Lower Bounds for Off-Chain Protocols: Exploring the Limits of Plasma

Stefan Dziembowski
University of Warsaw, Poland
Stefan.Dziembowski@crypto.edu.pl

Grzegorz Fabiański
University of Warsaw, Poland
Grzegorz.Fabianski@crypto.edu.pl

Sebastian Faust
Technische Universität Darmstadt, Germany
sebastian.faust@tu-darmstadt.de

Siavash Riahi
Technische Universität Darmstadt, Germany
siavash.riahi@tu-darmstadt.de

Abstract

Blockchain is a disruptive new technology introduced around a decade ago. It can be viewed as a method for recording timestamped transactions in a public database. Most of blockchain protocols do not scale well, i.e., they cannot process quickly large amounts of transactions. A natural idea to deal with this problem is to use the blockchain only as a timestamping service, i.e., to hash several transactions $tx_1, \ldots, tx_m$ into one short string, and just put this string on the blockchain, while at the same time posting the hashed transactions $tx_1, \ldots, tx_m$ to some public place on the Internet ("off-chain"). In this way the transactions tx_i remain timestamped, but the amount of data put on the blockchain is greatly reduced. This idea was introduced in 2017 under the name *Plasma* by Poon and Buterin. Shortly after this proposal, several variants of Plasma have been proposed. They are typically built on top of the Ethereum blockchain, as they strongly rely on so-called *smart contracts* (in order to resolve disputes between the users if some of them start cheating). Plasmas are an example of so-called *off-chain protocols*.

In this work we initiate the study of the inherent limitations of Plasma protocols. More concretely, we show that in every Plasma system the adversary can either (a) force the honest parties to communicate a lot with the blockchain, even though they did not intend to (this is traditionally called *mass exit*); or (b) an honest party that wants to leave the system needs to quickly communicate large amounts of data to the blockchain. What makes these attacks particularly hard to handle in real life is that these attacks do not have so-called *uniquely attributable faults*, i.e. the smart contract cannot determine which party is malicious, and hence cannot force it to pay the fees for the blockchain interaction. An important implication of our result is that the benefits of two of the most prominent Plasma types, called *Plasma Cash* and *Fungible Plasma*, cannot be achieved simultaneously.

Besides of the direct implications on real-life cryptocurrency research, we believe that this work may open up a new line of theoretical research, as, up to our knowledge, this is the first work that provides an impossibility result in the area of off-chain protocols.

2012 ACM Subject Classification Security and privacy; Security and privacy → Cryptography

Keywords and phrases blockchain, lower bounds, off-chain protocol, commit chain, plasma

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.72

Related Version A full version of the paper is available at `https://eprint.iacr.org/2020/175`.

Funding This work was partly supported by the FY18-0023 PERUN grant from the Ethereum Foundation, by the TEAM/2016-1/4 grant from the Foundation for Polish Science, by the DFG CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research (BMBF)

iBlockchain project (grant nr. 16KIS0902), and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

1 Introduction

What does it mean to *timestamp* a digital document? Haber and Stornetta in their seminal paper [17] define timestamping as a method to certify when a given document was created. In many settings the timestamped document T remains secret after it was timestamped, until its creator decides to make it public. This is often because of efficiency reasons – for example in the scheme of [17] what is really timestamped is the cryptographic hash[1] $H(T)$ (not T), which leads to savings in communication between T's creator and the timestamping service. Sometimes the secrecy of T is actually a desired feature. According to [29] several researchers in the past (including Galileo Galilei and Isaac Newton) have used ad-hoc methods to timestamp their research ideas before publishing them, in order to later claim priority.

Recently, in the context of blockchain, timestamping has been used in a slightly different way. Namely, in the paper that introduced this technology [25], the "timestamping" mechanism is such that T's creator does not only get a proof that T has been created at a given time, but also that it has been *made public* at this time. Let us call this kind of scheme *public timestamping*, and let us refer to the former type of timestamping as *secret* timestamping. The public timestamping feature of blockchain has been one of the main reasons why this technology attracted so much attention. In fact one of the first projects that use blockchain for purposes other than purely financial was *Namecoin* that used the timestamping to create a decentralized domain name system (see, e.g., [19] for more on this project).

In many cases timestamping is expensive, and its costs grow linearly with the length of the timestamped document. This is especially true for blockchain-based solutions, where all parties in the network need to reach *consensus* about what document was published and when. For example, Bitcoin (the blockchain system introduced in [25]) can process at most around 1MB of data per 10 minutes. In Ethereum, which is another very popular blockchain system [31] "timestamping" a word of 32 bytes cost currently around USD 0.80. A similar problem appears in several other blockchain protocols. We give a short introduction to blockchain in Sec. 1.1. For a moment let us just say that typical blockchains come with their own virtual currencies (also called *cryptocurrencies*). In the most standard case the "timestamped" messages define financial transfers between the network participants, and are hence called *transactions*. We will refer to timestamping a transaction as *posting it on the blockchain*.

To summarize, from the efficiency point of view the secret timestamping is better than the public one (as only hashes need to be timestamped). From the security perspective these two types of timestamping are incomparable as the fact that the timestamped document T needs to be published can be considered to be both an advantage and a disadvantage, depending on the particular application. For example, one could argue that considering timestamped hashes of academic papers as being sufficient evidence for claiming priority would slow down scientific progress, as it would disincentive making the papers public (until, say, the author writes down all followup papers that build upon the timestamped paper).

[1] In this paper we assume reader's familiarity with basic cryptographic notions such as hash functions, negligible functions, etc. For an introduction to this topic see, e.g., [20].

A natural question therefore is as follows. Suppose we have access to a timestamping service that is expensive to use (so we would rather timestamp only very short messages there). Can we use it to "emulate" public timestamping that would be cheap for long documents T? Of course this "emulation" cannot work in general, since some scenarios may simply require a proof that the whole T was available publicly at a given time. So the best we can hope for is to find *some* applications which permit such emulation. An idea for such emulation emerged recently within the cryptocurrency community under the names *Plasma* or *commit chains*[2] [26, 21, 16, 28, 24]. Very informally speaking, Plasma allows to "compress" a number of blockchain transactions $tx_1, \ldots, tx_m$ into one very short string $h = H(T)$, where $T := (tx_1, \ldots, tx_m)$. The transactions can from from different parties called *users*. The only document that gets posted on the blockchain is h. The compression and posting h is done by one designated party called the *operator* (denoted Op). Besides of posting h, the operator publishes all the transactions T on some public network (say: on her web-page). Since this publication is *not* done on the blockchain we also say that it is performed *off-chain*.

Publishing T off-chain is important since only then each user can verify that her transaction was indeed included into the hashed value. Moreover, typical Plasma designs use as H the so-called *Merkle-tree hashing* with $tx_1, \ldots, tx_m$ being the labels of the leaves (see the full version of this paper [11] for more on this technique). Thanks to this, every user can prove that his transaction was included in $H(T)$ with a proof of length $O(\log n)$. If the operator does not include some tx_i in T then the user U that produced tx_i can always post tx_i directly to the blockchain. In this case using Plasma does not bring any benefits to U compared to just using the blockchain directly from the beginning. However, in reality it is expected that this is not going to happen often, especially since the operators are envisioned to be commercial entities that will charge some fee for their services.

What is much more problematic is the case when the malicious operator does *not* publish T off-chain. This situation is typically referred to as *data unavailability*. Note that data unavailability is subjective, i.e., the parties can have different views on whether it happened or not. This is because, unlike the situation on the blockchain, there may be no consensus on what was published off-chain. Moreover, there is no way to produce a proof that data unavailability happens: even if some parties complain on the blockchain that they did not receive T there is no way to determine (just by looking at the blockchain) whether they are right, or if they are just falsely accusing the operator. The situation when a disagreement between parties happens but there is no "blockchain-only" way to verify which party is corrupt, is commonly referred to as a *non-uniquely attributable fault*. Finally, some Plasma protocols require all the honest parties to immediately act on blockchain after a data unavailability attack happened. This is called "mass exit", although (for the reasons explained in Sec. 1.1 in this paper we call it "large forced on-chain action")

Plasma comes in many variants and has been discussed in countless articles (see Sec. 1.1). One of the most fundamental distinction is between the two types of Plasma systems: *Plasma Cash* and *Fungible Plasma*. As we explain in more detail in Sec. 1.1 they both serve for the "emulation" that we outlined above, but have different incomparable features. From one point of view Fungible Plasma is better than Plasma Cash since it is "fungible", which means that the money can be arbitrarily divided and merged. On the other hand: Fungible Plasma suffers from some problems that Plasma Cash does not have, namely the adversary can cause a "non-uniquely attributable large forced mass actions" (we explain these notions Sec. 1.1). The cryptocurrency community has been unsuccessfully looking for a Plasma solution that

[2] In this paper we mostly use the name "Plasma" due to its brevity.

would have the benefits of both Plasma Cash and Fungible Plasma. The main result of this paper is that such Plasma cannot be achieved simultaneously. In other words, we show inherent limitations of the compression technique, at least for compressing blockchain financial transactions. Our paper can also be viewed as initiating the theoretical analysis of lower bounds for the smart contract protocols. We write more about our contribution in Sec. 1.2. First, however, let us provide some more introduction to blockchain and to Plasmas.

1.1 Introduction to Plasma

Let us start with providing some more background on blockchain and smart contracts. Blockchain can be viewed as a public *ledger* containing timestamped transactions that have to satisfy some correctness constraints. Moreover, several blockchains permit to execute the so-called *smart contracts* [27] (or simply: "contracts"), which informally speaking, are "self-executable" agreements described in form of computer programs. Examples of such blockchain platforms include *Ethereum*, *Hyperledger Fabric*, or *Cardano*. Typically, it is assumed that contracts are deterministic and have a public state. Moreover, they can own some coins. Executing a contract is done by posting transactions on the blockchain and it costs fees that depend on the computational complexity of the given operation, and on the amount of data that needs to be transmitted to the contract.

Let us now explain the basic idea of Plasma, and introduce some standard terminology. Since it is an informal presentation, we mix the definition of the protocol with its construction. In the formal sections of the paper these two parts are separated (the definition appears in Sec. 2, and the constructions in the full version of this paper [11]). As highlighted above Plasma address the scalability problem of blockchain by keeping the massive bulk of transactions outside of the blockchain ("off-chain"). The parties that are involved in the protocol rely on a smart contract that is deployed on the ledger of the underlying cryptocurrency, but they try to minimize interacting with it. Typically, this interaction happens only when the parties join and leave the protocol, or when they disagree. Since all parties know that in case of disagreement, disputes can always be settled on the ledger, there is no incentive for the users to disagree, and honest behavior is enforced.

In the optimistic case, when the parties involved in the protocol play honestly, and the off-chain transactions never hit the ledger, these protocols significantly reduce transaction fees and allow for instantaneous executions. Off-chain protocols also resemble an idea explored in cryptography around two decades ago under the name "optimistic protocols" [7, 1]. In this model the parties are given access to a trusted server that is "expensive to use", and hence they do not want to contact it, unless it is absolutely necessary

Plasma's operator *Op* provides a "simulated ledger", in which other parties can deposit their coins, and then perform operations between each other. The key requirement is that its users do not need to trust the operator, and in particular if they discover that she is cheating, then they can safely withdraw their funds. The latter is called an *exit* from the simulated ledger, and requires communication with the underlying ledger.

Plasma protocols come in different variants (see Sec. 1.1), however, they are all based on a single framework proposed in [26]. The parties that execute Plasma are: the *users* $U_1, \ldots, U_n$, and the *operator Op*. Moreover, the parties have access to a contract on the blockchain. In our formal modeling this contract will be represented as a trusted interactive machine Γ with public state, owning some amount of coins. Each user U_i has some number of coins initially deposited in his Plasma account which is maintained by Γ. This number is called a *balance* and is denoted with $b_i \in \mathbb{Z}_{\geq 0}$. Users' balances are changing dynamically during the execution of the protocol. The total number of coins owned by the contract Γ is

equal to the sum of all balances of its users. A vector $\overrightarrow{b} := (b_1, \ldots, b_n)$ is called a *Plasma chain*. When referring to the underlying blockchain (i.e. the one on which Γ is deployed) we use the term *main chain*. Note that the operator Op has no account and only facilitates transfers of the users. In some variants of Plasma (see Sec. 1.1) the operator blocks some amount of coins (called operator's *collateral*) that can be used to compensate the users their losses in case she misbehaves.

Let us briefly describe the different operations that parties of the Plasma protocol can execution during the lifetime of the system. We divide time into *epochs* (e.g. 1 epoch takes 1 hour). In the ith epoch the operator sends some information C_i to Γ. We can think of C_i as "compressed" information about the vector $(b_1, \ldots, b_n)$ containing the users' balances. By "compressed" we mean that $|C_i|$ is much shorter than the description of $(b_1, \ldots, b_n)$, and usually its length is constant in every epoch. We will refer to C_i as a "commitment" to $(b_1, \ldots, b_n)$. The length of C_i is called the *commit size*.

In each epoch every user U_i can request to *exit*, by which we mean that all b_i coins from her Plasma account get converted to the "real" coins on the main chain, and she is no longer a part of this Plasma chain (which, in our formal modeling will be indicated by setting $b_i := \perp$). Plasma's security properties guarantee that every user can exit with all the coins that she currently has in the given Plasma chain. It is often required that exiting can be done cheaply, and in particular that the total length of the messages sent by the exiting user to Γ is small. The amount of data that a user needs to send to Γ in order to exit the Plasma chain is called the *exit size*.

Finally, any two users of the same Plasma chain can make *transfers* between each other. Suppose U_k wants to transfer v coins from her account to U_j. This transfer operation involves only communicating with U_j, and with the operator Op, while no interaction with the contract Γ is needed. Under normal circumstances (i.e. when the operator is honest) the next Plasma block that is committed to the main chain will simply have v coins deduced from U_k's account and v coins added to U_j's account.[3]

1.1.1 Challenges in designing Plasma systems

The main challenge when designing a Plasma system is to guarantee that every user can exit with her money. This is usually achieved as follows: each C_i is a commitment to $(b_1, \ldots, b_n)$, computed using a Merkle tree. An honest operator Op is obliged to obey the following rule:

Explaining commitments – each time Op sends C_i to Γ, she sends the corresponding $\overrightarrow{b} := (b_1, \ldots, b_n)$ to all the users.

Technically, sending $\overrightarrow{b}$ to the users can be realized, e.g., by publishing it on the operator's web-page (i.e.: "off-chain"). Every user U_j can now check if she has the correct amount on her account and if C_i was computed correctly. Moreover, thanks to the properties of Merkle trees, U_j has a short proof of size $O(\log(n))$ that b_j has been "committed" into C_i.

The above description assume that the operator is honest. If she is corrupt things get more complicated. Note that C_i sent by the operator to Γ is publicly known (due to the properties of the underlying blockchain). Hence, we can assume that all the users agree on whether C_i was published and what is its exact value. The situation is different when it comes to the vector $\overrightarrow{b}$ that should be published off-chain. In particular, if $\overrightarrow{b}$ has *not* been published, then the users have no way to prove this to Γ. This is because whether some data

[3] To keep things simple, in this paper we do not discuss things like "transfer receipts", i.e., confirmations for the sender that the coins have been transferred.

has been published off-chain or not does not have a digital evidence that can be interpreted by Γ. This leads us to the following attack that can be carried out by a malicious operator, and is intensively studied by the cryptocurrency community (see, e.g., [3]).

Data unavailability attack – in this attack the corrupt operator publishes C_i but does not publish $\overrightarrow{b}$.

Note that this attack has no extra cost for the operator because from the point of view of the smart contract Γ, the operator behaves honestly, and hence the users cannot complain to Γ and request, e.g., that Op sends $\overrightarrow{b}$ to Γ. Furthermore determining if this attack happened is "subjective", i.e., every user U_j has to detect it herself. Moreover, in case of data unavailability it is impossible for Γ to determine whether U_j or Op is dishonest, since a sheer declaration of U_j that Op did not send him the data obviously cannot serve as a proof that it indeed happened. This leads to the following definition.

Non-uniquely attributable faults – this refers to the situation when the contract has to intervene in the execution of the protocol (because the protocol is under attack), but it unable to determine which party misbehaved (see, e.g., [2]).

Non-uniquely attributable faults appear typically in situations when a party claims that it has not received a message from another party. In contrast if a contract *is* able to determine which party is corrupt then we have a *uniquely*-attributable fault. A typical example of such a fault is when a party signs two contradictory messages. Unfortunately, non-uniquely attributable faults are hard to handle in real life, since it is not clear which party should pay the fees for executing the smart contract, or which party should be punished for misbehaving. In particular, what is unavoidable in such a case is that a malicious party P can force another participant P' to lose money on fees (potentially also loosing money herself). This phenomenon is known as *griefing* [2].

When a user realizes that the operator is dishonest, then she often needs to start quickly interacting with Γ in order to protect her coins. This action has to be done quickly, and has to be performed by each honest user. This leads to the following definition.

Forced on-chain action of size α – this term refers to the situation that honest parties who did not intend to perform an exit are forced by the adversary to quickly interact with Γ, and the total length of the messages sent by them to Γ is α. Informally, when α is large (e.g. $\alpha = \Omega(n)$) we say this is a *mass* forced on-chain action.

Note that this definition talks about all honest "parties", and hence it includes also the case when the operator is honest, but it is forced to act because of the behavior of the corrupt users. Typically $\alpha = \Omega(n)$ and by "quickly" we mean "1 epoch". In most Plasma proposals [23, 26, 4] "interacting with the smart contract" means simply exiting the Plasma chain with all the coins. Hence, a more common term for this situation is "mass exit". Since in our work we are dealing with the lower bounds, we need to be ready to cover also other, non-standard, ways of protecting honest users' coins. For example, it could be the case that a user U_j does *not* exit immediately, but, instead, keeps her coins in a special account "within Γ" and withdraws them much later. Of course, this requires interacting with Γ immediately, but, technically speaking does not require "exiting". To capture such situations, we use the term "forced on-chain *action*", instead of "mass *exit*".

After a party announces an exit, we need to ensure that she is exiting with the right amount of coins. The main problem comes from the fact that we cannot require that users exit from the last C_i by sending the explanation for her balance b_i to Γ. This is because it could be the case that a given user does not know the explanation $\overrightarrow{b}$ of C_i (due to the data unavailability attack). For a description of how this can be done in practice see [26, 4], or the full version of this paper [11].

Mass exits (or large forced on-chain actions) caused by data unavailability are considered a major problem for Plasma constructions. They are mentioned multiple times in the original Plasma paper [26] (together with some ad-hoc mechanism for mitigating them). They are also routinely discussed on "Ethereum's Research Forum"[4], with even conferences organized on this topic[5]. One of the main reasons why the mass exits are so problematic is that they may results in blockchain congestion (i.e., situations when too many users want to send transactions to the underlying blockchain). Moreover, the adversary can choose to attack Plasma precisely in the moments when the blockchain is already close to being congested (see, e.g., [30] for a description of real-life incident of the Ethereum blockchain congestion). She can also attack different Plasma chains (established over the same main chain) so their users simultaneously send large amounts of data to the blockchain. In order to be prepared for such events in real-life Plasma proposals it is sometimes suggested that the time T for reacting to data unavailability should be very large (e.g. $T = 2$ weeks). This, unfortunately, has an important downside, namely that also an honest coin withdrawal requires time T.

Consider a non-fungible Plasma that supports coin identifiers from some set $\mathcal{C}$. In a non-fungible Plasma the Plasma chain is of a different form than before: instead of a vector of balances $\overrightarrow{b}$, it is a function $f : \mathcal{C} \to \{U_1, \ldots, U_n, \perp\}$ that assigns to every coin $c \in \mathcal{C}$ its current owner $f(c)$ (or $\perp$ if the coin has been withdrawn). Similar to before the commitment to the value of f will be done using Merkle trees. Whenever a coin is withdrawn its identifier c is sent to the smart contract Γ, and hence it becomes public. This is important for the mechanism that prevents parties from stealing coins. To this end, each user U monitors Γ, and sends a complaint whenever some (corrupt) user U' tries to withdraw one of U's coins. For the contract Γ to decide if c belongs to U or U' can require some additional interaction, but the system is designed in such a way that the honest user is guaranteed that finally she will win such dispute. Hence, every malicious attempt to withdraw someone else's coin will be stopped.

The main difference between Plasma Cash and Fungible Plasma is that in Plasma Cash every user has to "protect" only her own coins. Thanks to this, even in case of the data unavailability attack, each honest user U does *not* need to immediately take any action. Instead, she can just monitor Γ, and has to act only if someone tries to withdraw one of U's coins. Of course, the corrupt user can still force all the honest ones to quickly act on the blockchain. However, this requires much more effort from them than in Fungible Plasma, namely: they need to withdraw many coins of the honest users at once, hence forcing the honest users to react. This is "fairer" both honest and malicious users have to make similar effort. Most importantly, however, this attack has *uniquely-attributable faults*.

This advantage of Plasma Cash comes at a price, namely the "exit size" is not constant anymore, as it depends on the number of coins that a user has (since each coin has to be withdrawn "independently"). The Ethereum research community has been making some efforts to deal with this problem. One promising approach is to "compress" the information about withdrawn coins. For example one could assume that the identifiers in $\mathcal{C}$ are natural numbers. Then a user U who owns coins from some interval $[a, \ldots, b]$ (with $a, b \in \mathbb{N}$) could simply withdraw them by posting a message "User U withdraws all coins from the interval $[a, \ldots, b]$ (instead of withdrawing each $i \in [a, \ldots, b]$ independently). This, of course, works only if the coins that users own can be divided into such intervals. Some authors (in particular V. Buterin) have been suggesting "defragmentation" techniques for achieving

[4] Available at: `ethresear.ch`.
[5] See: `ethresear.ch/t/data-unavailability-unconference-devcon4`.

such a distribution of coins. This is based on the assumption that the parties periodically *cooperate* to "clean up" the system. Hence, it does not work in a fully malicious settings[6] (if the goal of the adversary is to prevent the cleaning procedure).

1.1.2 The landscape of Plasmas

Soon after the original groundbreaking work on Plasma [26], some concrete variants have been proposed. Some of them we already described in Sec. 1.1. Since this paper focuses on the *impossibility* results, we do not provide a complete overview of the many different variants that exist and what features they achieve. Plasma projects that are frequently mentioned in the media are Loom, Bankex, NOCUST and OmiseGO [28, 24]. This area is mostly developed by a very vibrant on-line community that typically communicates results in form of so-called "white-papers", blog articles, or post on discussion forums (such as the "Ethereum Research Forum", see footnote 4 on page 7). See also the diagram called "Plasma World Map" illustrating the different flavors of Plasma in the full version of this paper [11]. A notable exception are NOCUST and NOCUST-ZKP described in [21]. This work, up to our knowledge, is the first academic paper on this topic. It provides a formal protocol description (with several interesting innovations such as " Merkle interval trees") and a security argument. Moreover, the authors of [21] describe a version of NOCUST that puts a collateral on the operator (this is done in order to achieve instant transaction confirmation). The authors of [21] (see also [16]) introduce the term "commit chains". Yet, unfortunately, they do not define a full formal security model that we could re-use in our work.

Let us also mention some of the so-called "distributed exchanges" that look very similar to Plasma. One example is StarkDEX (informally described in [15]), which is also based on the idea of a central operator batching transactions using Merkle trees, and a procedure for the users to "escape" from the system if something goes wrong. This protocol uses non-interactive zero-knowledge protocols to ensure correctness of the operator's actions (similar approach has been informally sketched in the original Plasma paper [26], and has been also used in NOCUST-ZKP [21]). While zero knowledge can be used to demonstrate that some data was computed correctly, it *cannot* be used to prove that the off-chain data *was published at all*. Consequently, the authors of this system also encountered the challenge of handling the data unavailability attack. Currently, in StarkDEX this problem is solved by introducing an external committee that certifies if data is available.[7] StarkDEX plans to eventually replace the committee-based solution with an approach that is only based on trusting the underlying blockchain. Our result however shows that in general this will be impossible, as long as fungibility and short exits are required (unless the operator puts a huge collateral).

1.2 Our contribution and organization of the paper

We initiate the study of lower bounds (or: "impossibility results") in the area of off-chain protocols. Our results can also be viewed as a part of a general research program of "bringing order to Plasma". We believe that the scientific cryptographic community can provide significant help in the efforts to systematize this area, and to determine the formal security guarantees of the protocols (in a way that is similar to the work on "Bitcoin backbone" [14], or more recently on "Mimblewimble" [13], state channels [9], or the Lightning Network [22]).

---

[6] See `ethresear.ch/t/plasma-cash-defragmentation` and subsequent posts by Buterin on the Ethereum Research Forum.

[7] See their FAQs at `https://www.starkdex.io`.

Investigating the limits of what Plasma can achieve is part of this process. We focus on proving lower bounds that concern the necessity of mass forced on-chain actions, especially caused by the attack that have no uniquely attributable faults (as a result of data unavailability). This is motivated by the fact that such attacks are particularly important for the off-chain protocols: since the main goal of such protocols is to move the transactions *off*-chain, the necessity of quickly acting *on*-chain can be viewed as a big disadvantage.

We start with a formal definition of Plasma (this is done in Sec. 2). Since in this work we are interested only in the impossibility results, our definition is very restrictive for practical systems. By "restrictive" we mean that we make several assumptions about how the protocol operates. For example we have very strict synchronicity rules, and in particular we only allow the users to start the Plasma operations in certain moments (see "payment orders" and "exit orders" phases in Sec. 2.1). Obviously, such restrictions make our lower bounds only stronger, since they also apply to a more realistic model (without such restrictions). We believe that fully formalizing real-life Plasma (e.g. in the style of [12, 10, 22]) is an important future research project, but it is beyond the scope of this work.

Our main result is presented in Thm. 1, stated formally in Sec. 3. It states that in certain cases the adversary can force mass on-chain actions of the honest users of *any* Plasma system. One subtle point that we want to emphasize is that whenever we talk about "forcing actions" on the honest users, we mean a situation when the users that did *not* want to exit (in a given epoch) are forced to act on-chain. This is important, as otherwise our theorem would hold trivially (one can always imagine a scenario when lots of users decide to exit Plasma because of some other, external, reasons). The notion of "wanting to exit" is formalized by an *environment machine* $\mathcal{Z}$ (borrowed from the *Universal Composability* framework [8]) that "orders" the parties to behave in certain way.

More formally, Thm. 1 states that in Plasma either there exists an attack that provokes a mass action, or there is an attack that requires a party that exits to post long messages on the blockchain (i.e. this Plasma has large exits). Moreover, both attacks have *no* uniquely attributable faults. Note that, strictly speaking, this theorem also covers Plasma systems where the commit size is large (even $\Omega(n)$)[8], but in this case it holds trivially since the honest operator needs to send the large commitments to Γ even if everybody is honest (hence: there is an "unprovoked" mass action in every epoch).

The most interesting practical implication of this theorem is that it confirms the need for "two different" Plasma flavors, as long as the operator is not required to put aside a collateral of size comparable to the total amount of coins in the system[9]. One way to look at it is: either we want to have a Plasma system that does not have large exits, in which case we need to have (non-uniquely attributable) mass actions (this is Fungible Plasma/Plasma MVP); or alternatively we insist on having Plasma without such mass actions, but then we have to live with large exits (as in Plasma Cash). Our theorem implies that there is no Plasma that would combine the benefits of both Fungible Plasma and Plasma Cash, and hence can serve as a justification why both approaches are complementary. Before our result one could hope that the opposite is true and that, e.g., the only reason why Plasma Cash is popular is its relative simplicity (compared to Fungible Plasma). Besides of this reason, and the general scientific interest, we believe that our lower bound has some other important practical applications. In particular, lower bounds often serve as a guideline for constructing new systems or tweaking

[8] In practice, Plasma systems with unbounded commit size are not interesting since they do not bring any advantages to the users. Moreover, they can be trivially constructed just by putting every transaction on the main chain.

[9] This is clearly impractical for most of the applications. Actually most of Plasma constructions assume no such collateral at all.

the definitions. We hope it will contribute to consolidate the countless research efforts in constructing new Plasma systems[10] and simplify identifying proposals that are not sound (e.g., because they claim to achieve the best of both worlds).

Let us also stress that our Thm. 1 does *not* rely on any assumptions of complexity-theoretic type and does *not* use a concept of "black-box separations" [18]. This means that the lower bound that we prove cannot be circumvented by introducing any kind of strong cryptographic assumptions. Hence, of course, it also holds for Plasmas that use non-interactive zero-knowledge (like NOCUST-ZKP [21] or StarkDEX [15]). Moreover, we manage to generalize our lower bound. Thm. 1 even holds for Plasma systems where the operator deposits a certain amount of coins for compensating parties for malicious behavior (e.g., it could be used when a malicious operator does not explain commitments).

For completeness, we also describe (in the full version of this paper [11]) "positive" results, i.e., two protocols that satisfy our security definition (Plasma Cash and Fungible Plasma). We stress that we do not consider it to be a part of our main contribution, and we do not claim novelty with these constructions, as they strongly rely on ideas published earlier (in particular [26, 5, 23, 4, 21, 15]).

Notation. For a formal definition of an *interactive (Turing) machine* and a *protocol*, see, e.g., [8]. In our modeling the communication between the parties is synchronous and happens in *rounds* (see Section 2). During the execution of the protocol a party P may send messages to a party P'. A *transcript* of the messages sent from P to P' is a sequence $\{(m_i, t_i)\}_{i=1}^{\ell}$, where each m_i was sent by P to P' in the t_i-th round. A transcript of messages sent from some set of parties to a different set of parties is a sequence $\{(W_i, W'_i, m_i, t_i)\}_{i=1}^{\ell}$, where each m_i was sent by W_i to W'_i in the t_i-th round. By the *length of a transcript* we mean its bit-length (in some fixed encoding). We sometimes refer to it also as *communication size* (between the parties).

2 Plasma Payment Systems

A *Plasma payment system* (or "Plasma" for short) is a protocol Π consisting of a randomized non-interactive machine Ψ representing the *setup* of the system; deterministic[11] interactive poly-time machines $U_1, \ldots, U_n, Op$ representing the *users* and the *operator* of the Plasma system (respectively); and a deterministic interactive poly-time machine Γ, which represents the *Plasma contract*. We use the notation $\mathcal{U} = \{U_1, \ldots, U_n\}$ to refer to the set of users of the system. The contract machine Γ has no secret state, and moreover its entire execution history is known to all the parties. We can think of it as a Turing machine that keeps the entire log of its execution history, and moreover all the other parties in the system have a (read-only) access to this log. The Plasma system comes with a parameter $\gamma \in \mathbb{R}_{\geq 0}$ called *operator's collateral fraction*. Informally, this parameter describes the amount of coins that are held by the operator as a "collateral" (as a fraction of user's coins). These coins can be used to cover users' losses if the operator misbehaves. This is formally captured in Section 2.2 (see "limited responsibility of the operator"). If $\gamma = 0$ then we say that the operator is *not collateralized*. We introduce the notion of collateral in order to make our results stronger and to cover also cases of real-life systems that have such a collateral (e.g., NOCUST, see Section 1.1).

[10] see `https://ethresear.ch/c/plasma/`.

[11] We assume that these machines are deterministic, since all their internal randomness will be passed to them by Ψ.

The protocol is attacked by a randomized poly-time adversary $\mathcal{A}$. We assume that $\mathcal{A}$ can corrupt any number of users and the operator except the contract Γ (hence Γ can be seen as a trusted third party). Once $\mathcal{A}$ corrupts a party P, she learns all its secrets, and takes full control over it (i.e. she can send messages on behalf of P). A party that has not been corrupted is called *honest*. An execution of a Plasma payment system Π is parametrized by the *security parameter* 1^λ.

To model the fact that users perform actions, we use the concept of an environment $\mathcal{Z}$ (which is also a poly-time machine) that is responsible for "orchestrating" the execution of the protocol. The environment can send and receive messages from all the parties (it also has full access to the state of Γ). It also knows which parties are corrupt and which are honest. For an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, a pair $(\mathcal{A}, \mathcal{Z})$ will be called an *attack* (on a given Plasma system Π). The contract machine Γ can output special messages $(\mathsf{attribute\text{-}fault}, P)$ (where $P \in \mathcal{U} \cup \{Op\}$). In this case we say that Γ *attributed a fault to* P. We require that the probability that Γ attributes a fault to an honest party is negligible in λ. An attack $(\mathcal{A}, \mathcal{Z})$ *has no attributable faults* if the probability that Γ attributes a fault to some party is negligible.

2.1 Protocol operation

Let us now describe the general scheme in which a Plasma payment protocol operates. In this section, we focus only on describing what messages are sent between the parties. The "semantics" of these messages, and the security properties of the protocol are described in Section 2.2. We assume that all the parties are connected by authenticated and secret communication channels, and a message sent by a party P in the ith round, arrives to P' at the beginning of the $(i+1)$th round. The communication is synchronous and happens in *rounds*. It consists of three stages, namely: "setup", "initialization", and "payments". The execution starts with the *setup stage*. In this stage parameter 1^λ is passed to all the machines in Π. Upon receiving this parameter, machine Ψ samples a tuple $(\psi_{U_1}, \ldots \psi_{U_n}, \psi_{Op}, \psi_\Gamma)$ (where each $\psi_P \in \{0,1\}^*$). Then for each $P \in \{U_1, \ldots, U_n, Op, \Gamma\}$ the string ψ_P is passed to P. Afterwards, the parties proceed to the *initialization stage*. In this stage the environment generates a sequence $(a_1^{\mathsf{init}}, \ldots, a_n^{\mathsf{init}})$ of non-negative integers and passes it to the contract Γ (recall that the state of Γ is public, and hence, as a consequence, all the parties in the system also learn the a_i^{init}'). Then the protocol proceeds to the *payment stage*. This stage consist of an unbounded number of *epochs*. Each ith epoch (for $i = 1, 2, \ldots$) is divided into two *phases*.

Payment phase. In this phase the environment sends a number of *payment orders* to the users (for simplicity we assume that this happens simultaneously in a single round). Each order has a form of a message "(send, v, U_i)", where $v \in \mathbb{Z}_{\geq 0}$, and $U_i \in \mathcal{U}$. It can happen that some users receive no payment orders in a given epoch. It is also ok if a user receives more than one order in an epoch. Informally, the meaning of these messages is as follows: if a user U_j receives a "(send, v, U_i)" message, then she is ordered to transfer v coins to user U_i. We require that this message can only be sent if none of b_i and b_j are equal to $\perp$ (i.e.: if none of U_i and U_j "exited", see below). The parties execute a multiparty sub-protocol. During this executions some of the users send a message "$(\mathsf{received}, v, U_i)$" to the environment $\mathcal{Z}$. This sub-protocol ends when Γ outputs a message $\mathsf{payments\text{-}processed}$.

Exit phase. In this phase the environment sends *exit orders* to some of the users (again: this happens in a single round). Each such order is simply a message "exit". Informally, sending this message to some U_i means that U_i is ordered to exit the system with all

her coins. The environment can send an exit message to U_i only if $b_i \neq \perp$ (i.e. U_i has not already "exited", see below). The parties again execute a multiparty protocol. The protocol ends when Γ outputs a sequence

$$\{(\mathsf{exited}, U_{i_j}, v_{i_j})\}_{j=1}^{m}, \tag{1}$$

where m is some non-negative integer, and each $U_{i_j} \in \mathcal{U}$ and $v_{i_j} \in \mathbb{Z}_{\geq 0}$. For each U_{i_j} in Eq. (1) we say that U_{i_j} *exited (with* v_{i_j} *coins)*, and we let $b_{i_j} := \perp$. We require that no party can exit more than once. In other words: it cannot happen that two messages $(\mathsf{exited}, U_i, v)$ and $(\mathsf{exited}, U_i, v')$ are issued by Γ.

We make some assumptions on the communication between the parties. Informally we require that if U and U' are some honest users, then the procedure of transferring coins from U to U' is done by a "sub-protocol" involving only parties in the set U and U'. Since we do not have a concept of "sub-protocol" this is formalized as follows:

Communication locality. Two honest users U and U' exchange messages only in epochs in which they do transactions between each other (i.e. a message (send, U, v) is sent by $\mathcal{Z}$ to U', for some v).

This requirement is very natural since Plasma is supposed to work even when an arbitrary set of users is corrupt. Hence, relying on the other users' help in financial transfers would be impractical. Up to our knowledge all "pure" Plasma proposals in the literature satisfy this requirement. On the other hand: it may *not* hold if we incorporate some techniques that assume some type of cooperation between larger sets of parties (e.g. consensus mechanisms). Examples include: Buterin's Plasma Chash defragmentation (where a large set of users has to regularly cooperate in order to "clean-up" the system), and StarkDEX's "data availability committee" (see Section 1.1), if we treat the committee members as "users". One way to view our result is that it implies that such techniques are inherent for every fungible Plasma.

2.2 Security properties

During the interaction with the protocol, the environment keeps track of balances of *honest* users (we do not define balances of dishonest users). Formally, for each honest user U_i it maintains a variable $b_i \in \mathbb{Z}_{\geq 0} \cup \{\perp\}$ (a *balance* of U_i), where the symbol "$\perp$" means that a party exited. It also maintains a variable $t \in \mathbb{Z}_{\geq 0}$ (initially set to 0) that is used to keep track on the amount of coins that have been withdrawn. The rules for maintaining these variables are as follows. Initially, for each $i := 1, \ldots, n$ the environment $\mathcal{Z}$ lets $b_i := a_i^{\mathsf{init}}$. Whenever Γ outputs $(\mathsf{exited}, U_i, v)$ (for some U_i and v) we let $b_i := \perp$ and increment t by v. Each time $\mathcal{Z}$ receives a message $(\mathsf{received}, v, U_i)$ from some *honest* U_j, it adds v to b_j (recipient balance) and, if U_i (sender) is honest too, subtracts v from b_i. We require that the environment never issues an order if U_i or U_j exited (i.e. if $b_i = \perp$ or $b_j = \perp$). The environment also never sends an order exit to the same user more than once, and it never sends exit order to a user U_i that already exited (i.e. such that $b_i = \perp$). We have the following security properties.

Responsiveness to "send" orders. Suppose Op, U_i, and U_j are honest, and the environment issued an order (send, v, U_i) to U_j then in the same epoch party U_i sends a message "$(\mathsf{received}, v, U_j)$" to the environment.

Correctness of "received" messages. Suppose U_i and U_j are honest and U_i outputs a message "$(\mathsf{received}, v, U_j)$", then environment has issued an order (send, v, U_i) to U_j in the same epoch.

Responsiveness to "exit" orders. Suppose U_i is honest and the environment issued an order exit to U_i then in the same epoch Γ outputs a message $(\mathsf{exited}, U_i, v)$ (for some v).

No forced exits if operator honest. Suppose Op and U_i are honest and Γ outputs message $(\mathsf{exited}, U_i, v)$ at epoch r, then environment has sent the order exit to U_i in the same epoch.

Fairness for the users. If Γ outputs a message $(\mathsf{exited}, U_i, v)$ (for some honest U_i) then $v \geq b_i$ (where b_i is the current balance of U_i).

Limited responsibility of the operator. If the operator is honest, then the total amount of coins that are withdrawn from the system is at most $a_1^{\mathsf{init}} + \cdots + a_n^{\mathsf{init}}$. Otherwise (if she is dishonest) the total amount of coins that are withdrawn from the system is at most $\lceil (1+\gamma)(a_1^{\mathsf{init}} + \cdots + a_n^{\mathsf{init}}) \rceil$. This definition captures the notion of operator's collateral, and the fact that it is used (to cover users' losses) if the operator is caught cheating.

If an attack $(\mathcal{A}, \mathcal{Z})$ succeeds to violate any of the requirements from this section, then we say that $(\mathcal{A}, \mathcal{Z})$ *broke a given Plasma payment system*. We say that Π is *secure* if for every environment $(\mathcal{A}, \mathcal{Z})$ the probability that $\mathcal{A}$ breaks Π is negligible in 1^λ.

As explained in the introduction, certain attacks on Plasma are of particular importance, due to the fact that they are hard to handle in real life. We say that $(\mathcal{A}, \mathcal{Z})$ *force an on-chain action of size M (in some epoch i)* if the following happened. Let T be the set of honest parties that did *not* receive any order from $\mathcal{Z}$ in epoch i. Then the total length of messages sent by parties from T to Γ is at least M. As explained in the introduction, the term that is more standard than "forced on-chain action" is "mass *exit*". See 1.1 for a discussion why "forced on-chain action" is a better term when impossibility results are considered.

3 Our main result

We now present Thm. 1, which is the main result of this paper. The main implication of this theorem is that for every non-collateralized Plasma system there exists an attack that provokes a mass forced on-chain action, i.e., it forces the honest users to make large communication with the contract even if they did not receive any exit order from the environment (see point 1 in the statement of the theorem), unless a given Plasma system has large exits (point 2). Moreover, this can be done by an attack that has *no* uniquely attributable faults. This fact cannot be circumvented by putting a collateral on the operator, unless this collateral is very large.

▶ **Theorem 1** (Mass forced on-chain actions or large exits without uniquely attributable faults are necessary)**.** *Let Π be a secure Plasma payment system with n users and let $\gamma \geq 0$ be the operator's collateral fraction. Then either*

1. *there exists an attack on Π that causes a forced on-chain action of size greater than $(n - \lceil \gamma n \rceil \cdot \log_2 n - 5)/4$ with probability at least $1/16 + \mathsf{negl}(\lambda)$, or*
2. *there exists an attack on Π such that* one *honest user, when ordered to exit by the environment, makes communication to Γ of size at least $(n - \lceil \gamma n \rceil \cdot \log_2 n - 5)/4$ with probability at least $1/16 + \mathsf{negl}(\lambda)$.*

Moreover, both attacks have no uniquely attributable faults.

One way to look at this theorem is as follows. First, consider a non-collateralized Plasma, i.e., assume that $\gamma = 0$. Let $\mathcal{P}^1$ be a class of non-collateralized Plasma that with overwhelming probability *do not* have uniquely attributable forced on-chain actions (of any size larger than 0). In this case point 1 cannot hold, and hence, every Plasma $\Pi \in \mathcal{P}^1$ needs to satisfy point 2. This means that there exists an attack on every $\Pi \in \mathcal{P}^1$ such that one honest user, when

ordered to exit by the environment, makes communication to Γ of size at least $(n-5)/4$ with probability around $1/16$. Or, in other words: every Plasma from class $\mathcal{P}^1$ must have a large exist size with noticeable probability. We know Plasma with such properties: it is essentially Plasma Cash (see the full version of this paper [11])

On the other hand, let $\mathcal{P}^2$ be a class of non-collateralized Plasmas that with high probability have no large exits, in the sense of point 2 of Thm. 1. This means that point 1 has to hold, which implies that every $\Pi \in \mathcal{P}^2$ needs to have large (at least around $(n-5)/4$) non-uniquely attributable mass forced on-chain actions. Plasma with such properties is called Fungible Plasma (see the full version of this paper [11]) Hence, informally speaking, Thm. 1 states that we cannot have the "best of two types of Plasma" simultaneously.

If we consider non-zero collaterals, i.e., we let $\gamma > 0$ then the situation does not improve much, unless the collateral fraction is large, i.e., the total collateral blocked by the operator is at least around $n \cdot \gamma = n/\log_2 n$[12]. This essentially means that we cannot get around the bounds from Thm. 1 by introducing collateral, unless the amount of coins blocked in operator's collateral is of roughly the same order as the total amount of coins stored by the users.

Note that even trivial versions of Plasma "fit" into Thm. 1. For example, consider Plasma in which the operator always puts all the transactions on-chain. Of course, the details would need to be worked out, but clearly such a Plasma can be made secure. The existence of such a trivial Plasma does not contradict our Thm. 1, since it clearly satisfies point 1: a large number of transfers in one epoch will cause a forced mass on-chain action (by the operator). The same also holds if every user needs to put each transaction on-chain.

4 Proof of Theorem 1

Before we present the proof let us introduce some auxiliary machinery. This is done in the next section.

4.1 Isolation scenario

Let Π be a Plasma payment system, let $\mathcal{Z}$ be an environment, and let $\mathcal{W}$ be some subset of the users of Π. We now introduce a procedure that we call *isolation of* $\mathcal{W}$. In this scenario Π is executed as in the normal execution, except that we "isolate" the users $\mathcal{W} \subseteq \mathcal{U}$ from the operator. More precisely: all the messages sent between any $U \in \mathcal{W}$ and the operator Op are dropped, i.e., they never arrive to the destination. This scenario can be viewed as an "attack" although it does not fit into the framework from Section 2.1, since it violates the assumption that messages sent by an honest party to another honest party always arrive to the destination.

Although the isolation scenario cannot be performed within our model, it can be "emulated" by corrupting either the operator Op, or the users from $\mathcal{W}$. In the first case we corrupt the operator and instruct him to behave as if she was honest, except that she does not send messages to the users in $\mathcal{W}$ and ignores all messages sent by these users. This will be called the *data unavailability (DU) attack against* $\mathcal{W}$ *by the operator*. In the second, symmetric case (the *pretended data unavailability (PDU) attack by* $\mathcal{W}$ *on the operator*) we corrupt the users in $\mathcal{W}$. Then, every user $U \in \mathcal{W}$ behaves as if she was honest, except that she does not send messages to Op, and ignores all messages from Op.

[12] This is because we need to have $\gamma \approx 1/\log_2 n$ to make the expression "$(n - \lceil \gamma n \rceil \cdot \log_2 n - 5)/4$" equal to 0.

If this is the only type of malicious behavior, then "from the point of view" of all the other parties, and, most importantly, from the point of view of the contract machine Γ, it is impossible to say who is corrupt (the users in $\mathcal{W}$ or the operator Op). More precisely, we have the following.

▶ **Observation 1.** *Let Π be a Plasma payment system and consider the attack that isolates users in some set $\mathcal{W}$ from the operator. Let $\mathcal{Z}$ be an arbitrary environment and let $\mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{isolate}}$ be the random variable denoting the transcript of messages received by Γ. Moreover, let $\mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{PDU}}$ and $\mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{DU}}$ be the random variable denoting the transcripts of messages received by Γ in the PDU attack and in the DU attack (respectively), both with environment $\mathcal{Z}$. Then $\mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{DU}} \stackrel{d}{=} \mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{isolate}} \stackrel{d}{=} \mathcal{T}^{\mathcal{W},\mathcal{Z}}_{\text{PDU}}$.*

This fact is useful in the proof of the following simple lemma.

▶ **Lemma 1.** *Fix an arbitrary Plasma Π. Let $\mathcal{W}$ be some set of users. Suppose $\mathcal{A}$ performs a DU attack against $\mathcal{W}$ or a PDU attack by $\mathcal{W}$ (either by corrupting the operator or by corrupting the users), and let $\mathcal{Z}$ be arbitrary. Then the attack $(\mathcal{A},\mathcal{Z})$ has no uniquely attributable faults.*

Proof. From the security of Π we get that if the users are corrupt then the probability that Γ attributes a fault to them is negligible. Symmetrically, if the operator is corrupt then the probability that Γ attributes a fault to her is negligible. By Observation 1 the transcripts of messages received by Γ in both attacks are distributed identically, so the probability that Γ attributes *any* fault has to be negligible. ◀

4.2 Proof overview

Fix some secure Plasma payment system Π that works for n users. We construct either an attack such that

$$\Pr\left[\begin{array}{c}\text{the set of all honest users makes communication to } \Gamma \text{ of size}\\ \text{at least } (n-\lceil \gamma n\rceil \cdot \log_2 n - 5)/4\\ \text{(without receiving an } \mathsf{exit} \text{ order from the environment)}\end{array}\right] \geq 1/16 + \mathsf{negl}(\lambda), \quad (2)$$

or an attack such that

$$\Pr\left[\begin{array}{c}\text{user } U_1\text{, when ordered to exit by the environment, makes}\\ \text{communication to } \Gamma \text{ of size at least}\\ (n-\lceil \gamma n\rceil \cdot \log_2 n - 5)/4\end{array}\right] \geq 1/16 + \mathsf{negl}(\lambda). \quad (3)$$

In both of these attacks the amount of coins given to the users is n, but our proof can be generalized to cover cases when it is required that the amount of coins is larger than n (we comment more on this at the end of Section 4). On the other hand, the proof does not go through in the (unrealistic) case when this amount is very small (sublinear in n).

The attacks that we construct in both cases ((2) and (3)) have no uniquely attributable faults. Note that for $n \leq 5$ Eq. (3) holds trivially, and therefore we can assume that $n > 5$. Let Υ denote the family of all *non-empty proper* subsets of $\{U_2,\ldots,U_n\}$, i.e. sets $\mathcal{V}$ such that $\emptyset \subsetneq \mathcal{V} \subsetneq \{U_2,\ldots,U_n\}$ (note that $U_1 \notin \mathcal{V}$). Since we assumed that $n > 5$ we have that $\log|\Upsilon| = \log_2(2^{n-1}-2) \geq n-2$, and, in particular, Υ is non-empty. In the proof we construct an experiment (denoted by $\mathsf{Exp}(\mathcal{V})$ and presented in details in the full version of this paper [11]) and analyze its performance, assuming that $\mathcal{V}$ is sampled uniformly at random from Υ. Depending on this analysis, the experiment $\mathsf{Exp}(\mathcal{V})$ can be "transformed" into an attack that satisfies Eq. (2) or Eq. (3).

Experiment $\mathsf{Exp}(\mathcal{V})$ "simulates" an execution of two epochs of Plasma II. In the first epoch the adversary isolates the users in $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$ from the operator (in the attacks that we construct later this will be done either by corrupting these users, or the operator). The environment gives 1 coin to each user $U \in \mathcal{U}$. Then, in the "payment" phase of the first epoch all the users from $\mathcal{V}$ transfer their coins to U_1. In the "exit" phase of the first epoch user U_1 receives an exit order from the environment and consequently exits with all her coins. Note that in the first epoch every party behaved honestly (except of the isolation attack against the users in $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$), and hence U_1 is guaranteed to successfully exit with her coins (she has 1 such coin from the "initialization" phase, and $|\mathcal{V}|$ coins that were transferred to her by the users in $\mathcal{V}$).

Of course the honest parties from $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$ will usually realize that they are isolated from the operator. As a reaction to this they may send some messages to Γ. This, in turn can provoke the other parties to react by sending their messages to Γ. Hence, in general there can be a longer interaction between all the parties and Γ in this phase. Let $\mathcal{T}^1$ be the transcript of the messages sent by the users in $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$ to Γ in both phases, let $\mathcal{T}^2$ be the messages sent by the users in $\mathcal{V}$ and the operator to Γ in both phases, let $\mathcal{T}^3$ be the messages sent by U_1 to Γ in the "payment" phase, and finally let $\mathcal{T}^4$ be the messages sent by U_1 to Γ in the "exit" phase. The first epoch of the experiment $\mathsf{Exp}(\mathcal{V})$ and the transcripts are depicted on Fig. 1.

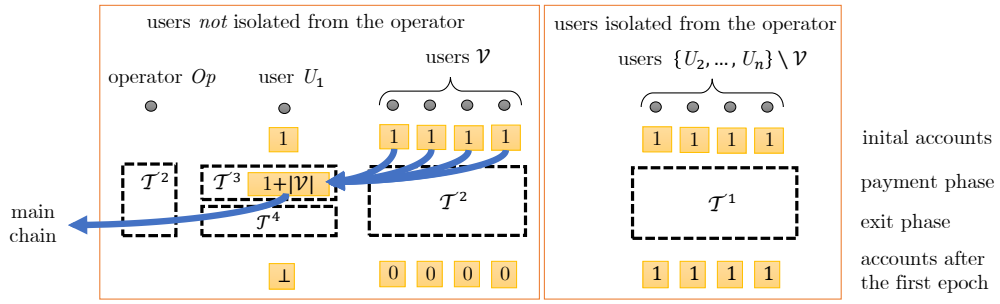

Figure 1 The first epoch of the experiment $\mathsf{Exp}(\mathcal{V})$. Gray circles denote the parties, and the $\mathcal{T}^i$'s denote the transcripts of the communication with Γ (see, e.g., Section 4.2 for their definitions).

Before discussing the second epoch of the experiment, let us note that in the first epoch the only way in which we deviate from the totally honest execution is the "isolation" of $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$. This will later allow us to be "flexible" and corrupt different sets of parties ($\{Op\}$ or $\{U_2, \ldots, U_n\} \setminus \mathcal{V}$) depending on the results of our analysis of $\mathsf{Exp}(\mathcal{V})$. This will be different in the second epoch, where we always assume that parties from $\mathcal{V}$ are corrupt. This is ok because while constructing the attacks that satisfy (2) or (3) we will only use the first epoch of $\mathsf{Exp}(\mathcal{V})$. The only reason to have the second epoch of $\mathsf{Exp}(\mathcal{V})$ is to make sure that the users have to send large amounts of data to Γ during the first epoch, as otherwise corrupt $\mathcal{V}$ can steal the money (in the second epoch).

Let us now present some more details of the second epoch. Initially we corrupt all the users from $\mathcal{V}$ and "rewind" them to the state that they had at the beginning of the first epoch. This is done in order to let them "pretend" that they still have their coins. We then let all of them try to ("illegally") exit with these coins. Technically, "rewinding a user U" is done via a procedure denoted $\mathsf{Reconstruct}_U$. This procedure outputs the state that U would have at the end of the "payment" phase if she did not transfer her coins to U_1. To make it look consistent with the state of Γ, this procedure takes as input the transcripts defined

above. Then, each user $U \in \mathcal{V}$ tries to exit (in the "exit" phase) from her state computed by $\mathsf{Reconstruct}_U$. Also the honest users try to exit (they receive an "exit" order from the environment). Let $\mathcal{Q}$ be the set of users that managed to exit with at least 1 coin. From the security of Plasma we get that $\mathcal{Q}$ is equal to the set of honest users ($\{U_2, \ldots, U_n\} \setminus \mathcal{V}$) plus a small (of size at most $\lceil \gamma n \rceil$) subset $\mathcal{D}$ of dishonest users.

The key observation is now that all that is needed to "simulate" the second epoch of $\mathsf{Exp}(\mathcal{V})$ are the transcripts $\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3$, and $\mathcal{T}^4$. On the other hand $\mathcal{V}$ can be approximately computed from $\mathcal{Q}$ (i.e., we can compute $\mathcal{V}$ with elements $\mathcal{D}$ missing, where $|\mathcal{D}| = \lceil \gamma n \rceil$). Hence the variable $(\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3, \mathcal{T}^4)$ carries enough information to "approximately" describe $\mathcal{V}$. Thanks to this we can construct a "compression" algorithm that "compresses" a random $\mathcal{V} \leftarrow\!\!\$\ \Upsilon$ by simulating the first epoch of $\mathsf{Exp}(\mathcal{V})$ and obtaining $(\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3, \mathcal{T}^4)$ and then "decompresses" it by simulating the second epoch, and computing the output as $\mathcal{V} := \{U_2, \ldots, U_n\} \setminus \mathcal{Q}$ (the additional $\lceil \gamma n \rceil$ elements can be simply listed as an additional output of C and passed to D as input that has to be added to the output of D).

On the other hand, clearly (for completeness we show this fact in the full version of this paper [11]), a random $\mathcal{V} \leftarrow\!\!\$\ \Upsilon$ with high probability cannot be compressed to a string that is significantly shorter than $\log |\Upsilon| \geq n - 2$. This implies that with a noticeable probability $|(\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3, \mathcal{T}^4)| \approx n - \lceil \gamma n \rceil \log_2 n$, where $\lceil \gamma n \rceil \log_2 n$ is the number of bits needed to describe set $\mathcal{D}$.

Obviously, the above fact implies that for at least one $i \in \{1, \ldots, 4\}$ we have that $\mathcal{T}^i \geq (n - \lceil \gamma n \rceil \log_2 n)/4$ with noticeable probability for concrete parameters). The rest of the proof of Thm. 1 is based on the case analysis of the implications of "$\mathcal{T}^i \geq n/4$" for different i's. More concretely, we show that in the first three cases ($i = 1, 2$, and 3) we can construct attacks that satisfy Eq. (2), and in case $i = 4$ – an attack that satisfies Eq. (3). All these attacks are based on the experiment $\mathsf{Exp}(\mathcal{V})$, but are only using its first epoch. In the proof we exploit the fact that the only malicious behavior that happens in this epoch is the "isolation" (i.e., not sending messages). Hence, we can use Observation 1 and "switch" between scenarios when different groups of parties are corrupt (while still getting the same transcripts $\mathcal{T}^i$). Moreover these attacks do not have uniquely attributable faults.

The detailed proof of Thm. 1 can be found in the full version of this paper [11].

▶ Remark 1. Our proof would also go through even if the total balance of the users a is arbitrarily large. The only difference would be that instead of giving 1 coin to every user, the environment would give to each user U_i (for $i > 1$) $\lfloor a/n \rfloor$ coins, and to user U_1 the environment would give the remaining coins (say). The rest of the proof would be essentially identical to the proof of Thm. 1.

▶ Remark 2. Although the attack presented requires two epochs, the second epoch only captures the scenario where the underlying protocol is insecure and hence it can be seen as a "thought experiment". In other words, if the honest parties do not make large communication with the contact Γ in the first epoch, they risk loosing their coins in the second epoch. Therefore, under the assumption that the plasma system is indeed secure and consequently parties make large communication with Γ in the first epoch, the adversary cannot steal any coins in the second epoch and hence the second epoch would become obsolete.

Conclusion

The main contribution of this work is that we have shown that the distinction between Plasma Cash and Fungible Plasma is inherent, i.e., we ruled out the possibility of constructing Plasma that combines benefits of both Plasmas. We believe that, besides of the general

scientific interest, our work (especially ruling out existence of some Plasma constructions) can help the practical blockchain community in developing Plasma protocols, and in general can bringing more understanding in what is possible and what is impossible in the area of off-chain protocols, and under what assumptions. It can also serve as a formal justification why "hybrid" approaches (such a "rollups") [6] may be needed in real life. We also hope that this work may expand the scope of theory by identifying a new area where theoretical lower bounds can have direct impact on the real life problems.

References

1 N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997*, pages 7–17. ACM, 1997. doi:10.1145/266420.266426.

2 Vitalik Buterin. A note on limits on incentive compatibility and griefing factors. URL: https://vitalik.ca/files/extortion_griefing_bounds.pdf.

3 Vitalik Buterin. Scalability, part 2: Hypercubes. URL: https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/.

4 Vitalik Buterin. Minimal viable plasma, 2018. URL: https://ethresear.ch/t/minimal-viable-plasma.

5 Vitalik Buterin. Plasma cash: Plasma with much less per-user data checking, 2018. URL: https://ethresear.ch/t/plasma-cash-plasma-with-much-less-per-user-data-checking/1298.

6 Vitalik Buterin. The dawn of hybrid layer 2 protocols. https://vitalik.ca/general/2019/08/28/hybrid_layer_2.html, August 2019. (Accessed on 02/08/2020).

7 Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2000. doi:10.1007/3-540-44598-6_6.

8 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959888.

9 Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 967–984. ACM, 2018. doi:10.1145/3243734.3243857.

10 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 625–656. Springer, 2019. doi:10.1007/978-3-030-17653-2_21.

11 Stefan Dziembowski, Grzegorz Fabiański, Sebastian Faust, and Siavash Riahi. Lower bounds for off-chain protocols: Exploring the limits of plasma. Cryptology ePrint Archive, Report 2020/175, 2020. URL: https://eprint.iacr.org/2020/175.

12 Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 949–966. ACM, 2018. doi:10.1145/3243734.3243856.

13 Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 657–689. Springer, 2019. doi:10.1007/978-3-030-17653-2_22.

14 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. doi:10.1007/978-3-662-46803-6_10.

15 Lior Goldberg and Oren Katz. Starkdex deep dive: Contracts & statement - starkware - medium. https://medium.com/starkware/tagged/starkdex-specs, 2019. (Accessed on 02/08/2020).

16 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 201–226. Springer, 2020. doi:10.1007/978-3-030-51280-4_12.

17 Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991. doi:10.1007/BF00196791.

18 Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 44–61. ACM, 1989. doi:10.1145/73007.73012.

19 Harry A. Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *14th Annual Workshop on the Economics of Information Security, WEIS 2015, Delft, The Netherlands, 22-23 June, 2015*, 2015. URL: http://www.econinfosec.org/archive/weis2015/papers/WEIS_2015_kalodner.pdf.

20 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

21 Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. Cryptology ePrint Archive, Report 2018/642, 2018. URL: https://eprint.iacr.org/2018/642.

22 Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. *IACR Cryptology ePrint Archive*, 2019:778, 2019. URL: https://eprint.iacr.org/2019/778.

23 Georgios Konstantopoulos. Plasma cash: Towards more efficient plasma constructions, 2019. URL: https://www.gakonst.com/plasmacash.pdf.

24 Rajarshi Mitra. Plasma breakthrough: Omisego (omg) announces the launch of ari. https://www.fxstreet.com/cryptocurrencies/news/plasma-breakthrough-omisego-omg-announces-the-launch-of-ari-201904120245. (Accessed on 02/08/2020).

25 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL: http://bitcoin.org/bitcoin.pdf.

26 Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts, 2017. URL: http://plasma.io/plasma.pdf.

27 Nick Szabo. Smart contracts: Building blocks for digital markets. Extropy Magazine.

28 Trustnodes. Ethereum transactions fall off the cliff, three plasma projects close to release says buterin, 2018. URL: https://www.trustnodes.com/2018/07/05/ethereum-transactions-fall-off-cliff-three-plasma-projects-close-release-says-buterin.

29 Wikipedia. Trusted timestamping. URL: https://en.wikipedia.org/wiki/Trusted_timestamping.

30 Joon Ian Wong. The ethereum network is getting jammed up because people are rushing to buy cartoon cats on its blockchain. Quartz, 2017. URL: https://qz.com/1145833/cryptokitties-is-causing-ethereum-network-congestion/.

31 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2016. URL: http://gavwood.com/paper.pdf.