathcal{C}_{2of2}$. We call the fulfilling transaction T_{cash} the *cash-out transaction*. Given that $\mathcal{A}$ doesn't control both PK_1 and PK_2 (*i.e.,* doesn't know the corresponding secret keys), we also timelock the T_{escr} transaction for a time window tw. Thus, if a valid T_{cash} is not confirmed by the blockchain within time window tw, the escrowed bitcoins can be reclaimed by $\mathcal{A}$. Therefore, $\mathcal{A}$'s bitcoins are escrowed until either (1) the time window expires and $\mathcal{A}$ reclaims her bitcoins or (2) a valid T_{cash} is confirmed. TumbleBit uses 2-of-2 escrow to establish pairwise payment channels, per Figure 1.

[4]This is why [21] is not Bitcoin-compatible. [21] requires a blind signature to be computed over an arbitrary message. Also, ECDSA-Secp256k1 does not support blind signatures.

[5]Unlike cryptographic multisignatures, a Bitcoin 2-of-2 multisignature is a tuple of two distinct signatures and not a joint signature.

III. TumbleBit: An Unlinkable Payment Hub

Our goal is to allow a *payer*, Alice $\mathcal{A}$, to unlinkably send 1 bitcoin to a *payee*, Bob $\mathcal{B}$. Naturally, if Alice $\mathcal{A}$ signed a regular Bitcoin transaction indicating that $Addr_A$ pays 1 bitcoin to $Addr_B$, then the blockchain would record a link between Alice $\mathcal{A}$ and Bob $\mathcal{B}$ and anonymity could be harmed using the techniques of [31], [42], [8]. Instead, TumbleBit funnels payments from multiple payer-payee pairs through the Tumbler $\mathcal{T}$, using cryptographic techniques to ensure that, as long as $\mathcal{T}$ does not collude with TumbleBit's users, then no one can link a payment from payer $\mathcal{A}$ to payee $\mathcal{B}$.

A. Overview of Bob's Interaction with the Tumbler

We overview TumbleBit's phases under the assumption that Bob $\mathcal{B}$ receives a single payment of value 1 bitcoin. TumbleBit's Anonymity properties require all payments made in the system to have the same denomination; we use 1 bitcoin for simplicity. In our full version [20] we also discuss how Bob can receive multiple payments of denomination 1 bitcoin each.

TumbleBit has three phases (Fig 1). Off-blockchain TumbleBit payments take place during the middle *Payment Phase*, which can last for hours or even days. Meanwhile, the first *Escrow Phase* sets up payment channels, and the last *Cash-Out Phase* closes them down; these two phases require on-blockchain transactions. All users of TumbleBit know exactly when each phase begins and ends. One way to coordinate is to use block height; for instance, if the payment phase lasts for 1 day (*i.e.,* ≈ 144 blocks) then the Escrow Phase is when block height is divisible by 144, and the Cash-Out Phase is when blockheight$+1$ is divisible by 144.

1: Escrow Phase. Every Alice $\mathcal{A}$ that wants to send payments (and Bob $\mathcal{B}$ that wants to receive payments) during the upcoming Payment Phase runs the escrow phase with $\mathcal{T}$. The escrow phase has two parts:

(a) Payee $\mathcal{B}$ asks the Tumbler $\mathcal{T}$ to set up a payment channel. $\mathcal{T}$ escrows 1 bitcoin on the blockchain via a 2-of-2 escrow transaction (Section II) denoted as $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ stipulating that 1 bitcoin can be claimed by any transaction signed by both $\mathcal{T}$ and $\mathcal{B}$. $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ is timelocked to time window tw_2, after which $\mathcal{T}$ can reclaim its bitcoin. Similarly, the payeer $\mathcal{A}$ escrows 1 bitcoin in a 2-of-2 escrow with $\mathcal{T}$ denoted as $T_{\text{escr}(\mathcal{A},\mathcal{T})}$, timelocked for time window tw_1 such that $tw_1 < tw_2$.

(b) Bob $\mathcal{B}$ obtains a puzzle z through an off-blockchain cryptographic protocol with $\mathcal{T}$ which we call the *puzzle-promise protocol*. Conceptually, the output of this protocol is a promise by $\mathcal{T}$ to pay 1 bitcoin to $\mathcal{B}$ in exchange for the solution to a puzzle z. The puzzle z is just an RSA encryption of a value ϵ

$$z = f_{RSA}(\epsilon, e, N) = \epsilon^e \mod N \quad (1)$$

where (e, N) is the TumbleBit RSA public key of the Tumbler $\mathcal{T}$. "Solving the puzzle" is equivalent to decrypting z and thus obtaining its "solution" ϵ. Meanwhile, the "promise" c is a symmetric encryption under key ϵ

$$c = \text{Enc}_\epsilon(\sigma)$$

where σ is the Tumbler's ECDSA-Secp256k1 signature on the transaction $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ which transfers the bitcoin escrowed in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ from $\mathcal{T}$ to $\mathcal{B}$. (We use ECDSA-Secp256k1 for compatibility with the Bitcoin protocol.) Thus, the solution to a puzzle z enables $\mathcal{B}$ to claim 1 bitcoin from $\mathcal{T}$. To prevent misbehavior by the Tumbler $\mathcal{T}$, our puzzle-promise protocol requires $\mathcal{T}$ to provide a proof that the puzzle solution ϵ is indeed the key which decrypts the promise ciphertext c. The details of this protocol, and its security guarantees, are in Section VI.

2: Payment Phase. Once Alice $\mathcal{A}$ indicates she is ready to pay Bob $\mathcal{B}$, Bob $\mathcal{B}$ chooses a random blinding factor $r \in Z_N^*$ and blinds the puzzle to

$$\overline{z} = r^e z \mod N. \quad (2)$$

Blinding ensures that even $\mathcal{T}$ cannot link the original puzzle z to its blinded version $\overline{z}$. Bob $\mathcal{B}$ then sends $\overline{z}$ to $\mathcal{A}$. Next, $\mathcal{A}$ solves the blinded puzzle $\overline{z}$ by interacting with $\mathcal{T}$. This *puzzle-solver protocol* is a fair exchange that ensures that $\mathcal{A}$ transfers 1 bitcoin to T iff $\mathcal{T}$ gives a valid solution to the puzzle $\overline{z}$. Finally, Alice $\mathcal{A}$ sends the solution to the blinded puzzle $\overline{\epsilon}$ back to Bob $\mathcal{B}$. Bob unblinds $\overline{\epsilon}$ to obtain the solution

$$\epsilon = \overline{\epsilon}/r \mod N \quad (3)$$

and accepts Alice's payment if the solution is valid, *i.e.,* $\epsilon^e = z \mod N$.

3: Cash-Out Phase. Bob $\mathcal{B}$ uses the puzzle solution ϵ to decrypt the ciphertext c. From the result $\mathcal{B}$ can create a transaction $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ that is signed by both $\mathcal{T}$ and $\mathcal{B}$. $\mathcal{B}$ posts $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ to the blockchain to receive 1 bitcoin from $\mathcal{T}$.

Our protocol crucially relies on the algebraic properties of RSA, and RSA blinding. To make sure that the Tumbler is using a valid RSA public key (e, N), TumbleBit also has an one-time setup phase:

0: Setup. Tumbler $\mathcal{T}$ announces its RSA public key (e, N) and Bitcoin address $Addr_T$, together with a non-interactive zero-knowledge proof of knowledge π[6] of the corresponding RSA secret key d. Every user of TumbleBit validates (e, N) using π.

[6]Such a proof could be provided using the GQ identification protocol [19] made non-interactive using the Fiat-Shamir heuristic [14] in the random oracle model.

B. Overview of Alice's Interaction with the Tumbler

We now focus on the puzzle-solving protocol between $\mathcal{A}$ and the Tumbler $\mathcal{T}$ to show how TumbleBit allows $\mathcal{A}$ to make many off-blockchain payments via only *two* on-blockchain transactions (aiding scalability).

During the Escrow Phase, Alice opens a payment channel with the Tumbler $\mathcal{T}$ by escrowing Q bitcoins in an on-blockchain transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$. Each escrowed bitcoin can pay $\mathcal{T}$ for the solution to one puzzle. Next, during the off-blockchain Payment Phase, $\mathcal{A}$ makes off-blockchain payments to $j \leq Q$ payees. Finally, during the Cash-Out Phase, Alice $\mathcal{A}$ pays the Tumbler $\mathcal{T}$ by posting a transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}(j)$ that reflects the new allocation of bitcoins; namely, that $\mathcal{T}$ holds j bitcoins, while $\mathcal{A}$ holds $Q - j$ bitcoins. The details of Alice $\mathcal{A}$'s interaction with $\mathcal{T}$, which are based on a technique used in micropayment channels [36, p. 86], are as follows:

1: Escrow Phase. Alice $\mathcal{A}$ posts a 2-of-2 escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ to the blockchain that escrows Q of Alice's bitcoins. If no valid transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}$ is posted before time window tw_1, then all Q escrowed bitcoins can be reclaimed by $\mathcal{A}$.

2: Payment Phase. Alice $\mathcal{A}$ uses her escrowed bitcoins to make off-blockchain payments to the Tumbler $\mathcal{T}$. For each payment, $\mathcal{A}$ and $\mathcal{T}$ engage in an off-blockchain puzzle-solver protocol (see Sections V-A,V-C).

Once the puzzle is solved, Alice signs and gives $\mathcal{T}$ a new transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$. $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$ points to $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and reflects the new balance between $\mathcal{A}$ and $\mathcal{T}$ (*i.e.,* that $\mathcal{T}$ holds i bitcoins while $\mathcal{A}$ holds $Q - i$ bitcoins). $\mathcal{T}$ collects a new $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$ from $\mathcal{A}$ for each payment. If Alice refuses to sign $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$, then the Tumbler refuses to help Alice solve further puzzles. Importantly, each $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$ for $i = 1...j$ (for $j < Q$) is signed by Alice $\mathcal{A}$ but *not* by $\mathcal{T}$, and is *not* posted to the blockchain.

3: Cash-Out Phase. The Tumbler $\mathcal{T}$ claims its bitcoins from $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ by signing $T_{\text{cash}(\mathcal{A},\mathcal{T})}(j)$ and posting it to the blockchain. This fulfills the condition in $T_{\text{escr}(\mathcal{A},\mathcal{T})}$, which stipulated that the escrowed coins be claimed by a transaction signed by *both* $\mathcal{A}$ and $\mathcal{T}$. (Notice that all the $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$ point to the *same* escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$. The blockchain will therefore only confirm one of these transactions; otherwise, double spending would occur. Rationally, the Tumbler $\mathcal{T}$ always prefers to confirm $T_{\text{cash}(\mathcal{A},\mathcal{T})}(j)$ since it transfers the maximum number of bitcoins to $\mathcal{T}$.) Because $T_{\text{cash}(\mathcal{A},\mathcal{T})}(j)$ is the only transaction signed by the Tumbler $\mathcal{T}$, a cheating Alice cannot steal bitcoins by posting a transaction that allocates fewer than j bitcoins to the Tumbler $\mathcal{T}$.

Remark: Scaling Bitcoin. A similar (but more elaborate) technique can be applied between $\mathcal{B}$ and $\mathcal{T}$ so that

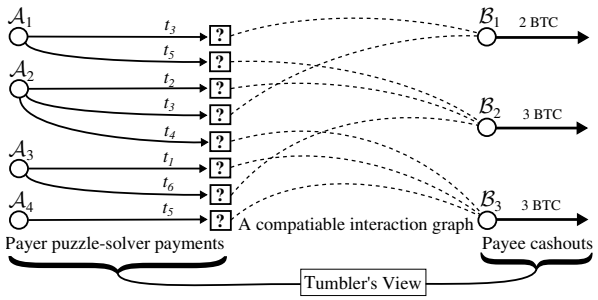

Fig. 2. Our unlinkability definition: The Tumblers view and a compatible interaction multi-graph.

only *two* on-blockchain transactions suffice for Bob $\mathcal{B}$ to receive an arbitrary number of off-blockchain payments. Details are in the full version [20]. Given that each party uses *two* on-blockchain transactions to make multiple off-blockchain playments, Tumblebit helps Bitcoin scale.

C. TumbleBit's Security Properties

Unlinkability. We assume that the Tumbler $\mathcal{T}$ does not collude with other users. The *view* of $\mathcal{T}$ consists of (1) the set of escrow transactions established between (a) each payer $\mathcal{A}_j$ and the Tumbler ($\mathcal{A}_j \overset{\text{escrow},a_i}{\longrightarrow} \mathcal{T}$) of value a_i and (b) the Tumbler and each payee $\mathcal{B}_i$ ($\mathcal{T} \overset{\text{escrow},b_i}{\longrightarrow} \mathcal{B}_i$), (2) the set of puzzle-solver protocols completed with each payer $\mathcal{A}_j$ at time t during the Payment Phase, and (3) the set of cashout transactions made by each payer $\mathcal{A}_j$ and each payee $\mathcal{B}_i$ during the Cash-Out Phase.

An *interaction multi-graph* is a mapping of payments from payers to payees (Figure 2). For each successful puzzle-solver protocol completed by payer $\mathcal{A}_j$ at time t, this graph has an edge, labeled with time t, from $\mathcal{A}_j$ to some payee $\mathcal{B}_i$. An interaction graph is *compatible* if it explains the view of the Tumbler $\mathcal{T}$, *i.e.,* the number of edges incident on $\mathcal{B}_i$ is equal to the total number of bitcoins cashed out by $\mathcal{B}_i$. Unlinkability requires all compatible interaction graphs to be equally likely. Anonymity therefore depends on the number of compatible interaction graphs.

Notice that payees $\mathcal{B}_i$ have better anonymity than payers $\mathcal{A}_j$. (This follows because the Tumbler $\mathcal{T}$ knows the time t at which payer $\mathcal{A}_j$ makes each payment. Meanwhile, the Tumbler $\mathcal{T}$ only knows the aggregate amount of bitcoins cashed-out by each payee $\mathcal{B}_i$.)

A high-level proof of TumbleBit's unlinkability is in Section VII, and the limitations of unlinkability are discussed in Section VII-C.

Balance. The system should not be exploited to print new money or steal money, *even when parties collude*. As in [18], we call this property *balance*, which establishes that no party should be able to cash-out more bitcoins than what is dictated by the payments that were successfully completed in the Payment Phase. We discuss how TumbleBit satisfies balance in Section VII.

DoS and Sybil protection. TumbleBit uses transaction fees to resist DoS and Sybil attacks. Every Bitcoin transaction can include a *transaction fee* that is paid to the Bitcoin miner who confirms the transaction on the blockchain as an incentive to confirm transactions. However, because the Tumbler $\mathcal{T}$ does not trust Alice $\mathcal{A}$ and Bob $\mathcal{B}$, $\mathcal{T}$ should not be expected to pay fees on the transactions posted during the Escrow Phase. To this end, when Alice $\mathcal{A}$ establishes a payment channel with $\mathcal{T}$, she pays for both the Q escrowed in transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and for its transaction fees. Meanwhile, when the Tumbler $\mathcal{T}$ and Bob $\mathcal{B}$ establish a payment channel, the Q escrowed bitcoins in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ are paid in the Tumbler $\mathcal{T}$, but the transaction fees are paid by Bob $\mathcal{B}$ (Section III-A). Per [9], fees raise the cost of an DoS attack where $\mathcal{B}$ starts and aborts many parallel sessions, locking $\mathcal{T}$'s bitcoins in escrow transactions. This similarly provides Sybil resistance, making it expensive for an adversary to harm anonymity by tricking a user into entering a run of TumbleBit where all other users are Sybils under the adversary's control.

IV. TumbleBit: Also a Classic Tumbler.

We can also operate TumbleBit as classic Bitcoin Tumbler. As a classic Tumbler, TumbleBit operates in epoches, each of which (roughly) requires two blocks to be confirmed on the blockchain (≈ 20 mins). During each epoch, there are exactly $\aleph$ distinct bitcoin addresses making payments (payers) and $\aleph$ bitcoin addresses receiving payments (payees). Each payment is of denomination 1 bitcoin, and the mapping from payers to payees is a bijection. During one epoch, the protocol itself is identical to that in Section III with the following changes: (1) the duration of the Payment Phase shrinks to seconds (rather than hours or days); (2) each payment channel escrows exactly $Q = 1$ bitcoin; and (3) every payee Bob $\mathcal{B}$ receives payments at an ephemeral bitcoin address $Addr_B$ chosen freshly for the epoch.

A. Anonymity Properties

As a classic tumbler, TumbleBit has the same *balance* property, but stronger anonymity: *k-anonymity within an epoch* [21], [9]. Specifically, while the blockchain reveals which payers and payees participated in an epoch, no one (not even the Tumbler $\mathcal{T}$) can tell which payer paid which payee during that specific epoch. Thus, if k payments successfully completed during an epoch, the anonymity set is of size k. (This stronger property follows directly from our unlinkability definition (Section III-C): there are k compatible interaction graphs because the interaction graph is bijection.)

Recovery from anonymity failures. It's not always the case that $k = \aleph$. The exact anonymity level achieved in an epoch can be established only after its Cash-Out Phase. For instance, anonymity is reduced to $k = \aleph - 1$ if $\mathcal{T}$ aborts an payment made by payer $\mathcal{A}_j$. We deal with this by requiring $\mathcal{B}$ to uses an *ephemeral* Bitcoin address

$Addr_B$ in each epoch. As in [21], Bob $\mathcal{B}$ discards $Addr_B$ if (1) the Tumbler $\mathcal{T}$ maliciously aborts $\mathcal{A}_j$'s payment in order to infer that $\mathcal{A}_j$ was attempting to pay $\mathcal{B}$ (see Section VIII-C); or (2) k-anonymity was too small. (In case (2), $\mathcal{B}$ can alternatively re-tumble the bitcoin in $Addr_B$ in a future epoch.)

Remark: Intersection attacks. While this notion of k-anonymity is commonly used in Bitcoin tumblers (*e.g.,* [9], [21]), it does suffer from the following weakness. Any adversary that observes the transactions posted to the blockchain within one epoch can learn which payers and payees participated in that epoch. Then, this information can be correlated to de-anonymize users across epochs (*e.g.,* using frequency analysis or techniques used to break k-anonymity [15]). See also [9], [32].

DoS and Sybil Attacks. We use fees to resist DoS and Sybil attacks. Alice again pays for both the Q escrowed in transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and for its transaction fees. However, we run into a problem if we want Bob $\mathcal{B}$ to pay the fee on the escrow transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$. Because Bob $\mathcal{B}$ uses a freshly-chosen Bitcoin address $Addr_B$, that is not linked to any prior transaction on the blockchain, $Addr_B$ cannot hold any bitcoins. Thus, Bob $\mathcal{B}$ will have to pay the Tumbler $\mathcal{T}$ out of band. The anonymous fee vouchers described in [21] provide one way to address this, which also has the additional feature that payers $\mathcal{A}$ cover all fees.

V. A Fair Exchange for RSA Puzzle Solving

We now explain how to realize a Bitcoin-compatible *fair-exchange* where Alice $\mathcal{A}$ pays Tumbler $\mathcal{T}$ one bitcoin iff the $\mathcal{T}$ provides a valid solution to an RSA puzzle. The Tumbler $\mathcal{T}$ has an RSA secret key d and the corresponding public key (e, N). The RSA puzzle y is provided by Alice, and its solution is an RSA secret-key exponentiation

$$\epsilon = f_{RSA}^{-1}(y, d, N) = y^d \bmod N \tag{4}$$

The puzzle solution is essentially an RSA decryption or RSA signing operation.

This protocol is at the heart of TumbleBit's Payment Phase. However, we also think that this protocol is of independent interest, since there is also a growing interest in techniques that can fairly exchange a bitcoin for the solution to a computational "puzzle". (The full version [20] reviews the related work [27], [30], [24], [6].) Section V-A presents our RSA-puzzle-solver protocol as a stand-alone protocol that requires two blocks to be confirmed on the blockchain. Our protocol is fast—solving 2048-bit RSA puzzles faster than [30]'s protocol for solving 16x16 Sudoku puzzles (Section VIII)). Also, the use of RSA means that our protocol supports solving *blinded* puzzles (see equation (2)), and thus can be used to create an unlinkable payment scheme. Section V-C shows how our protocol is integrated into TumbleBit's Payment Phase. Implementation results are in Table II of Section VIII-B.

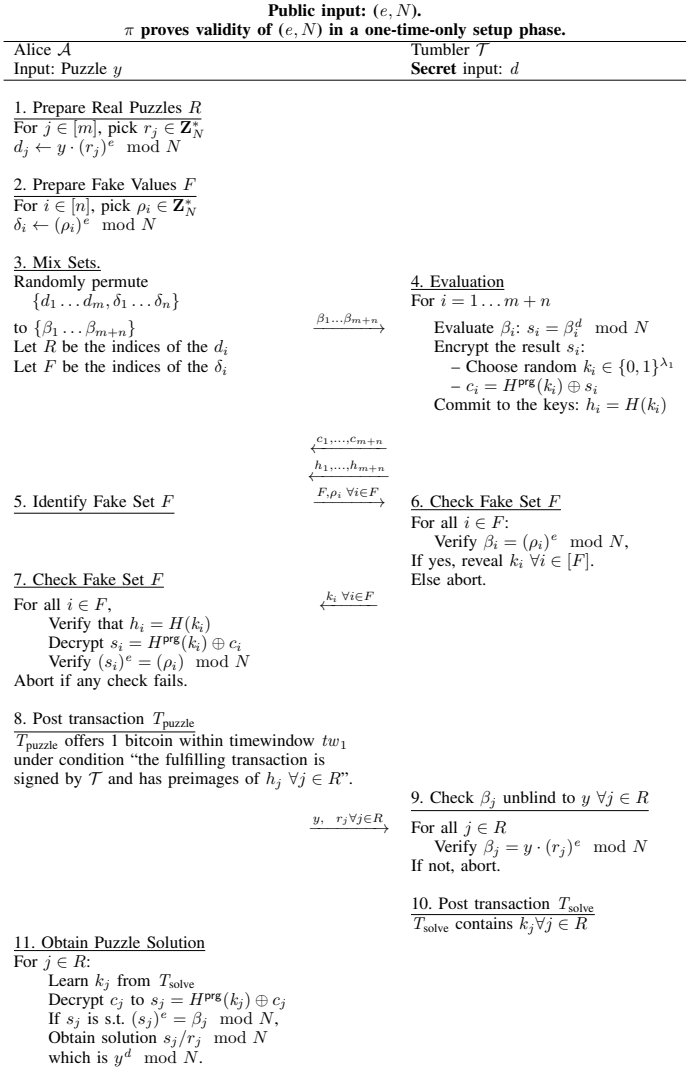

Fig. 3. RSA puzzle solving protocol. H and H^{prg} are modeled as random oracles. In our implementation, H is RIPEMD-160, and H^{prg} is ChaCha20 with a 128-bit key, so that $\lambda_1 = 128$.

A. Our (Stand-Alone) RSA-Puzzle-Solver Protocol

The following stand-alone protocol description assumes Alice $\mathcal{A}$ wants to transfer 1 bitcoin in exchange for one puzzle solution. Section V-C shows how to support the transfer of up to Q bitcoins for Q puzzle solutions (where each solution is worth 1 bitcoin).

The core idea is similar to that of contingent payments [27]: Tumbler $\mathcal{T}$ solves Alice's $\mathcal{A}$'s puzzle y by computing the solution $y^d \mod N$, then encrypts the solution under a randomly chosen key k to obtain a ciphertext c, hashes the key k under bitcoin's hash as $h = H(k)$ and finally, provides (c, h) to Alice. Alice $\mathcal{A}$ prepares T_{puzzle} offering one bitcoin in exchange for the preimage of h. Tumbler $\mathcal{T}$ earns the bitcoin by posting a transaction T_{solve} that contains k, the preimage of h, and thus fulfills the condition in T_{puzzle} and claims a

bitcoin for $\mathcal{T}$. Alice $\mathcal{A}$ learns k from T_{solve}, and uses k to decrypt c and obtain the solution to her puzzle.

Our challenge is to find a mechanism that allows $\mathcal{A}$ to validate that c is the encryption of the correct value, without using ZK proofs. Thus, instead of asking $\mathcal{T}$ to provide just *one* (c, h) pair, $\mathcal{T}$ will be asked to provide $m + n$ pairs (Step 3). Then, we use cut and choose: $\mathcal{A}$ asks $\mathcal{T}$ to "open" n of these pairs, by revealing the randomly-chosen keys k_i's used to create each of the n pairs (Step 7). For a malicious $\mathcal{T}$ to successfully cheat $\mathcal{A}$, it would have to correctly guess all the n "challenge" pairs and form them properly (so it does not get caught), while at the same time malforming *all* the m unopened pairs (so it can claim a bitcoin from $\mathcal{A}$ without actually solving the puzzle). Since $\mathcal{T}$ cannot predict which pairs $\mathcal{A}$ asks it to open, $\mathcal{T}$ can only cheat with very low probability $1/\binom{m+n}{n}$.

However, we have a problem. Why should $\mathcal{T}$ agree to open *any* of the (c, h) values that it produced? If $\mathcal{A}$ received the opening of a correctly formed (c, h) pair, she would be able to obtain a puzzle solution without paying a bitcoin. As such, we introduce the notion of "fake values". Specifically, the n (c, h)-pairs that $\mathcal{A}$ asks $\mathcal{T}$ to open will open to "fake values" rather than "real" puzzles. Before $\mathcal{T}$ agrees to open them (Step 7), $\mathcal{A}$ must prove that these n values are indeed fake (Step 6).

We must also ensure that $\mathcal{T}$ cannot distinguish "real puzzles" from "fake values". We do this with RSA blinding. The real puzzle y is blinded m times with different RSA-blinding factors (Step 1), while the n fake values are RSA-blinded as well (Step 2). Finally, $\mathcal{A}$ randomly permutes the real and fake values (Step 3).

Once Alice confirms the correctness of the opened "fake" (c, h) values (Step 7), she signs a transaction T_{puzzle} offering one bitcoin for the keys k that open all of the m "real" (c, h) values (Step 8). But what if Alice cheated, so that each of the "real" (c, h) values opened to the solution to a *different* puzzle? This would not be fair to $\mathcal{T}$, since $\mathcal{A}$ has only paid for the solution to a single puzzle, but has tricked $\mathcal{T}$ into solving multiple puzzles. We solve this problem in Step 9: once $\mathcal{A}$ posts T_{puzzle}, she proves to $\mathcal{T}$ that all m "real" values open to *the same* puzzle y. This is done by revealing the RSA-blinding factors blinding puzzle y. Once $\mathcal{T}$ verifies this, $\mathcal{T}$ agrees to post T_{solve} which reveals m of the k values that open "real" (c, h) pairs (Step 10).

B. Fair Exchange

Fair exchange exchange entails the following: (1) *Fairness for $\mathcal{T}$:* After one execution of the protocol $\mathcal{A}$ will learn the correct solution $y^d \mod N$ to at most one puzzle y of her choice. (2) *Fairness for $\mathcal{A}$:* $\mathcal{T}$ will earn 1 bitcoin iff $\mathcal{A}$ obtains a correct solution.

We prove this using the real-ideal paradigm [16]. We call the ideal functionality $\mathcal{F}_{\mathsf{fair\text{-}RSA}}$ and present it the full version [20]. $\mathcal{F}_{\mathsf{fair\text{-}RSA}}$ acts like a trusted party between $\mathcal{A}$ and $\mathcal{T}$. $\mathcal{F}_{\mathsf{fair\text{-}RSA}}$ gets a puzzle-solving request (y, 1 bitcoin) from $\mathcal{A}$, and forwards the request to $\mathcal{T}$. If $\mathcal{T}$ agrees to solve puzzle y for $\mathcal{A}$, then $\mathcal{T}$ gets 1 bitcoin and $\mathcal{A}$ gets the puzzle solution. Otherwise, if $\mathcal{T}$ refuses, $\mathcal{A}$ gets 1 bitcoin back, and $\mathcal{T}$ gets nothing. Fairness for $\mathcal{T}$ is captured because $\mathcal{A}$ can request a puzzle solution only if she sends 1 bitcoin to $\mathcal{F}_{\mathsf{fair\text{-}RSA}}$. Fairness for $\mathcal{B}$ is captured because $\mathcal{T}$ receives 1 bitcoin only if he agrees to solve the puzzle. The following theorem is proved in the full version [20]:

Theorem 1: Let λ be the security parameter, m, n be statistical security parameters, let $N > 2^\lambda$. Let π be a publicly verifiable zero-knowledge proof of knowledge in the random oracle model. If the RSA assumption holds in $\mathbf{Z}_N^*$, and if functions H^{prg}, H are independent random oracles, there exists a negligible function ν, such that protocol in Figure 3 securely realizes $\mathcal{F}_{\mathsf{fair\text{-}RSA}}$ in the random oracle model with the following security guarantees. The security for $\mathcal{T}$ is $1-\nu(\lambda)$ while security for $\mathcal{A}$ is $1 - \frac{1}{\binom{m+n}{n}} - \nu(\lambda)$.

C. Solving Many Puzzles and Moving Off-Blockchain

To integrate the protocol in Figure 3 into TumbleBit, we have to deal with three issues. First, if TumbleBit is to scale Bitcoin (Section III-B), then Alice $\mathcal{A}$ must be able to use only *two* on-blockchain transactions $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and $T_{\text{cash}(\mathcal{A},\mathcal{T})}$ to pay for the an *arbitrary* number of Q puzzle solutions (each worth 1 bitcoin) during the Payment Phase; the protocol in Figure 3 only allows for the solution to a single puzzle. Second, per Section III-B, the puzzle-solving protocol should occur entirely off-blockchain; the protocol in Figure 3 uses two *on-blockchain* transactions T_{puzzle} and T_{solve}. Third, the T_{solve} transactions are longer than typical transactions (since they contain m hash preimages), and thus require higher transaction fees.

To deal with these issues, we now present a fair-exchange protocol that uses only *two* on-blockchain transactions to solve an *arbitrary* number of RSA puzzles.

Escrow Phase. Before puzzle solving begins, Alice posts a 2-of-2 escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ to the blockchain that escrows Q bitcoins, (per Section III-B). $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ is timelocked to time window tw_1, and stipulates that the escrowed bitcoins can be transferred to a transaction signed by both $\mathcal{A}$ and $\mathcal{T}$.

Payment Phase. Alice $\mathcal{A}$ can buy solutions for up to Q puzzles, paying 1 bitcoin for each. Tumbler $\mathcal{T}$ keeps a counter of how many puzzles it has solved for $\mathcal{A}$, making sure that the counter does not exceed Q. When $\mathcal{A}$ wants her i^{th} puzzle solved, she runs the protocol in Figure 3 with the following modifications after Step 8 (so that it runs entirely off-blockchain):

(1) Because the Payment Phase is off-blockchain, transaction T_{puzzle} from Figure 3 is *not posted* to the

blockchain. Instead, Alice $\mathcal{A}$ forms and signs transaction T_{puzzle} and sends it to the Tumbler $\mathcal{T}$. Importantly, Tumbler $\mathcal{T}$ does *not* sign or post this transaction yet.

(2) Transaction T_{puzzle} points to the escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$; T_{puzzle} changes its balance so that $\mathcal{T}$ holds i bitcoin and Alice $\mathcal{A}$ holds $Q-i$ bitcoins. T_{puzzle} is timelocked to time window $tw1$ and stipulates the same condition in Figure 3: "the fulfilling transaction is signed by T and has preimages of $h_j \forall j \in R$."

(Suppose that $\mathcal{T}$ deviates from this protocol, and instead immediately signs and post T_{puzzle}. Then the bitcoins in $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ would be transferred to T_{puzzle}. However, these bitcoins would remain locked in T_{puzzle} until either (a) the timelock tw expired, at which point Alice $\mathcal{A}$ could reclaim her bitcoins, or (b) $\mathcal{T}$ signs and posts a transaction fulfilling the condition in T_{puzzle}, which allows Alice to obtain the solution to her puzzle.)

(3) Instead of revealing the preimages $k_j \forall j \in R$ in an on-blockchain transaction T_{solve} as in Figure 3, the Tumbler $\mathcal{T}$ just sends the preimages directly to Alice.

(4) Finally, Alice $\mathcal{A}$ checks that the preimages open a valid puzzle solution. If so, Alice signs a regular cash-out transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}$ (per Section III-B). $T_{\text{cash}(\mathcal{A},\mathcal{T})}$ points to the escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and reflects the new balance between $\mathcal{A}$ and $\mathcal{T}$.

At the end of the i^{th} payment, the Tumbler $\mathcal{T}$ should have two new signed transactions from Alice: $T_{\text{puzzle}}(i)$ and $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$, each reflecting the (same) balance of bitcoins between $\mathcal{T}$ (holding i bitcoins) and $\mathcal{A}$ (holding $Q-i$ bitcoins). However, Alice $\mathcal{A}$ already has her puzzle solution at this point (step (4) modification above). What if she refuses to sign $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$?

In this case, the Tumbler immediately begins to cash out, even without waiting for the Cash-Out Phase. Specifically, Tumbler $\mathcal{T}$ holds transaction $T_{\text{puzzle}}(i)$, signed by $\mathcal{A}$, which reflects a correct balance of i bitcoins to $\mathcal{T}$ and $Q-i$ bitcoins to $\mathcal{A}$. Thus, $\mathcal{T}$ signs $T_{\text{puzzle}}(i)$ and posts it to the blockchain. Then, $\mathcal{T}$ claims the bitcoins locked in $T_{\text{puzzle}}(i)$ by signing and posting transaction T_{solve}. As in Figure 3, T_{solve} fulfills T_{puzzle} by containing the m preimages $k_j \forall j \in R$. The bitcoin in $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ will be transferred to T_{puzzle} and then to T_{solve} and thus to the Tumbler $\mathcal{T}$. The only harm done is that $\mathcal{T}$ posts two longer transactions $T_{\text{puzzle}}(i), T_{\text{solve}}(i)$ (instead of just $T_{\text{cash}(\mathcal{A},\mathcal{T})}$), which require higher fees to be confirmed on the blockchain. (Indeed, this is why we have introduced the $T_{\text{cash}(\mathcal{A},\mathcal{T})}(i)$ transaction.)

Cash-Out Phase. Alice has j puzzle solutions once the the Payment Phase is over and the Cash-Out Phase begins. If the Tumbler $\mathcal{T}$ has a transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}(j)$ signed by Alice, the Tumbler $\mathcal{T}$ just signs and post this transaction to the blockchain, claiming its j bitcoins.

VI. Puzzle-Promise Protocol

We present the puzzle-promise protocol run between $\mathcal{B}$ and $\mathcal{T}$ in the Escrow Phase. Recall from Section III-A, that the goal of this protocol is to provide Bob $\mathcal{B}$ with a puzzle-promise pair (c, z). The "promise" c is an encryption (under key ϵ) of the Tumbler's ECDSA-Secp256k1 signature σ on the transaction $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ which transfers the bitcoin escrowed in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ from $\mathcal{T}$ to $\mathcal{B}$. Meanwhile the RSA-puzzle z hides the encryption key ϵ per equation (1).

If Tumbler $\mathcal{T}$ just sent a pair (c, z) to Bob, then Bob has no guarantee that the promise c is actually encrypting the correct signature, or that z is actually hiding the correct encryption key. On the other hand, $\mathcal{T}$ cannot just reveal the signature σ directly, because Bob could use σ to claim the bitcoin escrowed in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ without actually being paid (off-blockchain) by Alice $\mathcal{A}$ during TumbleBit's Payment Phase.

To solve this problem, we again use cut and choose: we ask $\mathcal{T}$ to compute many puzzle-promise pairs (c_i, z_i), and have Bob $\mathcal{B}$ test that some of the pairs are computed correctly. As in Section V-A, we use "fake" transactions (that will be "opened" and used only to check if the other party has cheated) and "real" transactions (that remain "unopened" and result in correctly-formed puzzle-promise pairs). Cut-and-choose guarantees that $\mathcal{B}$ knows that *at least one* of the unopened pairs is correctly formed. However, how does $\mathcal{B}$ know which puzzle z_i is correctly formed? Importantly, $\mathcal{B}$ can only choose one puzzle z_i that he will ask Alice $\mathcal{A}$ to solve during TumbleBit's Payment Phase (Section III-A). To deal with this, we introduce an *RSA quotient-chain technique* that ties together all puzzles z_i so that solving one puzzle z_{j1} gives the solution to all other puzzles.

In this section, we assume that $\mathcal{B}$ wishes to obtain only a single payment of denomination 1 bitcoin; the protocol as described in Figure 4 and Section VI-A suffices to run TumbleBit as a classic tumbler. We discuss its security properties in Section VI-B and implementation in Section VIII-B. In the full version [20], we show how to modify this protocol so that it allows $\mathcal{B}$ to receive arbitrary number of Q off-blockchain payments using only two on-blockchain transactions.

A. Protocol Walk Through

$\mathcal{B}$ prepares μ distinct "real" transactions and η "fake" transactions, hides them by hashing them with H' (Step 2-3), permutes them (Step 4), and finally sends them to $\mathcal{T}$ as $\beta_1, ..., \beta_{m+n}$. $\mathcal{T}$ the evaluates each β_i to obtain a puzzle-promise pair (c_i, z_i) (Step 5).

Next, $\mathcal{B}$ needs to check that the η "fake" (c_i, z_i) pairs are correctly formed by $\mathcal{T}$ (Step 8). To do this, $\mathcal{B}$ needs $\mathcal{T}$ to provide the solutions ϵ_i to the puzzles z_i in fake pairs. $\mathcal{T}$ reveals these solutions only after $\mathcal{B}$ has proved that the η pairs really are fake (Step 7).

Once this check is done, $\mathcal{B}$ knows that $\mathcal{T}$ can cheat with probability less than $1/\binom{\mu+\eta}{\eta}$.

Now we need our new trick. We want to ensure that if *at least one* of the "real" (c_i, z_i) pairs opens to a valid ECDSA-Secp256k1 signature σ_i, then just one puzzle solution ϵ_i with $i \in R$, can be used to open this pair. (We need this because $\mathcal{B}$ must decide which puzzle z_i to give to the payer $\mathcal{A}$ for decryption without knowing which pair (c_i, z_i) is validly formed.) We solve this by having $\mathcal{T}$ provide $\mathcal{B}$ with $\mu - 1$ quotients (Step 9). This solves our problem since knowledge of $\epsilon = \epsilon_{j_1}$ allows $\mathcal{B}$ to recover of *all* other ϵ_{j_i}, since

$$\epsilon_{j_i} = \epsilon_1 \cdot q_2 \cdot, \ldots, \cdot q_i$$

On the flip side, what if $\mathcal{B}$ obtains *more than one* valid ECDSA-Secp256k1 signatures by opening the (c_i, z_i) pairs? Fortunately, however, we don't need to worry about this. The escrow transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ offers 1 bitcoin in exchange for a ECDSA-Secp256k1 signature under an *ephemeral* key $PK_{\mathcal{T}}^{eph}$ used *only once* during this protocol execution with this specific payee $\mathcal{B}$. Thus, even if $\mathcal{B}$ gets many signatures, only one can be used to form the cash-out transaction $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ that redeems the bitcoin escrowed in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$.

B. Security Properties

We again capture the security requirements of the puzzle-promise protocol using real-ideal paradigm [16]. The ideal functionality $\mathcal{F}_{\text{promise-sign}}$ is presented the full version [20]. $\mathcal{F}_{\text{promise-sign}}$ is designed to guarantee the following properties: (1) *Fairness for $\mathcal{T}$:* Bob $\mathcal{B}$ learns nothing except signatures on *fake* transactions. (2) *Fairness for $\mathcal{B}$:* If $\mathcal{T}$ agrees to complete the protocol, then Bob $\mathcal{B}$ obtains at least one puzzle-promise pair. To do this, $\mathcal{F}_{\text{promise-sign}}$ acts a trusted party between $\mathcal{B}$ and $\mathcal{T}$. Bob $\mathcal{B}$ sends the "real" and "fake" transactions to $\mathcal{F}_{\text{promise-sign}}$. $\mathcal{F}_{\text{promise-sign}}$ has access to an oracle that can compute the Tumbler's $\mathcal{T}$ signatures on any messages. (This provides property (2).) Then, if Tumbler $\mathcal{T}$ agrees, $\mathcal{F}_{\text{promise-sign}}$ provides Bob $\mathcal{B}$ with signatures on each "fake" transaction only. (This provides property (1).) The following theorem is proved the full version [20]:

Theorem 2: Let λ be the security parameter. If RSA trapdoor function is hard in $\mathbf{Z}_N^*$, if H, H', H^{shk} are independent random oracles, if ECDSA is strong existentially unforgeable signature scheme, then the puzzle-promise protocol in Figure 4 securely realizes the $\mathcal{F}_{\text{promise-sign}}$ functionality. The security for $\mathcal{T}$ is $1 - \nu(\lambda)$ while security for $\mathcal{B}$ is $1 - \frac{1}{\binom{\mu+\eta}{\eta}} - \nu(\lambda)$.

VII. TumbleBit Security

We discuss TumbleBit's unlinkability and balance properties. See Section III-C for DoS/Sybil resistance.

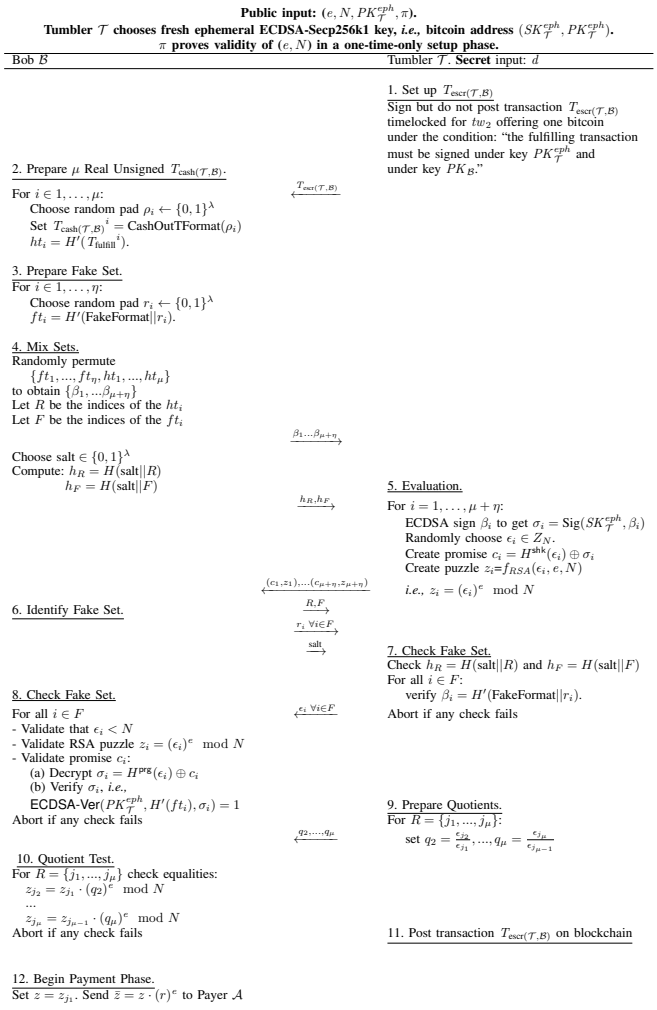

Fig. 4. Puzzle-promise protocol when $Q = 1$. $(d, (e, N))$ are the RSA keys for the tumbler $\mathcal{T}$. (Sig, ECDSA-Ver) is an ECDSA-Secp256k1 signature scheme. We model H, H' and H^{shk} as random oracles. In our implementation, H is HMAC-SHA256 (keyed with salt) . H' is 'Hash256', *i.e.,* SHA-256 cascaded with itself, which is the hash function used in Bitcoin's "hash-and-sign" paradigm with ECDSA-Secp256k1. H^{shk} is SHA-512. CashOutTFormat is shorthand for the unsigned portion of a transaction that fulfills $T_{\text{escr}(\mathcal{T},\mathcal{B})}$. The protocol uses ρ_i to ensure the output of CashOutTFormat contains sufficient entropy. FakeFormat is a distinguishable public string.

A. Balance

The balance was defined, at high-level, in Section III-C. We analyze balance in several cases.

Tumbler $\mathcal{T}^*$ is corrupt. We want to show that all the bitcoins paid to $\mathcal{T}$ by all $\mathcal{A}_j$'s can be later claimed by the $\mathcal{B}_i$'s. (That is, a malicious $\mathcal{T}^*$ cannot refuse a payment to Bob after being paid by Alice.) If $\mathcal{B}_i$ successfully completes the puzzle-promise protocol with $\mathcal{T}^*$, fairness for this protocol guarantees that $\mathcal{B}_i$ gets a correct "promise" c and puzzle z. Meanwhile, the fairness of the puzzle-solver protocol guarantees that each $\mathcal{A}_j$ gets a correct puzzle solution in exchange for

her bitcoin. Thus, for any puzzle z solved, some $\mathcal{B}_i$ can open promise c and form the cash-out transaction $T_{\text{cash}(\mathcal{T},\mathcal{B})}$ that allows $\mathcal{B}_i$ to claim one bitcoin. Moreover, transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ has timelock tw_1 and transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ has timelock tw_2. Since $tw_1 < tw_2$, it follows that either (1) $\mathcal{T}^*$ solves $\mathcal{A}$'s puzzle or (2) $\mathcal{A}$ reclaims the bitcoins in $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ (timelock tw_1), before $\mathcal{T}$ can (3) steal a bitcoin by reclaiming the bitcoins in $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ (timelock tw_2).

Case $\mathcal{A}_j^*$ and $\mathcal{B}_i^*$ are corrupt. Consider colluding payers $\mathcal{B}_i^*$ and payees $\mathcal{A}_j^*$. We show that the sum of bitcoins cashed out by all $\mathcal{B}_i^*$ is no more than the number of puzzles solved by $\mathcal{T}$ in the Payment Phase with all $\mathcal{A}_j^*$.

First, the fairness of the puzzle-promise protocol guarantees that any $\mathcal{B}_i^*$ learns only (c, z) pairs; thus, by the unforgeability of ECDSA signatures and the hardness of solving RSA puzzles, $\mathcal{B}^*$ cannot claim any bitcoin at the end of the Escrow Phase. Next, the fairness of the puzzle-solver protocol guarantees that if $\mathcal{T}$ completes SP_j successful puzzle-solver protocol executions with $\mathcal{A}_j^*$, then $\mathcal{A}_j^*$ gets the solution to exactly SP_j puzzles. Payees $\mathcal{B}_i^*$ use the solved puzzles to claim bitcoins from $\mathcal{T}$. By the unforgeability of ECDSA signatures (and assuming that the blockchain prevents double-spending), all colluding $\mathcal{B}_i^*$ cash-out no more than $\min(t, \sum_j SP_j)$ bitcoin in total, where t is the total number of bitcoins escrowed by $\mathcal{T}$ across all $\mathcal{B}_i^*$.

Case $\mathcal{B}_i^*$ and $\mathcal{T}$ collude. Now suppose that $\mathcal{B}_i^*$ and $\mathcal{T}^*$ collude to harm $\mathcal{A}_j$. Fairness for $\mathcal{A}_j$ still follows directly from the fairness of the puzzle-solver protocol. This follows because the only interaction between $\mathcal{A}_j$ and $\mathcal{B}_i^*$ is the exchange of a puzzle (and its solution). No other secret information about $\mathcal{A}_j$ is revealed to $\mathcal{B}_i^*$. Thus, $\mathcal{B}_i^*$ cannot add any additional information to the view of $\mathcal{T}$, that $\mathcal{T}$ can use to harm fairness for $\mathcal{A}_j$.

We do note, however, that an irrational Bob $\mathcal{B}_i^*$ can misbehave by handing Alice $\mathcal{A}_j$ an incorrect puzzle z^*. In this case, the fairness of the puzzle-solver protocol ensures that Alice $\mathcal{A}_j$ will pay the Tumbler $\mathcal{T}$ for a correct solution ϵ^* to puzzle z^*. As such, Bob $\mathcal{B}_i$ will be expected to provide Alice $\mathcal{A}_j$ with the appropriate goods or services in exchange for the puzzle solution ϵ^*. However, the puzzle solution ϵ^* will be of no value to Bob $\mathcal{B}_i$, *i.e.,* Bob cannot use ϵ^* to claim a bitcoin during the Cash-Out Phase. It follows that the only party harmed by this misbehavior is Bob $\mathcal{B}_i$ himself. As such, we argue that such an attack is of no importance.

Case $\mathcal{A}_j^*$ and $\mathcal{T}$ collude. Similarly, even if $\mathcal{A}_j^*$ and $\mathcal{T}$ collude, fairness for an honest $\mathcal{B}_i$ still follows from the fairness of the puzzle-promise protocol. This is because $\mathcal{A}_j^*$'s interaction with $\mathcal{B}_i$ is restricted in receiving a puzzle z, and handing back a solution. While $\mathcal{A}_j^*$ can always give $\mathcal{B}_i$ an invalid solution ϵ^*, $\mathcal{B}_i$ can easily check that the solution is invalid (since $(\epsilon^*)^e \neq z \mod N$) and refuse to provide goods or services.

Case $\mathcal{A}_j^*$, $\mathcal{B}_i^*$ and $\mathcal{T}$ collude. Suppose $\mathcal{A}_j^*$, $\mathcal{B}_i^*$ and $\mathcal{T}$ all collude to harm some other honest $\mathcal{A}$ and/or $\mathcal{B}$. This can be reduced to one of the two cases above because an honest $\mathcal{A}$ will only interact with $\mathcal{B}_i^*$ and $\mathcal{T}^*$, while an honest $\mathcal{B}$ will only interact with $\mathcal{A}_j^*$ and $\mathcal{T}$.

B. Unlinkability

Unlinkability is defined in Section III-C and must hold against a $\mathcal{T}$ that *does not collude* with other users. We show that *all* interaction multi-graphs $\mathcal{G}$ compatible with $\mathcal{T}$'s view are equally likely.

First, note that all TumbleBit payments have the same denomination (1 bitcoin). Thus, $\mathcal{T}^*$ cannot learn anything by correlating the values in the transactions. Next, recall from Section III-A, that all users of TumbleBit coordinate on phases and epochs. Escrow transactions are posted at the same time, during the Escrow Phase only. All $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ cash-out transactions are posted during the Cash-Out Phase only. All payments made from $\mathcal{A}_i$ and $\mathcal{B}_j$ occur during the Payment Phase only, and payments involve no direct interaction between $\mathcal{T}$ and $\mathcal{B}$. This rules out timing attacks where the Tumbler purposely delays or speeds up its interaction with some payer $\mathcal{A}_j$, with the goal of distinguishing some behavior at the intended payee $\mathcal{B}_i$. Even if the Tumbler $\mathcal{T}^*$ decides to cash-out with $\mathcal{A}_j$ before the Payment Phase completes (as is done in Section V-C when $\mathcal{A}_j$ misbehaves), all the $\mathcal{B}_i$ still cash out at the same time, during the Cash-Out Phase.

Next, observe that transcripts of the puzzle-promise and puzzle-solver protocols are information-theoretically unlinkable. This follows because the puzzle $\overline{z}$ used by any $\mathcal{A}_j$ in the puzzle-solver protocol is *equally likely* to be the blinding of *any* of the puzzles z that appear in the puzzle-promise protocols played with any $\mathcal{B}_i$ (see Section III-A, equation (2)).

Finally, we assume secure channels, so that $\mathcal{T}^*$ cannot eavesdrop on communication between $\mathcal{A}_j$'s and $\mathcal{B}_i$'s, and that $\mathcal{T}^*$ cannot use network information to correlate $\mathcal{A}_j$'s and $\mathcal{B}_i$'s (by *e.g.,* observing that they share the same IP address). Then, the above two observations imply that all interaction multi-graphs, that are compatible with the view of $\mathcal{T}^*$, are equally likely.

C. Limitations of Unlinkability

TumbleBit's unlinkability (see Section III-C) is inspired by Chaumian eCash [12], and thus suffers from similar limitations. (The full version [20] discusses the limitations of Chaumian eCash [12] in more detail.) In what follows, we assume that Alice has a single Bitcoin address $Addr_{\mathcal{A}}$, and Bob has Bitcoin address $Addr_{\mathcal{B}}$.

Alice/Tumbler collusion. Our unlinkability definition assumes that the Tumbler does not collude with other TumbleBit users. However, collusion between the Tumbler and Alice can be used in a *ceiling attack*. Suppose that some Bob has set up a TumbleBit payment

channel that allows him to accept up to Q TumbleBit payments, and suppose that Bob has already accepted Q payments at time t_0 of the Payment Phase. Importantly, the Tumbler, working alone, cannot learn that Bob is no longer accepting payments after time t_0. (This follows because the Tumbler and Bob do not interact during the Payment Phase.) However, the Tumbler can learn this by colluding with Alice: Alice offers to pay Bob at time t_0, and finds that Bob cannot accept her payment (because Bob has "hit the ceiling" for his payment channel). Now the Tumbler knows that Bob has obtained Q payments at time t_0, and he can rule out any compatible interaction graphs that link any payment made after time t_0 to Bob.

If we can prevent ceiling attacks (*e.g.,* by requiring Bob to initiate every interaction with Alice) then Bob's puzzle z *cannot* be linked to *any* payee's Bitcoin address $Addr_{\mathcal{B}_1}, ..., Addr_{\mathcal{B}_\iota}$, even if Alice and the Tumbler collude; see the full version [20].

Bob/Tumbler collusion. Bob and the Tumbler can collude to learn the true identity of Alice. Importantly, this collusion attack is useful only if Bob can be paid by Alice without learning her true identity (*e.g.,* if Alice is a Tor user). The attack is simple. Bob reveals the blinded puzzle value $\overline{z}$ to the Tumbler. Now, when Alice asks that Tumbler to solve puzzle $\overline{z}$, the Tumbler knows that this Alice is attempting to pay Bob. Specifically, the Tumbler learns that Bob was paid by the Bitcoin address $Addr_{\mathcal{A}}$ that paid for the solution to puzzle $\overline{z}$.

There is also a simple way to mitigate this attack. Alice chooses a fresh random blinding factor $r' \in Z_N^*$ and asks the Tumbler to solve the double-blinded puzzle

$$\overline{\overline{z}} = (r')^e \cdot \overline{z} \mod N. \tag{5}$$

Once the Tumbler solves the double-blinded puzzle $\overline{\overline{z}}$, Alice can unblind it by dividing by r' and recovering the solution to single-blinded puzzle $\overline{z}$. This way, the Tumbler cannot link the double-blinded puzzle $\overline{\overline{z}}$ from Alice to the single-blinded puzzle $\overline{z}$ from Bob.

However, even with double blinding, there is still a timing channel. Suppose Bob colludes with the Tumbler, and sends the blinded puzzle value $\overline{z}$ to both Alice and the Tumbler at time t_0. The Tumbler can rule out the possibility that any payment made by any Alice prior to time t_0 should be linked to this payment to Bob. Returning to the terminology of our unlinkability definition (Section III-C), this means that Bob and the Tumbler can collude to use timing information to rule out some compatible interaction graphs.

Potato attack. Our definition of unlinkability does not consider external information. Suppose Bob sells potatoes that costs exactly 7 bitcoins, and the Tumbler knows that no other payee sells items that cost exactly 7 bitcoins. The Tumbler can use this external information rule out compatible interaction graphs. For instance, if Alice made 6 TumbleBit payments, the Tumbler infers that Alice could not have bought Bob's potatoes.

Intersection attacks. Our definition of unlinkability applies only to a single epoch. Thus, as mentioned in Section IV-A and [9], [32], our definition does not rule out the correlation of information across epochs.

Abort attacks. Our definition of unlinkability applies to payments that complete during an epoch. It does not account for information gained by strategically aborting payments. As an example, suppose that the Tumbler notices that during several TumbleBit epochs, (1) Alice always makes a single payment, and (2) Bob hits the ceiling for his payment channel. Now in the next epoch, the Tumbler aborts Alice's payment and notices that Bob no longer hits his ceiling. The Tumbler might guess that Alice was trying to pay Bob.

VIII. Implementation

To show that TumbleBit is performant and compatible with Bitcoin, we implemented TumbleBit as a classic tumbler. (That is, each payer and payee can send/receive $Q = 1$ payment/epoch.) We then used TumbleBit to mix bitcoins from 800 payers (Alice $\mathcal{A}$) to 800 payees (Bob $\mathcal{B}$). We describe how our implementation instantiates our TumbleBit protocols. We then measure the off-blockchain performance, *i.e.,* compute time, running time, and bandwidth consumed. Finally, we describe two on-blockchain tests of TumbleBit.

A. Protocol Instantiation

We instantiated our protocols with 2048-bit RSA. The hash functions and signatures are instantiated as described in the captions to Figure 3 and Figure 4.[7]

Choosing m and n in the puzzle-solving protocol. Per Theorem 1, the probability that $\mathcal{T}$ can cheat is parameterized by $1/\binom{m+n}{m}$ where m is the number of "real" values and n is the number of "fake" values in Figure 3. From a security perspective, we want m and n to be as large as possible, but in practice we are constrained by the Bitcoin protocol. Our main constraint is that m RIPEMD-160 hash outputs must be stored in T_{puzzle} of our puzzle-solver protocol. Bitcoin P2SH scripts (as described below) are limited in size to 520 bytes, which means $m \leq 21$. Increasing m also increases the transaction fees. Fortunately, n is not constrained by the Bitcoin protocol; increasing n only means we perform more off-blockchain RSA exponentiations. Therefore, we chose $m = 15$ and $n = 285$ to bound $\mathcal{T}$'s cheating probability to 2^{-80}. (2^{-80} equals RIPEMD-160's collision probability.)

[7] There were slight difference between our protocols as described in this paper and the implementation used in some of the tests. In Figure 3, $\mathcal{A}$ reveals blinds $r_j \forall j \in R$ to $\mathcal{T}$, our implementation instead reveals an encrypted version $r_j^e \forall j \in R$. This change does not affect performance, since $\mathcal{A}$ hold both r_j and r_j^e. Also, our implementation omits the index hashes h_R and h_F from Figure 4; these are two 256-bit hash outputs and thus should not significantly affect performance. We have since removed these differences.

Choosing μ and η in the puzzle-promise protocol. Theorem 2 also allows $\mathcal{T}$ to cheat with probability $1/\binom{\mu+\eta}{\mu}$. However, this protocol has no Bitcoin-related constraints on μ and η. Thus, we take $\mu = \eta = 42$ to achieve a security level of 2^{-80} while minimizing the number of off-blockchain RSA computations performed in Figure 4 (which is $\mu + \eta$).

Scripts. By default, Bitcoin clients and miners only operate on transactions that fall into one of the five standard Bitcoin transaction templates. We therefore conform to the Pay-To-Script-Hash (P2SH) [3] template. To format transaction T_{offer} (per Section II) as a P2SH, we specify a *redeem script* (written in Script) whose condition $\mathcal{C}$ must be met to fulfill the transaction. This redeem script is hashed and stored in transaction T_{offer}. To transfer funds out of T_{offer}, a transaction T_{fulfill} is constructed. T_{fulfill} includes (1) the redeem script and (2) a set of input values that the redeem script is run against. To programmatically validate that T_{fulfill} can fulfill T_{offer}, the redeem script T_{fulfill} is hashed, and the resulting hash value is compared to the hash value stored in T_{offer}. If these match, the redeem script is run against the input values in T_{fulfill}. T_{fulfill} fulfills T_{offer} if the redeem script outputs true. All our redeem scripts include a time-locked refund condition, that allows the party offering T_{offer} to reclaim the funds after a time window expires. To do so, the party signs and posts a *refund transaction* T_{refund} that points to T_{offer} and reclaims the funds locked in T_{offer}. We reproduce our scripts in the full version [20].

B. Off-Blockchain Performance Evaluation

We evaluate the performance for a run of our protocols between one payer Alice $\mathcal{A}$, one payee Bob $\mathcal{B}$ and the Tumbler $\mathcal{T}$. We used several machines: an EC2 t2.medium instance in Tokyo (2 Cores at 2.50 GHz, 4 GB of RAM), a MacBook Pro in Boston (2.8 GHz processor, 16 GB RAM), and Digital Ocean nodes in New York, Toronto and Frankfurt (1 Core at 2.40 GHz and 512 MB RAM).

Puzzle-solver protocol (Table II). The total network bandwidth consumed by our protocol was 269 Kb, which is roughly $1/8$th the size of the "average webpage" per [45] (2212 Kb). Next, we test the total (off-blockchain) computation time for our puzzle-solver protocol (Section V-A) by running both parties ($\mathcal{A}$ and $\mathcal{T}$) on the Boston machine. We test the impact of network latency by running $\mathcal{A}$ in Boston and $\mathcal{T}$ in Tokyo, and then with $\mathcal{T}$ in New York. (The average Boston-to-Tokyo Round Trip Times (RTT) was 187 ms and the Boston-to-New York RTT was 9 ms.) From Table II, we see the protocol completes in < 4 seconds, with running time dominated by network latency. Indeed, even when $\mathcal{A}$ and $\mathcal{T}$ are very far apart, our 2048-bit RSA puzzle solving protocol is still faster than [30]'s 16x16 Sudoku puzzle solving protocol, which takes 20 seconds.

TumbleBit as a classic tumbler (Table II). Next, we consider classic Tumbler mode (Section IV). We consider a scenario where $\mathcal{A}$ and $\mathcal{B}$ use the same machine, because Alice $\mathcal{A}$ wants anonymize her bitcoin by transferring it to a fresh ephemeral bitcoin address that she controls. Thus, we run (1) $\mathcal{A}$ and $\mathcal{B}$ in Boston and $\mathcal{T}$ in Tokyo, and (2) $\mathcal{A}$ and $\mathcal{B}$ in Boston and $\mathcal{T}$ in New York. To prevent the Tumbler $\mathcal{T}$ for linking $\mathcal{A}$ and $\mathcal{B}$ via their IP address, we also tested with (a) $\mathcal{B}$ connecting to $\mathcal{T}$ over Tor, and (b) both $\mathcal{A}$ and $\mathcal{B}$ connected through Tor. Per Table II, running time is bound by network latency, but is < 11 seconds even with when both parties connect to Tokyo over Tor. Connecting to New York (in clear) results in ≈ 1 second running time. Compute time is only 0.6 seconds, again measured by running $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{T}$ on the Boston machine. Thus, TumbleBit's performance, as a classic Tumbler, is bound by the time it takes to confirm 2 blocks on the blockchain (≈ 20 minutes).

Performance of TumbleBit's Phases. (Table III) Next, we break out the performance of each of TumbleBit's phases when $Q = 1$. We start by measuring compute time by running all $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{T}$ on the Boston machine. Then, we locate each party on different machines. We first set A in Toronto, B in Boston and $\mathcal{T}$ in New York and get RTTs to be 22 ms from Boston to New York, 23 ms from New York to Toronto, and 55 ms from Toronto to Boston. Then we set A in Frankfurt, B in Boston and $\mathcal{T}$ in Tokyo and get RTTs to be 106 ms from Boston to Frankfurt, 240 ms from Frankfurt to Tokyo, and 197 ms from Tokyo to Boston. An off-blockchain payment in the Payment Phase completes in under 5 seconds and most of the running time is due to network latency.

C. Blockchain Tests

Our on-blockchain tests use TumbleBit as a classic tumbler, where payers pay themselves into a fresh ephemeral Bitcoin address. All transactions are visible on the blockchain. Transaction IDs (TXIDs) are hyperlinked below. The denomination of each TumbleBit payment (*i.e.,* the price of puzzle solution) was 0.0000769 BTC (roughly $0.04 USD on 8/15/2016). Table IV details the size and fees[8]used for each transaction.

Test where everyone behaves. In our first test, all parties completed the protocol without aborting. We tumbled 800 payments between $\aleph = 800$ payers and $\aleph = 800$ payees, resulting in 3200 transactions posted to the blockchain and a k-anonymity of $k = 800$.The puzzle-promise escrow transactions $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ are all funded from this TXID and the puzzler-solver escrow transactions $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ are all funded from this TXID.

[8]We use a lower transaction fee rate of 15 Satoshi/byte (see Table IV) for T_{puzzle} and T_{solve} because we are in less of hurry to have them confirmed. Specifically, if $\mathcal{A}$ refuses to sign $T_{\text{cash}(\mathcal{A},\mathcal{T})}$, then $\mathcal{T}$ ends the Payment Phase with $\mathcal{A}$ early (even before the Cash-Out Phase begins), and immediately posts T_{puzzle} and then T_{solve} to the blockchain. See Section V-C.

TABLE II. AVERAGE PERFORMANCE OF RSA-PUZZLE-SOLVER AND CLASSIC TUMBLER, IN SECONDS. (100 TRIALS).

	Compute Time	Running Time (Boston-New York)	RTT (Boston-New York)	Running Time (Boston-Tokyo)	RTT (Boston-Tokyo)	Bandwidth
RSA-puzzle-solving protocol	0.398	0.846	0.007949	4.18	0.186	269 KB
Classic Tumbler (in clear)	0.614	1.190	0.008036	5.99	0.187	326 KB
Classic Tumbler (B over Tor)	0.614	3.10	0.0875	8.37	0.273	342 KB
Classic Tumbler (Both over Tor)	0.614	6.84	0.0875	10.8	0.273	384 KB

TABLE III. AVERAGE OFF-BLOCKCHAIN RUNNING TIMES OF TUMBLEBIT'S PHASES, IN SECONDS. (100 TRIALS)

	Compute Time	Running Time (Boston-New York-Toronto)	Running Time (Boston-Frankfurt-Tokyo)
Escrow	0.2052	0.3303	1.5503
Payment	0.3878	1.1352	4.3455
Cash-Out	0.0046	0.0069	0.0068

TABLE IV. TRANSACTION SIZES AND FEES IN OUR TESTS.

Transaction	Size	Satoshi/byte	Fee (in BTC)
T_{escr}	190B	30	0.000057
T_{cash}	447B	30	0.000134
T_{refund} for T_{escr}	373B	30	0.000111
T_{puzzle}	447B	15	0.000067
T_{solve}	907B	15	0.000136
T_{refund} for T_{puzzle}	651B	20	0.000130

This test completed in 23 blocks in total, with Escrow Phase completing in 16 blocks, Payment Phase taking 1 block, and Cash-Out Phase completing in 6 blocks.

We note, however, that our protocol could also have completed much faster, *e.g.,* with 1 block for the Escrow Phase, and 1 block for the Cash Out Phase. A Bitcoin block can typically hold ≈ 5260 of our 2-of-2 escrow transactions T_{escr} and ≈ 2440 of our cash-out transaction T_{cash}. We could increase transaction fees to make sure that our Escrow Phase and Cash-Out phase (each confirming 2×800 transactions) occur within one block. In our tests, we used fairly conservative transaction fees (Table IV). As a classic Tumbler, we therefore expect TumbleBit to have a higher denomination than the 0.0000769 BTC we used for our test. For instance, transaction fees of 60 Satoshi per Byte (0.0007644 BTC/user) are $\approx 1/1000$ of a denomination of 0.5 BTC.

Test with uncooperative behavior. Our second run of only 10 users (5 payers and 5 payees) demonstrates how fair exchange is enforced in the face of uncooperative or malicious parties. Transactions $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ and T_{puzzle} were timelocked for 10 blocks and $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ was timelocked for 15 blocks. All escrow transactions $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ are funded by TXID and all escrow transactions $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ are funded by TXID. Two payer-payee pairs completed the protocol successfully. For the remaining three pairs, some party aborted the protocol:

Case 1: The Tumbler $\mathcal{T}$ (or, equivalently, Alice $\mathcal{A}_1$) refused to cooperate after the Escrow Phase. Alice $\mathcal{A}_1$ reclaims her bitcoins from escrow transaction $T_{\text{escr}(\mathcal{A},\mathcal{T})}$ via a refund transaction after the timelock expires. The Tumbler $\mathcal{T}$ reclaims its bitcoins from his payment channel with Bob $\mathcal{B}_1$ escrow transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ via a refund transaction after the timelock expires.

Case 2: The Tumbler aborts the puzzle-solver protocol by posting the transaction T_{puzzle} but refusing to provide the transaction T_{solve}. No payment completes from $\mathcal{A}_2$ to $\mathcal{B}_2$. Instead, $\mathcal{A}_2$ reclaims her bitcoin from T_{puzzle} via a refund transaction after the timelock in T_{puzzle} expires. Tumbler reclaims its bitcoins from its payment channel with Bob $\mathcal{B}_2$ via a refund transaction after the timelock on the escrow transaction $T_{\text{escr}(\mathcal{T},\mathcal{B})}$ expires.

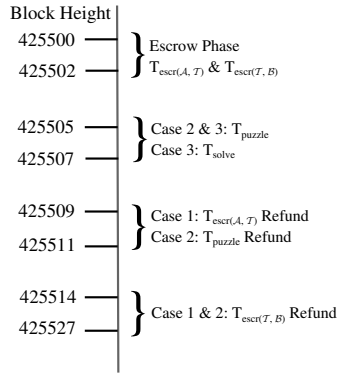

Fig. 5. Timeline of test with uncooperative behavior, showing block height when each transaction was confirmed.

Case 3: The Tumbler provides Alice $\mathcal{A}_3$ the solution to her puzzle in the puzzle-solver protocol, and the Tumbler has an T_{puzzle} signed by $\mathcal{A}$ (Section V-C). However, Alice refuses to sign the cash-out transaction $T_{\text{cash}(\mathcal{A},\mathcal{T})}$ to pay out from her escrow with the Tumbler. Then, the Tumbler signs and posts the transaction T_{puzzle} and its fulfilling transaction T_{solve} and claims its bitcoin. Payment from $\mathcal{A}_3$ to $\mathcal{B}_3$ completes but the Tumbler has to pay more in transaction fees. This is because the Tumbler has to post *both* transactions T_{puzzle} and T_{solve}, rather than just $T_{\text{cash}(\mathcal{A},\mathcal{T})}$; see Table IV.

Remark: Anonymity when parties are uncooperative. Notice that in Case 1 and Case 2, the protocol aborted without completing payment from Alice to Bob. k-anonymity for this TumbleBit run was therefore $k = 3$. By aborting, the Tumbler $\mathcal{T}$ learns that payers $\mathcal{A}_1, \mathcal{A}_2$ were trying to pay payees $\mathcal{B}_1, \mathcal{B}_2$. However, anonymity of $\mathcal{A}_1, \mathcal{A}_2, B_1, B_2$ remains unharmed, since $\mathcal{B}_1$ and $\mathcal{B}_2$ were using ephemeral Bitcoin addresses they now discard to safeguard their anonymity (see Section IV-A).

ACKNOWLEDGEMENTS

We thank Ethan Donowitz for assistance with the preliminary stages of this project, and Nicolas Dorier, Adam Ficsor, Gregory Galperin, Omer Paneth, Dimitris Papadopoulos, Leonid Reyzin, Ann Ming Samborski, Sophia Yakoubov, the anonymous reviewers and many members of the Bitcoin community for useful discussions and suggestions. Foteini Baldimtsi and Alessandra Scafuro performed this work while at Boston University. This work was supported by NSF grants 1012910, 1414119, and 1350733.

REFERENCES

[1] Bitcoin Fog. *Wikipedia*, 2016.

[2] Monero, https://getmonero.org/home. 2016.

[3] Gavin Andresen. BIP-0016: Pay to Script Hash. *Bitcoin Improvement Proposals*, 2014.

[4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *IEEE S&P*, pages 443–458, 2014.

[5] A Back, G Maxwell, M Corallo, M Friedenbach, and L Dashjr. Enabling blockchain innovations with pegged sidechains. *Blockstream, https://blockstream.com/sidechains.pdf*, 2014.

[6] Wacław Banasik, Stefan Dziembowski, and Daniel Malinowski. Efficient Zero-Knowledge Contingent Payments in Cryptocurrencies Without Scripts. Cryptology ePrint Archive, Report 2016/451, 2016.

[7] 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 (SP)*, pages 459–474, 2014.

[8] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of Clients in Bitcoin P2P Network. In *ACM-CCS*, pages 15–29, 2014.

[9] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. Sybil-resistant mixing for bitcoin. In *Workshop on Privacy in the Electronic Society*, pages 149–158, 2014.

[10] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE - SP*, 2015.

[11] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, JoshuaA. Kroll, and EdwardW. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security*, 2014.

[12] David Chaum. Blind signature system. In *CRYPTO*, 1983.

[13] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18. Springer, 2015.

[14] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[15] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. In *ACM SIGKDD*, pages 265–273, 2008.

[16] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*. ACM, 1987.

[17] Grams. Helixlight: Helix made simple. https://grams7enufi7jmdl.onion.to/helix/light.

[18] Matthew Green and Ian Miers. Bolt: Anonymous Payment Channels for Decentralized Currencies. *Cryptology ePrint Archive 2016/701*, 2016.

[19] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *EUROCRYPT*, 1988.

[20] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. Cryptology ePrint Archive 2016/575, 2016.

[21] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions. In *Workshop on Bitcoin and Blockchain Research at Financial Crypto*, February 2016.

[22] Chainalysis Inc. Chainalysis: Blockchain analysis, 2016. https://www.chainalysis.com/.

[23] Tom Elvis Jedusor. Mimblewimble. 2016.

[24] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *ACM-CCS*, 2014.

[25] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *ACM-CCS*, 2015.

[26] Elliptic Enterprises Limited. Elliptic: The global standard for blockchain intelligence, 2016. https://www.elliptic.co/.

[27] Gregory Maxwell. Zero Knowledge Contingent Payment. *Bitcoin Wiki*, 2011.

[28] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. *Bitcoin-talk*, 2013.

[29] Gregory Maxwell. CoinSwap: transaction graph disjoint trustless trading. *Bitcoin-talk*, 2013.

[30] Gregory Maxwell. The first successful Zero-Knowledge Contingent Payment. Bitcoin Core, February 2016.

[31] S Meiklejohn, M Pomarole, G Jordan, K Levchenko, GM Voelker, S Savage, and D McCoy. A fistful of bitcoins: Characterizing payments among men with no names. In *ACM-SIGCOMM Internet Measurement Conference, IMC*, 2013.

[32] Sarah Meiklejohn and Claudio Orlandi. Privacy-Enhancing Overlays in Bitcoin. In *Lecture Notes in Computer Science*, volume 8976. Springer Berlin Heidelberg, 2015.

[33] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Security and Privacy (SP)*, pages 397–411, 2013.

[34] Pedro Moreno-Sanchez, Tim Ruffing, and Aniket Kate. P2P Mixing and Unlinkable P2P Transactions. *Draft*, June 2016.

[35] Malte Möser and Rainer Böhme. Join Me on a Market for Anonymity. *Workshop on Privacy in the Electronic Society*, 2016.

[36] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies*. Princeton University Pres, 2016.

[37] Guevara Noubir and Amirali Sanatinia. Honey onions: Exposing snooping tor hsdir relays. In *DEF CON 24*, 2016.

[38] Henning Pagnia and Felix C. Grtner. On the impossibility of fair exchange without a trusted third party, 1999.

[39] Morgen Peck. DAO May Be Dead After $60 Million Theft. *IEEE Spectrum, Tech Talk Blog*, 17 June 2016.

[40] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft). https://lightning. network, 2015.

[41] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, 2010.

[42] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[43] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *ESORICS*, pages 345–364. Springer, 2014.

[44] Jeff Stone. Evolution Downfall: Insider 'Exit Scam' Blamed For Massive Drug Bazaar's Sudden Disappearance. *international business times*, 2015.

[45] the Internet Archive. Http Archive: Trends, 2015. http://httparchive.org/trends.php.

[46] Peter Todd. BIP-0065: OP CHECKLOCKTIMEVERIFY. *Bitcoin Improvement Proposal*, 2014.

[47] F. Tschorsch and B. Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys Tutorials*, PP(99), 2016.

[48] Luke Valenta and Brendan Rowan. Blindcoin: Blinded, accountable mixes for bitcoin. In *FC*, 2015.

[49] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. Coinparty: Secure multi-party mixing of bitcoins. In *CODASPY*, 2015.