PERUN

Non-technical summary

version 1.0
July 5, 2017
URL: perun.network
email: info@perun.network

1 Introduction

Cryptocurrencies such as Bitcoin have gained great popularity over the last 10 years. The backbone of these currencies is a technology called the *blockchain* (or *ledger*). Since their introduction in [9], several different cryptocurrencies have been proposed. One particular interesting example is the cryptocurrency Ethereum [11], which permits its users to create so-called *smart contracts*. Informally speaking these contracts are agreements written on the ledger and executed by the underlying blockchain protocol itself (see, e.g., [3] for more).

The "blockchain technology" requires each transaction to be stored on the ledger. This imposes a fundamental limit on how many transactions can be processed per second. This scalability problem is drastically amplified with the emergence of nanotransactions that allow users to transfer tiny amounts of money, typically less than 1 cent, and can enable many novel business models, e.g., fair sharing of WiFi connection, or devices paying to each other in the "Internet of Things". Besides the scalability, there are also several other challenges that need to be addressed by ledger-based cryptocurrencies before they can handle massive volumes of nanotransactions. First, in many settings nanotransactions have to be executed instantaneously, which is a problem, since confirmation of transactions in the ledger-based currencies takes some time. Secondly, and more importantly, posting transactions on the ledger usually costs money.

An exciting proposal to address the above challenges is a technology called *payment channels* [4], which allows two users to rapidly exchange money between each other without sending transactions to the ledger. Examples of such systems include *Lightning* and *Raiden*, where the payment channels can be linked to create payment *networks*. This is done via a mechanism of "routing payments" over chains of multiple channels using so-called "hash-locked transactions". They can also be generalized to *state* channels [5, 1], which, informally speaking, are channels that can serve for executing smart contracts between the users that established them.

2 Perun: virtual channels over Ethereum

In this note we informally introduce *Perun*[1], a new system for payment and state channels over Ethereum (for a more complete technical description see Perun's whitepaper [6]). Perun offers a new technique for connecting channels that has several advantages over the existing technique of "routing transactions" over multiple channels. Most importantly, it does *not* require involvement of the intermediary for every payment that is performed between the end users of the channel. This is achieved by constructing a new primitive called *virtual payment channels* over so-called *multistate channels*. We explain these notions below (for the sake of simplicity we ignore several practical issues such as, e.g., the transaction fees). We start with recalling the concept of simple payment channels (which in Perun are called the "basic payment channels").

2.1 Basic channels

A basic channel between two parties, Alice and Bob, allows them to keep massive bulk of transactions "off-chain". More precisely: the ledger is used only when parties involved in the payment channel disagree, or when they want to close the channel. As long as the parties are not in conflict, they can

[1] Perun is the god of thunder and lightning in the Slavic mythology. This choice of name reflects the fact that one of our main inspirations is the Lightning system. The name "Perun" also connotes with "peer" (which reflects its peer-to-peer nature), and "run" (which stresses the fact that the system is very fast).

freely *update* the balance of the channel (i.e. transfer money between each other's accounts).

Because off-chain transactions can always be settled on a ledger by the users involved, there is no incentive for the users to disagree, and hence honest behavior will be strongly preferred by them. In the optimistic case, when the two parties involved in the payment channel play honestly, and off-chain transactions never hit the ledger before the channel is closed, payment channels significantly reduce transaction fees, allow for instantaneous payments and limit the load put on the ledger.

Technically, the channels are implemented using a smart contract technology (that is used to guarantee that in case of dispute their channel can always be settled in a fair way). Below we describe only the functionality that these channels provide. For the implementation details see, e.g., [10] or [6].

Basic payment channels. At a high-level a basic payment channel between two parties Alice and Bob starts with a *creation procedure*, where Alice puts x_A coins into the channel and Bob commits x_B coins respectively. Initially, the *balance* of the channel can be described by a pair (x_A, x_B), meaning that Alice "has x_A coins in it" and Bob "has x_B coins in it" (x_A and x_B are also respectively called Alice's and Bob's *cash* in the channel). Pictorially, creating a channel in which Alice deposits 2 coins and Bob deposits 3 coins can be represented as follows:

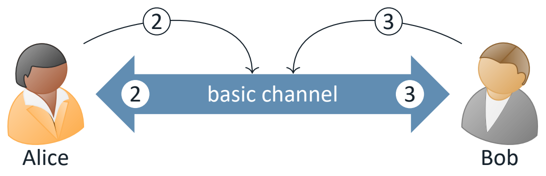

Fig. 1

After this set-up has been completed, Alice and Bob can *update* the distribution of funds in the channel without interacting with the blockchain. For example, if Alice transfers 1 coin to Bob in the channel from Fig. 1, then, after the update the balance of this channel will be $(1, 4)$, which can be depicted as follows:

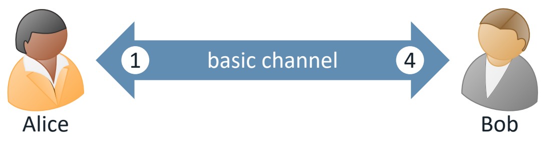

Fig. 2

The balance of a channel can be updated multiple times, subject to the invariant that the sum of Alice's and Bob's cash in the channel does not change. At some point one of the parties that created the channel can decide to *close* it. For example, if Alice wants to close out the channel, she commits the current balance (x'_A, x'_B) of the channel to the blockchain and the funds are distributed accordingly to Alice and Bob (i.e. Alice and Bob receive x'_A and x'_B coins respectively). For instance, if the channel from Fig. 2 is closed, then Alice and Bob receive 1 and 4 coins, respectively, which can be depicted as follows.

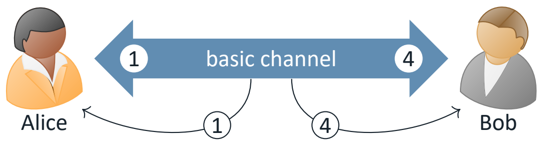

Fig. 3

Basic multistate channels and nanocontracts. As mentioned before, a further generalization of payment channels are the *state channels*, which significantly enrich the functionality of payment channels. State channels can, besides payments, execute smart contracts "inside of a channel" in an off-chain way. Very informally, this is done by letting the channel's state contain, in addition to the financial balance, a string σ that describes the current state of the contract's *storage* (i.e. the values of all the contract's variables), and a value x that indicates the amount of coins "blocked" in the contract. As long as there is no conflict between the users of the state channel, they can freely update σ. However, once one of the parties starts to misbehave, the other one can post the latest version of σ on the ledger, and the ledger will finish the execution of the contract (and take care of all its financial consequences), starting from storage σ. In most of the cases, the parties will not need to do it, and all the execution can be handled by simply updating the state σ in a "peaceful way".

One way to look at it is that the state channels provide a way to implement a "virtual 2-party ledger", in the following sense: two parties that established a state channel can maintain a "simulated contract ledger" between themselves and perform the ledger transactions on it without registering them on the *real* ledger (as the parties do not enter into a conflict). Security of this solution comes from the fact that at any time any party can pass the current state of the channel to the real ledger.

In Perun we extend this concept even further, by defining a notion of "multistate channels". The most important feature of a multistate channel is that it consists of several "independent" contract storage states $\sigma_1, \ldots, \sigma_m$ (in contrast with a *single* storage state σ, in the standard state channels). The σ_i's correspond to contracts that can be created, updated, and executed in parallel (we call such contracts the "nanocontracts"). Every σ_i has also an associated variable x_i that describes the amount of coins blocked in the corresponding nanocontract. Each time a new nanocontract is created, Alice and Bob have to block some of their coins in it. Hence, the sum of Alice's and Bob's cash in the channel can change over the lifetime of the channel.

To readers familiar with state channels it may look like there is not that much difference between multistate channels and standard state channels, as in principle we can simply view $(\sigma_1, \ldots, \sigma_m)$ as a *single* storage of a contract that operates on every σ_i independently. In [6] we explain why this is not the case (the main reason are the subtleties connected with parallel execution and update of different nanocontracts). The multistate channels are a crucial building block for the virtual channels that we we describe below.

2.2 Virtual channels

The main novelty of Perun is a new method for connecting basic channels that is alternative to "payment routing" used in existing payment channel networks. Namely, Perun offers the so-called "virtual channels" that minimize the need for interaction with the intermediaries in the channel chains, and in particular do not require to route individual payments over them. The construction of virtual channels is based on the idea of applying the channel technique recursively, by building a payment channel "on top of" the multistate channels. Our observation is that contracts in a state channel can be used to construct payment channels in a way similar to how smart contracts on the ledger can be used to build payment channels over the standard ledger.

There are many details that need to be taken care of in order for this general idea to work. What is especially delicate is handling *parallelism*, i.e., ensuring that several virtual channels (with overlapping sets of users) can be opened, updated, and closed simultaneously. This issues are described in detail in [6]. Below we provide only a high-level introduction to this topic.

Virtual payment channels of length 2. Consider 3 parties: Alice, Bob and Ingrid, and suppose that there exist a basic multistate channels X and Y between Alice and Ingrid and between Ingrid and Bob, respectively. Let (x_A, x_I) and (y_I, y_B) be the respective balances of these channels. The figure below illustrates this situation with $(x_A, x_I) = (3, 3)$ and $(y_I, y_B) = (4, 5)$

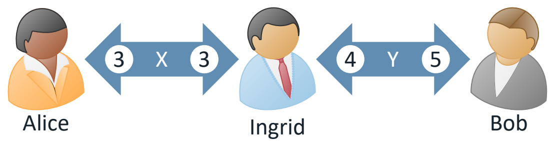

Fig. 4

In Perun, Alice and Bob can establish a payment channel with the help of Ingrid, but *without* touching the ledger. To distinguish such channels from the basic ones (that appear on the ledger) we call them "virtual". In case of virtual channels, as long as everybody is honest, Alice and Bob need to interact with Ingrid only when the channel is created and when it is closed. Each individual transaction between Alice and Bob that goes via this channel does *not* require interacting with Ingrid. Our scheme is secure against arbitrary corruptions of Alice, Ingrid, and Bob (in particular: no assumption about the honesty of the intermediary Ingrid is needed). Pictorially, a creation of a virtual channel in which Alice deposits 2 coins from channel X and Bob deposits 3 coins from channel Y can be represented as follows:

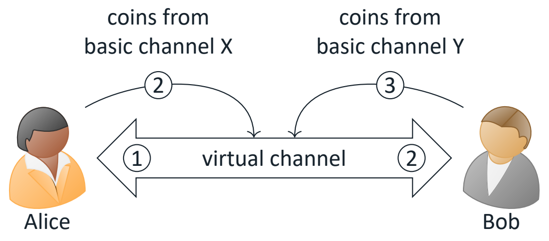

Fig. 5

In general, let us consider that Alice deposits v_A coins (from channel X) in the virtual channel, and Bob deposits v_B coins (from channel Y) in it. At a technical level, this is handled in the following way: (a) Alice and Ingrid create a nanocontract in channel X in which Alice blocks v_A of her coins, and Ingrid blocks v_B of her coins, and (b) Ingrid and Bob create a nanocontract in channel Y in which Ingrid blocks v_A of her coins, and Bob blocks v_B of his coins. As a result, the cash balance of channel X is now $(x_A - v_A, x_I - v_B)$, and the balance of channel Y is $(y_I - v_A, y_I - v_B)$. A creation of virtual channel from Fig. 5 over virtual channels X and Y from Fig. 4 is depicted below.

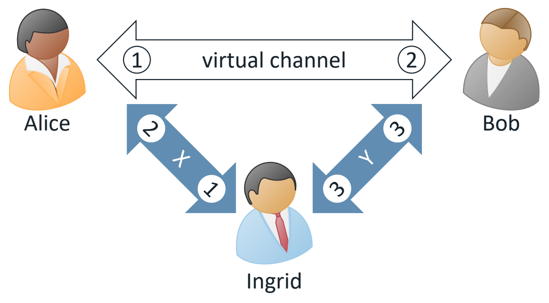

Fig. 6

Note that as a result of the virtual channel creation procedure Ingrid has to block her money in the basic channels, which can be viewed as a disadvantage. In Sect. 2.4 below we discuss a solution for this problem.

Once a virtual channel is created, it can be updated multiple times, exactly in the same way, as the basic channel. For example, the balance of a channel from Fig. 5 can be changed to $(2, 1)$:

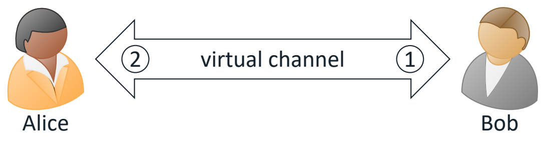

Fig. 7

A virtual channel is closed in a similar way as the basic channel (in the current technical write-up [6] it is assumed that a virtual channel is closed when some time called "validity" comes), but in reality the closing conditions can be much more general). The main difference compared to closing a basic channel is that the "financial consequences" of channel closing are visible on the basic channels X and Y, and not on the ledger. For example, closing the virutal channel from Fig. 7 can be illustrated as follows:

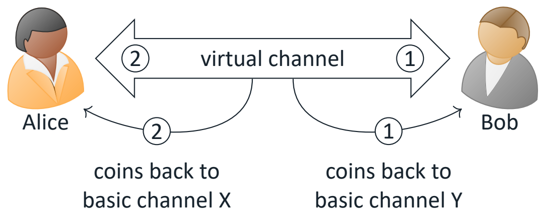

Fig. 8

More concretely, if the last balance of the virtual channel is (v'_A, v'_B) and the current balances of X and Y are (x'_A, x'_I), and (y'_I, y'_B) (respectively) then as a result of channel closing, the balances of X and Y become $(x'_A + v'_A, x'_I + v'_B)$, and $(y'_I + v'_A, y'_I + v'_B)$ (and the nanocontracts that "handles" the virtual channel in basic channels X and Y disappears). For example, the result of closing the virtual channel from Fig. 7 can be represented as follows:

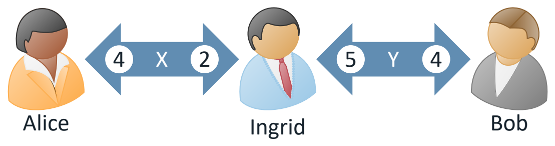

Fig. 9

Note that the consequences of creating and closing a channel are always "neutral" for Ingrid, i.e., if she looses money in one basic channel, then she gains the same amount in the other channel (cf. Figures 4 and 9).

2.3 Longer virtual multistate channels

One natural question is if one can have virtual channels that are (a) longer, (b) have states. It turns out that this is possible, and in fact these two questions are closely related. More precisely: if one constructs virtual *multistate* channels, then one can apply our idea recursively for an arbitrary number of times. We are currently in the process of formalizing these ideas in a subsequent paper.

2.4 Further extensions

One drawback of the construction outlined above is the fact that the parties needs to block the coins that are used for constructing the channel. This might be especially problematic for the intermediaries (since they have to block coins even though they are not going to perform any real payments with them). Of course in reality the intermediaries will be charging fees for their service, and hence one answer to this concern is that simply the costs of having money blocked will ultimately be charged on the end users of the channel, and will probably not be high, since they will correspond to the cost of risk-free credit granted for the time of the duration of the virtual channel. Another option is to frequently create virtual channels for payments of relatively small value, instead of creating one virtual channel of a larger value. Since creating a virtual channel involves interaction with the intermediary, this solution exhibits a trade-off between how much interaction is needed vs how much money needs to be blocked (observe that "routing payments" approach is an extreme case of this trade-off with lots of interaction, and little money being blocked). In Sect. 4 we show an example of an application that uses this approach.

The third option for addressing this problem is to slightly relax the security guarantees, and to use the notion of a reputation (in a very mild way). Namely, we can replace the full cheating-resilience by a weaker "cheating-*evidence*". More precisely, the security guarantee in this case would be: "if an intermediary cheats then you can either get your money back, or you can post an evidence of cheating on the blockchain". Hence the risk that one gets cheated by an intermediary that has been functioning for a long time already is low, and probably acceptable in practice in case of nanotransactions.

Another obvious problem is the need for permanent online availability by the parties, since they need to constantly monitor the network to see if the other party did not submit and old version of the channels. Again, this problem appears also in other payment networks, and solutions for this exists (e.g. network monitoring can be outsourced) [10]. We will provide an extension of Perun that includes such monitoring in subsequent work. Another possible extension is adding anonymity to Perun (in the style of [7]).

3 State of the project

The description of a protocol for multistate channels and virtual payment channels of length 2 appears in [6]. The prototype academic implementation is being developed on GitHub `lethdev.github.io/Perun/` (at the time of writing this text, this site contains the contract code, and part of the code for the user algorithms). The updates about the project are posted on the `perun.network` webpage. We are currently in the process of extending the formal description of Perun to cover longer virtual multistate channels (this will be accompanied with a formal security model and proofs), as well as other extensions described above.

4 Possible applications

In this section we briefly describe possible applications of Perun.

4.1 Selling nanoservices

Perun can be used, e.g., when two parties that do not have any trust relationship between each other want to do perform an electronic transaction. This occurs, for example, in applications like file-sharing and distributed storage (think of "BitTorrent with financial incentives to share data"), or incentive-driven data routing in networks. In such scenarios fair exchange (i.e. deciding what should happen first: the payment or the service delivery) is problematic.

More concretely, consider a situation when a Buyer pays to a Seller for delivering a service that in total is worth 1 coin, but the Buyer is not willing to pay upfront the whole sum (since he does not trust the Seller), and vice-versa: the Seller does not want to deliver the service without being payed first (for the same reason). The parties therefore break the transaction into 100 *nanoservices*, and each of them is sold for a nanopayment of 0.01 coins. For example, a nanoservice can be a packet of data delivered wirelessly from the Internet, or a small chunk of data in BitTorrent. The parties expect that, if the everything goes ok, then in total 100 of nanoservices will be performed.

Suppose the Buyer and the Seller have basic channels with an intermediary called "Payment Hub". The "payments routing" approach puts some inherent limitations on the "size" of each nanoservice, as each payment requires interaction with the Hub, which introduces delays and puts heavy communication load on it. By using Perun, the Hub is involved in the communication between the Seller and the Buyer in a much more limited way.

Typically, since the parties will not want to block too large amounts of coins in the basic channels for the virtual channel creation (see discussion in Sect. 2.4), this will be done by sequentially creating several virtual channels of a smaller value. More concretely, the virtual channel will be created for a small deposit, 0.1 coins, say, from the Buyer (the Seller puts no coins into the channel, since he will be only receiving payments from it). After each nanoservice delivery, the Buyer will transfer to the Seller a nanopayment (of a value 0.01 coins). If the payment is not delivered, then the Seller quits the protocol. Hence, the maximal risk for the Seller's perspective is that he will not be payed for a single nanoservice (and for the Buyer the protocol execution is risk-free).

The above exchange will be performed 10 times, after the channels capacity is reached. Call an execution of such a sequence of nanopayments a "microtransaction". A microtransaction for 0.1 coin broken into 10 nanotransaction (worth 0.01 coin each) is depicted on Fig. 10 below.

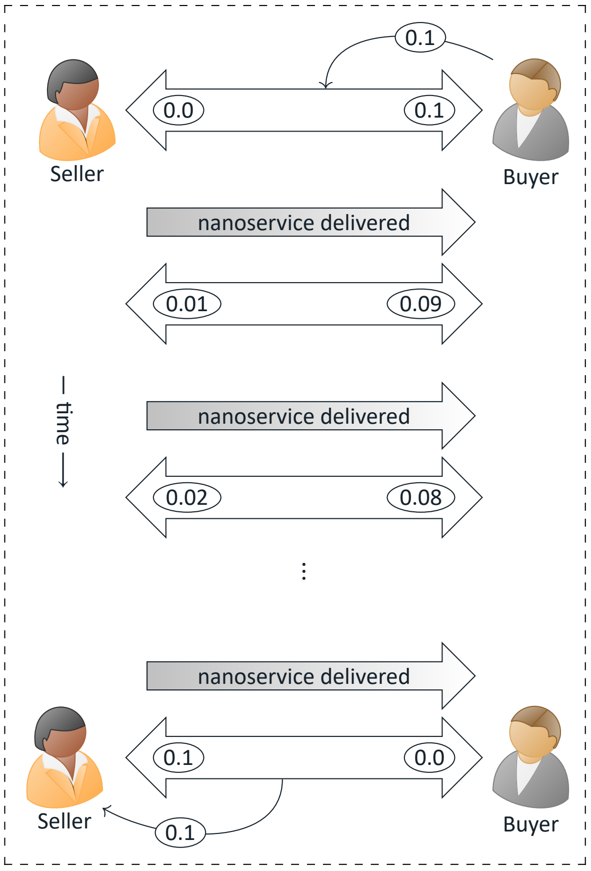

Fig. 10

The entire interaction between the Seller and the Buyer consist of 10 microtransactions executed one after another. This is depicted on Fig. 11.

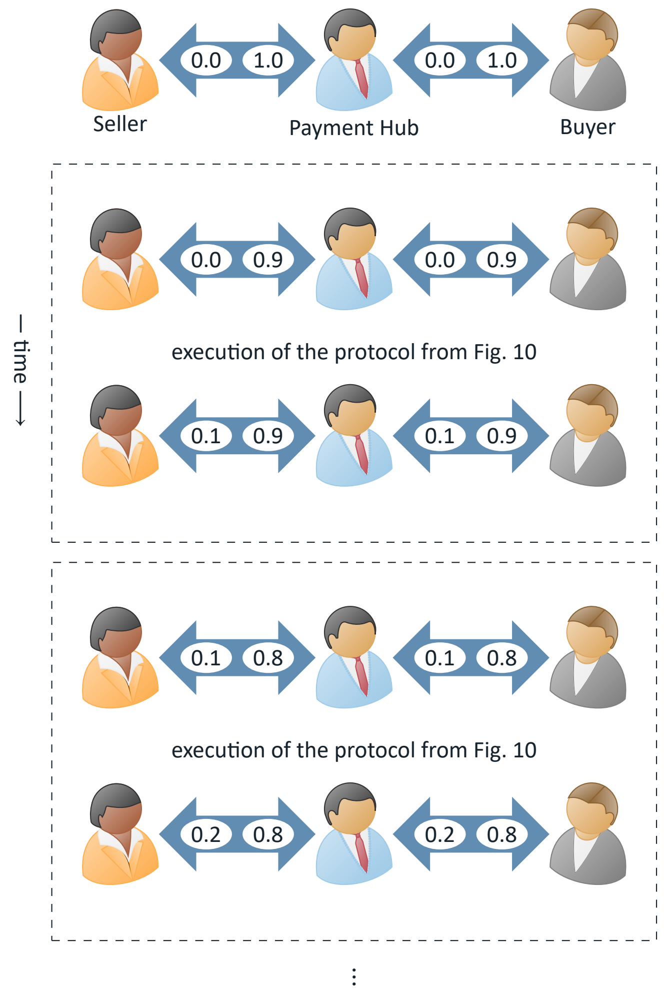

Fig. 11

4.2 Internet of Things

Another natural application of Perun is the Internet of Things (IoT). Due to the cost pressure to reduce the power consumption in many situations the IoT devices will be connected via some short-range communication technology (like Bluetooth or NFC), and will minimize the interaction with remote devices. Hence, routing each payment via a third party server may not be an option in such situations. Our technique removes the need for such interaction.

Another, related scenario where Perun can be applied is when the payment intermediary cannot be assumed to be always available. For example imagine payments in the vehicular ad hoc networks — here the permanent availability of the internet connections cannot be guaranteed (due to conditions like entering a tunnel, or a zone with no mobile phone network access).

4.3 Fair games

Virtual multistate channels can also be used to play fair games (see, e.g., [2, 8]) without interacting with a blockchain. Note that the use of virtual state channels has an additional advantage of protecting contract's privacy (as long as the participants are honest, contract's history remains secret).

References

[1] I. Allison. *Ethereum's Vitalik Buterin explains how state channels address privacy and scalability*. `https://tinyurl.com/n6pggct`. 2016.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. 'Secure Multiparty Computations on Bitcoin'. In: *2014 IEEE Symposium on Security and Privacy*. Berkeley, California, USA: IEEE Computer Society Press, 2014, pp. 443–458.

[3] *Bitcoin Wiki: Contract*. `https://en.bitcoin.it/wiki/Contract`. 2016.

[4] *Bitcoin Wiki: Payment Channels*. `https://en.bitcoin.it/wiki/Payment_channels`. 2016.

[5] J. Coleman. *State Channels*. `http://www.jeffcoleman.ca/state-channels/`. 2015.

[6] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. *PERUN: Virtual Payment Channels over Cryptographic Currencies*. Cryptology ePrint Archive, Report 2017/635. `http://eprint.iacr.org/2017/635`. 2017.

[7] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. *TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub*. Cryptology ePrint Archive, Report 2016/575. `http://eprint.iacr.org/2016/575`. 2016.

[8] R. Kumaresan, T. Moran, and I. Bentov. 'How to Use Bitcoin to Play Decentralized Poker'. In: *ACM CCS 15: 22nd Conference on Computer and Communications Security*. Ed. by I. Ray, N. Li, and C. Kruegel: Denver, CO, USA: ACM Press, 2015, pp. 195–206.

[9] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. `http://bitcoin.org/bitcoin.pdf`. 2009.

[10] J. Poon and T. Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Draft version 0.5.-9.2, available at `https://lightning.network/lightning-network-paper.pdf`. 2016.

[11] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. `http://gavwood.com/paper.pdf`. 2016.