TÉLÉCOM SUDPARIS

FINAL YEAR ENGINEERING PROJECT

Anonymizing transactions on the blockchain

Authors
Enguerrand DECLERCQ
Simon CHEREL

Supervisors
Olivier PAUL
Nesrine KAANICHE

Disclaimer : this report is still in development. It doesn’t reflect the final result.

ABSTRACT

Table of contents

I. Description of the project

This final-year project consists of several key objectives:

- Obtaining an extensive understanding of the blockchain technology along with its privacy limitations;
- Providing a technical overview of the state-of-the-art privacy-enhancing tools - specifically transaction mixers - blockchain users can benefit from;
- Breaking down the mechanisms and key principles powering Tornado, the controversial and most famous Ethereum-based transaction mixer;
- Implementing our own on-chain mixer on the Tezos blockchain while mitigating legal issues risks.

II. Network topologies

Before explaining what a blockchain is, the reader needs to get familiar with the different network topologies that exist. There are indeed three types of such topologies : centralized, decentralized and distributed.

1. Centralized networks

The first topology characterizes networks in which data is processed by a single operator node i.e. a central server. End users, each of them representing a node, do not need to talk to each other to transfer data, instead they use the central server as a relayer. Such a relayer is trusted to process data properly.

The advantage of such a network is that data is processed in a fast way, it is easy to maintain and outsources the infrastructure, however users need to trust the single node that their data is used in good faith. Besides, a lonely master node means having a single point of failure: if it fails to operate for any reason, the entire network collapses. The scalability of such a network is also limited, since its capacity to handle the workload depends on a single server, which has physical constraints in terms of bandwidth, computing power and memory.

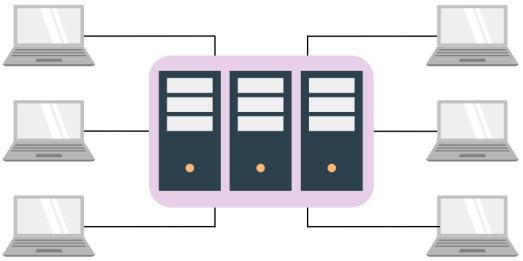

Fig. 1.a: a centralized network

2. Decentralized networks

On the other hand, a decentralized network architecture distributes the workload among several machines instead of relying on a single central server. Thereby, such a network is a lot more scalable and reliable. Besides, having multiple servers allows data redundancy. However, because there are a lot more machines than in a centralized network, it is more expensive and harder to maintain. Besides, accessing and mutating data is a highly complex task in a decentralized network. For instance, when two servers are being queried simultaneously to update the same piece of data, which server should be prioritized?

Nonetheless, having multiple master nodes is the only way to enhance a network's availability. In a blockchain, nodes do not need to trust each other and need to find an agreement on the state of data, which is done through a consensus protocol. In the blockchain world, a cloud provider such as AWS would not be qualified as a decentralized network even though they have multiple servers to operate and host data. Indeed, AWS servers trust each other and thus do not need any form of consensus to agree on the state of data since a single node would not benefit from acting maliciously.

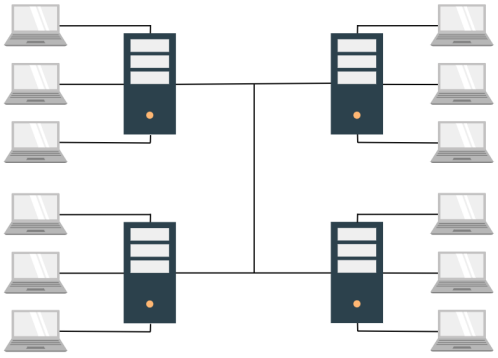

Fig. 1.b: a decentralized network

3. Distributed networks

A distributed network is a special kind of decentralized network. While they are often referred to interchangeably, the main difference between a distributed and a decentralized network is that the computing power is evenly spread across the network in a distributed topology. There are distributed networks in which each end-user is an actual operator node.

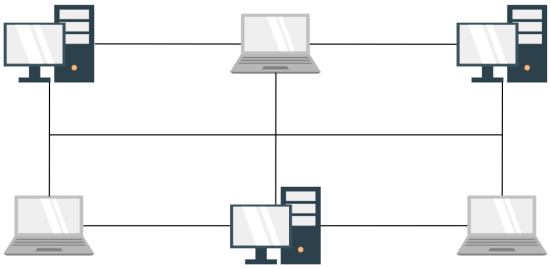

Fig. 1.c: a distributed network

III. The blockchain

1. An example of decentralized network

The word blockchain rather designates a way to store data - literally a tamper-proof chain of transaction blocks - than an underlying network. In that sense, the git protocol uses a blockchain mechanism to authenticate the commits history: to prevent manipulations, each commit hash is obtained thanks to the previous hashes. However, git is generally not referred to as a blockchain, since the word blockchain implies the existence of an underlying network of rewarded validators. Indeed, a blockchain relies on a network of supposedly independent master nodes which are paid to validate and host data.

The data stored in a blockchain can be seen as a public database that anyone with an Internet connection can read and write into. Users can write into the blockchain through network transactions, which do not necessarily refer to payments. In order to be added to the blockchain, a transaction needs to be valid. However, validating data without any trusted third party is challenging and is thereby performed by a theoretically decentralized quorum of master nodes.

One of the specificities of the blockchain is that users need to pay for the execution of their transactions. These indemnities are called gas fees; their amount is proportional to the computational effort (quantified in gas units) required to validate a given transaction. For instance, since sending money over the blockchain (its most primitive use case) requires less computation than the execution of complex smart contracts, and will thus be executed at a cheaper price. Gas fees are notably a convenient way to pay for the validators' electric consumption and to limit denial of service risks at the same time. Without gas fees,

2. Reaching consensus

The transaction verification process differs from one blockchain to another, depending on which consensus is used. A consensus is an agreement mechanism on the state of the machine itself. While there is a wide variety of consensus, it is important to remember Proof-of-Work and Proof-of-Stake are by far the most used. Some mechanisms, such as Binance Smart Chain's Proof-of-Authority consensus, is less a consensus than a convenient way to masquerade a centralized network as a decentralized one.

Type of consensus	Description
Proof-of-Work (PoW)	Based on a computational challenge
Proof-of-Stake (Pos)	Based on the locking of finance
Delegated Proof-of-Stake (DPoS)	Similar as the precedent but in return of staking the staker obtained voting influence to choose who influence the consensus
Proof-of-Importance	Based on the reputation score over some criteria showing that the validator is important
Proof-of-Capacity	Based on the potential ability to validate
Proof-of-Elaspsed-Time (PoET)	Based on time waited
Proof-of-Authority	Based on reputation relying on the previous ability to validate
Proof-of-Burn	Based on a fiducial sacrifice

Table 1: different consensus proposals

Bitcoin and Ethereum, which are the two most known and used blockchains, use respectively the Proof-of-Work and Proof-of-Stake mechanisms to validate transactions. In the case of Bitcoin, the validator called a miner needs to have a specific calculation machine called ASIC. This machine computes a mathematical challenge whose transactions validation and the integrity of previous validated blocks are the main goals. The first node who finds the solution of the challenge, broadcasts the new block and its solutions to the network. As a counterpart, the validator receives a financial reward.

In the case of Ethereum, the validator who is called a Staker, locks 32 Ethereum which represents at the current price of Ethereum (1314$) 42048 $. Thanks to this amount, he has the right to be randomly chosen to be the creator of the next blocks. When they are not chosen, they have to verify and validate the blocks created by the other Staker. Similarly as bitcoin validators, their actions are remunerated by the network.

3. Validating a block

To understand things more clearly some points need to be precised.
First, how Bitcoin's mathematical challenge works. In every block, the header needs to contain the precedent hash of the last calculated block and a hash target. Then the validator enters news transactions inside the block's body and adds a specific number called a nonce. The computational challenge is to find a hash of the block which is lower than the hash target by tweaking the nonce value. The computational complexity is defined by the protocol every 2016 new blocks which redefine the hash target.

This way of processing blocks, by stocking the hash value of the previous block also protects the integrity of all the precedent blocks and consequently the integrity of the bitcoin value. Indeed, if an

attacker wants to change precedent block value, he needs to recalculate every hash for every block contained between the modified one and the last recent block, and all before another validator finds the new block hash which is impossible.

Another way to attack the protocol would be to win every new mathematical challenge and puting inside the transaction of the attacker's choice. However to win every challenge you would need at least 51% of the entire calculation power workforce, which is also impossible.

Secondly, to understand how the PoS network is securised, we need to understand the structure of its validation. Since the Merge (which is the transition from PoW to PoS for the Ethereum network), the blockchain is divided on one main chain called beacon chain and other side chains called sharding. Shardings are chains where blocks are validated and then blocks are integrated into the beacon chain. For every block creation that is called a slot, a group of 128 validators is selected to validate the block. For every slot, these groups are different, avoiding agreement among them. The selection is made by selecting individuals with the best ratio between the least number of hash calculated in the past and the higher amount staked. If a validator tries to incorpore or validate a malicious transaction or if he disconnects when he is chosen, he is banned and lost its staking.

4. Smart contracts

Smart contracts have become a crucial part of blockchains in the last years. Created by Vitalik Buterin alongside the Ethereum network, smart contracts are programs that are executed by the blockchain's virtual machine (the Ethereum Virtual Machine, also known as the EVM, in the case of Ethereum). These programs can be written, compiled and deployed to the blockchain as long as they compile properly in blockchain-ready bytecode and follow specific constraints on memory and computation.

Smart contracts are written in blockchain-specific programming languages, Ethereum's Solidity being the most famous. While these languages tend to differ from one blockchain to another, they share a strongly-typed, object-oriented and Turing-complete (which basically means they can perform any kind of operation on data) design. Nonetheless, gas fees are a physical limitation to the complexity a contract can have. Since any required computational effort is billed to the end user - the gas payer -, a contract code must be extremely efficient. Because of gas fees, the usage of for and while loops is prohibitive and memory writes and reads should only be performed when necessary.

The main perk of smart contracts is that they are qualified as immutable: once deployed to the blockchain, neither the code can no longer be modified nor interactions with it can be prevented. However, the storage of a contract (the state of variables stored in its memory) can be updated by anyone that is authorized to do so. Smart contracts live in a sandboxed environment, which means. Another feature of contracts is that anyone can read the content of its memory. When it comes to the code itself, it can be retrieved from the blockchain by translating the public bytecode in a more human-readable format.

IV. Privacy-enhancing tools for the blockchain

Because of the aforementioned issues, researchers and developers have scratched their heads to work out ways to enhance privacy on the blockchain. Two types of tools have been specifically crafted for blockchain usages: privacy coins and privacy-enhancing smart contracts.

1. Issues with transparency by design

As previously mentioned, a blockchain is like a database everyone can read into. While this transparency-enhancing principle has many use cases, it is a constraint for users willing to use decentralized applications anonymously. Indeed, a blockchain is public by design and wallet addresses are mere pseudonyms, at most. While wallet addresses alone are nothing more than long, meaningless chains of characters, they can be linked to specific users, may it be against their will. Even though there is a one-to-n relationship between a user and a wallet (i.e. a single user can possess multiple wallets, which reduces the risks of being doxxed), this specificity of the blockchain is a discomfort to some users.

Moreover, governments have kept a watchful eye on the ecosystem for the past few years because of the exponentially increasing amount of assets flowing through the blockchains. The whole industry has kept raising concerns about money laundering, terrorism funding and massive Ponzi schemes. For instance, in France, each digital asset service provider has to pass the AMF's (*Autorité des Marchés Financiers*, the French equivalent of the American SEC) PSAN (*Prestataire de Service sur Actifs Numériques*) qualifications before selling anything to French customers. Among these latter qualifications, service providers such as central exchanges (like Binance or Coinbase) have to impose a KYC (Know-Your-Customer) procedure to their customers. However, centralized exchanges are for most people the only way to purchase cryptocurrencies. Consequently, the majority of today's wallet addresses are linked to specific users, further reducing the pseudonymity of the blockchain.

Besides, the blockchain's transparency is not limited to the transactions' history: anyone can read in any smart contract storage, which makes it impossible to properly store secrets. Therefore, a messaging service that would be based on a public blockchain could not guarantee any form of anonymity to its users since any third party could read the exchanged messages in the transactions' bodies, even if it would require a lot of digging.

2. Privacy coins

In most respects, privacy coins work just like any other cryptocurrency. They use a public blockchain network to record and validate transactions, and owners of the digital currencies can store them in various types of cryptocurrency wallets[1].

However, privacy coins wrap these basic functions in layers of additional security to anonymize the transactions and/or the identities of each wallet holder. A normal cryptocurrency such as Bitcoin or

[1]Source: https://www.fool.com/investing/stock-market/market-sectors/financials/cryptocurrency-stocks/privacy-coins/#:~:text=Some%20cryptocurrencies%20are%20designed%20with,each%20wallet%20on%20the%20blockchain.

Ethereum is easy to track as it moves from one digital wallet to another, and the amount of Bitcoin and Ethereum sitting in these wallets is public knowledge.

Each privacy coin uses a different strategy to anonymize its transaction data and wallet balances. Generally speaking, these strategies involve temporary addresses and splitting up each transaction into a large number of smaller ones. Crucial information about each whole or partial transaction can also be stored in encrypted form instead of a publicly readable ledger.

The most famous privacy coin is Monero (XMR), which is a privacy-enhanced version of Bitcoin. Like this latter coin, Monero's consensus relies on Proof-of-Work and gathers a network of miners. However, transaction validators' follow a Monero-specific algorithm called RandomX. Monero transactions are signed in a complex system known as a ring signature. In essence, every unique transaction comes with the digital signatures of many different signers, only one of whom actually performed the digital transfer. Figuring out which signature belongs to what transaction is too difficult for the best cracking methods known today. Monero also creates temporary addresses for every transaction, which are protected by two different private cryptographic keys. Only the holder of both private keys can see the final target address of a Monero transaction.

The main issue with privacy coins is that users need to buy them to benefit from their privacy-enhancing features. However, many central exchanges - which is the main gateway for people willing to buy cryptocurrencies - have forbidden purchasing and selling such privacy coins in order to comply with regulations. Ironically enough, using privacy coins in the US is not illegal. Besides, the DeFi (decentralized finance) is an actual alternative to circumvent the exchanges' policies since it does not rely on any centralized third-party.

Another issue is the lack of transparency of the privacy-enhancing blockchains. Since the whole transaction anonymization process is done off-chain, users need to trust validators, which are expected to act in good faith and to keep their software updated. Besides, the network's actual decentralization is an ongoing concern for such infrastructures, as there is no guarantee there are enough miners to prevent them from acting maliciously.

3. Privacy-enhancing smart contracts

Privacy-enhancing smart contracts (also known as on-chain mixers), such as Tornado, are an answer to the concerns privacy coins raise. Indeed, smart contracts do not need any third party to be executed: as long as there will be validators on their originating blockchain, they will keep providing the service they were designed to. Besides, their immutability guarantees the non-malleability of the protocol's rules: once deployed, their code cannot be edited. These contracts are generally open-source and their bytecode can be retrieved from any transaction, which helps users verify they contain no backdoor that could put confidentiality, integrity or availability at harm. At last, since their storage is public, anyone can verify the contract behaves correctly.

However, in spite of these multiple advantages, there are still inconvenients users need to take into account before using privacy-enhancing smart contracts. Even if no one can shut down a smart contract - assuming there is no backdoor - authorities still have methods to enforce their policy. Since every transaction on the blockchain is public, nothing prevents a central exchange from banning or freezing accounts interacting with the contracts. In the United States for instance, American citizens

can be convicted for using Tornado starting from August 2022. Besides, American validators (which represents 50% of the Ethereum network) cannot validate incoming or outgoing Tornado transactions without incurring penalties.

Another issue with on-chain mixers is that they are slow. Compared to privacy coins, their throughput is minimal and users need to be patient. Indeed, they encourage users to wait for other users to deposit funds before withdrawing theirs, so that the link between a sender and a receiver remains broken. Besides, because of the small quantity of pools per token, users who want to transfer non-power-of-ten amounts in stealth are to use the service several times. For instance, a sender S who wants to transfer 150 ETH using Tornado will have to use the protocol six times and use two different pools (specifically the 100 ETH and the 10 ETH pools).

4. Secretive browsing

On-chain mixers often recommend users to use privacy-enhancing networks (and respectively their associated web client) such as Tor (respectively the Tor browser). Note that it is a sheer recommendation and that an on-chain mixer cannot prevent users from using traditional browsers. However, not complying to this advice puts at risk the privacy of the mixing because validators can see the IP addresses of the transaction emitters in clear text. Authorities could therefore ask the validators to disclose said IP addresses for law enforcement. Using payable VPNs as proxies is not a sustainable solution either since users cannot know whether their data is kept private or sold.

V. Cryptographic materials

Both solutions we introduced in the previous chapter rely on cryptographic tools, more specifically Zero-Knowledge Proofs, also known as ZKP. In the first section of this chapter, we introduce in plain English the concept of ZKP; the second section aims at describing more thoroughly a peculiar ZKP method called zk-SNARK.

1. Zero-knowledge proofs

In cryptography, a zero-knowledge proof or zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true while the prover avoids conveying any additional information apart from the fact that the statement is indeed true. The essence of zero-knowledge proofs is that it is trivial to prove that one possesses knowledge of certain information by simply revealing it; the challenge is to prove such possession without revealing the information itself or any additional information[2]. Note that ZKP potential applications are not restricted to the blockchain, since proving solvency in zero-knowledge could enhance data privacy. Here are several use cases for ZKP:

- An individual could prove to a bank they are solvent enough to get a credit without disclosing their actual capital;

[2] Source: https://en.wikipedia.org/wiki/Zero-knowledge_proof

- An individual could prove to an online gambling platform they are at least eighteen without revealing their real age;
- An individual could pay the right amount of taxes without yielding their revenue;
- The US government could prove they do not mention aliens in their classified documents without disclosing their content.

An illustration of a zero-knowledge protocol is the Ali Baba cave experiment, first published in 1990 by Jean-Jacques Quisquater and others in their paper "How to Explain Zero-Knowledge Protocols to Your Children". It is common practice to label the two parties in a zero-knowledge proof as Peggy (the prover of the statement) and Victor (the verifier of the statement).

In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a ring, with the entrance on one side and the magic door blocking the opposite side. Victor wants to know whether Peggy knows the secret word; but Peggy, being a very private person, does not want to reveal her knowledge (the secret word) to Victor or to reveal the fact of her knowledge to the world in general.

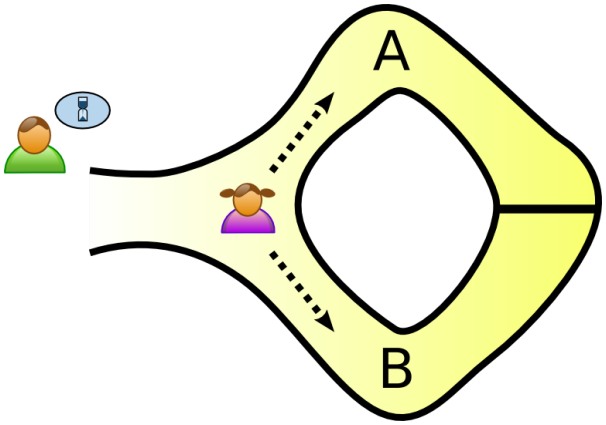

Fig: Peggy randomly takes either path A or B, while Victor waits outside.

They label the left and right paths from the entrance A and B. First, Victor waits outside the cave as Peggy goes in. Peggy takes either path A or B; Victor is not allowed to see which path she takes. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Provided she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path.

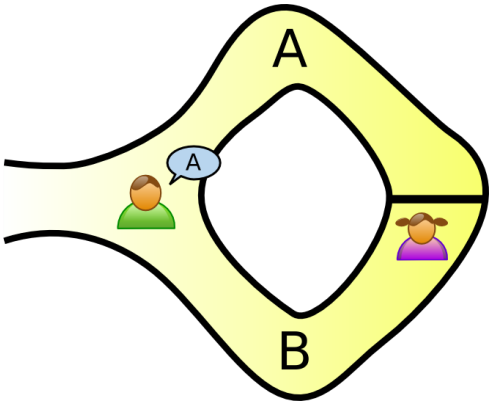

Fig: Victor chooses an exit path.

However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path by which she had entered. Since Victor would choose A or B at random, she would have a 50% chance of guessing correctly. If they were to repeat this trick many times, say 20 times in a row, her chance of successfully anticipating all of Victor's requests would become very small (1 in 220, or very roughly 1 in a million).

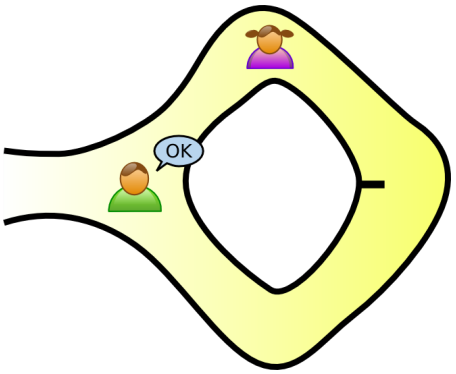

Fig: Peggy reliably appears at the exit Victor names

Thus, if Peggy repeatedly appears at the exit Victor names, he can conclude that it is extremely probable that Peggy does, in fact, know the secret word.

2. zk-SNARKs

Within the first part of this subsection, we explain why Zero-Knowledge Succinct Non-interactive ARgument of Knowledge - most of the time referred to as zk-SNARK, fortunately enough - are particularly convenient for blockchain usages compared to other zero-knowledge protocols. In the second, we break down the whole proving-verifying method.

a. Reason of being

Before anything else, we need to point out that the scope of zk-SNARKs is not restricted to sole transactions. Indeed, companies such as Aleo[3] allow their customers to obfuscate the source code of their smart contracts - and subsequently their business logic - using zk-SNARKs. However, zk-SNARKs are extremely convenient for blockchain needs, since their succinct and non-interactive properties guarantee minimal computation needs compared to non-succinct or interactive solutions. Besides, the main perk of zk-SNARKs compared to pair-based cryptography protocols such as Diffie-Hellman is that no secret needs to be stored publicly contract-side.

As mentioned in the previous chapters, each computation time performed on the blockchain needs to be paid for, as low as it may be. Besides, updating a contract's storage implies the agreement of the whole validating network, which itself is a relatively long process compared to updating data in a single trusted server. This is why we need as few communications as possible between a prover (an end-user) and a verifier (the smart contract) and an end-user's proof that is easy to verify (since verification is performed contract-side).

[3] https://www.aleo.org/

1. Non-interactivity

In an interactive ZKP protocol, verifying a proof takes multiple rounds of interactions between a prover and verifier. It is useful when there is a single verifier such as a compliance auditor but not in a blockchain environment where there are many validators. Using zk-SNARKs, the interaction between the prover and the verifier is simulated by the prover, which makes direct communications with the verifier unnecessary. There is thus a single message (the actual proof) that is sent to the verifier.

2. Succinctness

To minimize computation time - and thus gas fees -, we need the prover to generate a proof that is both short and fast to verify. A more formal definition of succinctness is given later in this chapter.

b. Concept

1. Arithmetic circuits

Computing zk-SNARK proofs relies on arithmetic circuits. In computational complexity theory, arithmetic circuits are the standard model for computing polynomials. Informally, an arithmetic circuit takes as inputs either variables or numbers, and is allowed to either add or multiply two expressions it has already computed. The size of an arithmetic circuit evaluates to the number of logical gates it contains. Arithmetic circuits provide a formal way to understand the complexity of computing polynomials[4].

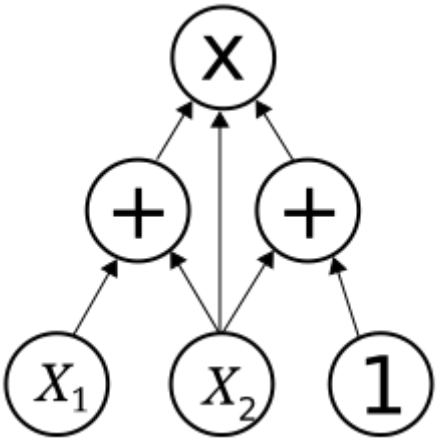

Fig: a simple arithmetic circuit C

In the above example, the computed polynomial P equals X2(X1 + X2)(X2 + 1). We can also state that the size |C| of this circuit C is 3 since there are three logical gates (two sums and a multiplier).

Of course, there are more interesting arithmetic circuits than this dummy example: for instance, the hash circuit $C_{hash}(h, m) = (h - SHA256(m)) = 0$ if SHA256(m) = h and not 0 otherwise. The size of this circuit $|C_{hash}|$ evaluates to approximately 20,000 gates, which represents a medium-sized specimen. $C_{sig}(pk, m, S)$ outputs 0 if S is a valid ECDSA signature on m with respect to the public key pk.

[4] https://en.wikipedia.org/wiki/Arithmetic_circuit_complexity

2. Argument system (interactive)

Now that we know what an arithmetic circuit is, we can focus on argument systems. In the case of an interactive ZKP protocol, we have a public arithmetic circuit C(x,w), where x is a public statement and w a secret witness. For a prover P possessing (x, w), the goal is to "convince" a verifier V - which only possesses x - that there exists a witness w such that C(x, w) = 0. V accepts or rejects the proof depending on its validity. Since P could correctly guess w such that C(x, w) = 0 by chance, V sends multiple challenges to P to minimize the odds of P being simply lucky.

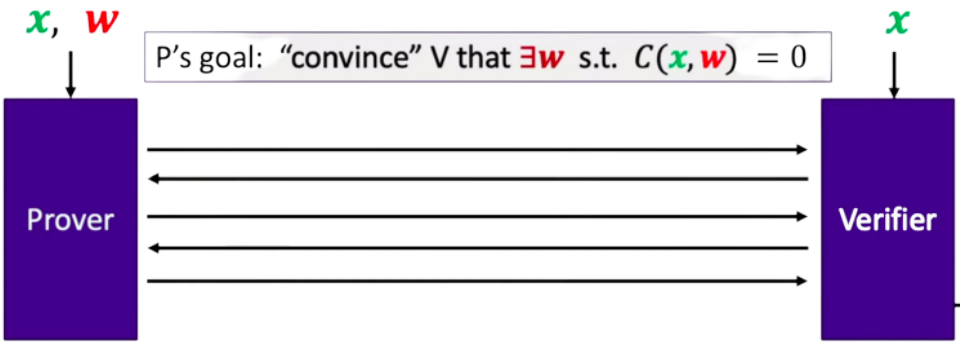

Fig: interactions between a prover and a verifier[5]

3. Preprocessing argument system (non-interactive)

The previous scenario does not quite meet the blockchain requirements of non-interactiveness. In the non-interactive preprocessing argument system, there is a preprocessing setup in which we preprocess the circuit C with an algorithm S.

There are actually three types of setup: trusted setup per circuit (S(C) uses data that must be kept secret), trusted but universal (updatable) setup (secrets in S(C) are independent of C) and transparent setup S(C), in which no secret data is used. The disadvantage of a trusted setup per circuit is that if the trusted setup is compromised i.e. data is intercepted by an eavesdropper, the latter can prove false statements. In an on-chain mixer context, such an eavesdropper could empty a pool redeeming funds that are not theirs. However, a trusted but universal setup consists of a single trusted setup and then performs deterministic circuit-setups without any secret data. Obviously, the transparent setup is the ultimate setup form since there is literally nothing to compromise.

Ironically enough, most on-chain mixers use trusted setup per circuit and more specifically the 2016 Groth's (also known as Groth16) trusted setup, because of its linear prover time, constant size proof and verifier time (both $O(1)$). Besides, it works perfectly fine as long as the setup ceremony is done correctly. Sonic19, Marlin19 and Plonk19 are examples of universal trusted setups. Dark19, Halo19 and STARK are transparent setup examples, although their inherent proofs are not so short both in length and computation time.

	Size of proof π	**Size of Sp**	**Verifier time**	**Trusted setup**
Groth16	O(1)	O(|C|)	O(1)	Yes/per circuit

[5] https://www.youtube.com/watch?v=gcKCW7CNu_M&t=217s

Plonk/Marlin	O(1)	O(|C|)	O(1)	Yes/universal
Bulletproofs	O(log(|C|))	O(1)	O(|C|)	No
STARK	O(log(|C|))	O(1)	O(log(|C|))	No
DARK	O(log(|C|))	O(1)	O(log(|C|))	No

Table: types of SNARKs and comparisons

S(C) outputs public parameters (Sp, Sv), the former parameter being for the prover P and the latter for the verifier V. In this scenario, P possesses (Sp, x, w) while V has got (Sv, x). After this preprocessing phase, P sends the proof π to V, which accepts or rejects it.

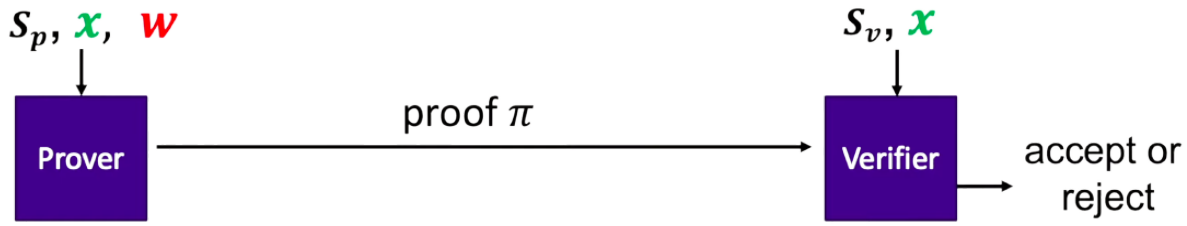

Fig: interactions between a prover and a verifier

To summarize, a non-interactive argument system is a triple (S, P, V) comprising:

- The preprocessed circuit S(C), outputting public parameters (Sp, Sv);
- The prover P(Sp, x, w) who produces the proof π;
- The verifier V(Sv, x, π) who accepts or rejects the proof π.

An argument system (S, P, V), may it be interactive or non-interactive, needs to verify several properties:

- Completeness: for any (x, w), having C(x, w) = 0 implies that the probability of having V accepting the proof π equals 1;
- Argument of knowledge: having V accepting the proof π implies that P "knows" w such that C(x, w) = 0 (the probability of V accepting the proof π from a malicious P* that does not know w is negligible in polynomial time);
- Zero-knowledge: the triple (Sv, x, π) does not reveal anything about w. This property is optional.

Moreover, a succinct non-interactive argument system requires P to generate a short proof and the verifier to accept or reject the proof quickly. More formally, a short proof π is a proof which length |π| is only logarithmic in the size of a circuit C. For instance, if C contains a million gates, the proof's length equals six. When it comes to the fast-to-verify property, it means that the time to verify a proof π can be linear in the size of the statement x (V needs at least to read the statement x that is being verified) and at most logarithmic in the size of a circuit C.

However, since the verifier V needs to be extremely fast, it does not have the time to read the whole circuit C. This is precisely why it was needed to preprocess it. Sv, the verifier-fed public parameter computed by the preprocessing algorithm S, actually contains a short summary of C. If an argument system (S, P, V) is both succinct and zero-knowledge, then it can be called a zk-SNARK.

4. The trivial argument system

At this point, it may be normal to wonder why P and V would require such a complex protocol. Indeed, a naive version of the proving-verifying protocol consisting in two simple steps could be implemented:

- P sends w to V;
- V checks if C(x, w) = 0 and accepts if so.

However, w might be secret and P does not want V to know the secret witness w. To put things into perspective, an end-user using an on-chain mixer does not want the smart contract to know which tokens are being redeemed. Furthermore, w might be long and we want a short proof. Finally, computing C(x, w) may be computation-intensive, which is to be avoided in a blockchain context.

VI. The case of Tornado.cash

1. Generalities

Tornado.cash is an open source Ethereum-based protocol which can be classified as a privacy-enhancing smart contract. Indeed, Tornado claims to provide non-custodial private transactions to Ethereum users.

It was developed and released in August 2019 by Roman Semenov, Alexey Pertsev, Roman Storm. As mentioned in their white paper, Tornado.Cash implements an Ethereum zero-knowledge privacy solution: a smart contract that accepts transactions in Ether (and other ERC-20 tokens) so that the amount can be later withdrawn with no reference to the original transaction. To break the bond between sending and receiving wallets, the protocol uses zk-SNARKs, with off-chain provers (the users of the protocol) and an on-chain verifier (the contract).

2. Overview of the protocol

Using Tornado comes in two phases: depositing funds in a pool and withdrawing them later. A pool is a smart contract storing funds and proofs. There is a dedicated pool for each deposit of N token, N being a power of ten integer[6]. This constraint is a way to obfuscate the link between the sender and the receiver: for instance, a sender S and a receiver R could easily be linked together if S deposited 123.45 ETH and R withdrew exactly the same amount. Users are also encouraged to delay their withdrawal after a deposit; the delay directly depends on the quantity of traffic flowing through the protocol.

[6] In Tornado's beta version, there are four pools for each ERC-20 token T: a 0.1 T, a 1 T, a 10 T and a 100 T pool.

Finally, a redeeming wallet can claim previously deposited tokens without disclosing which. One could wonder why zero-knowledge proofs are needed to anonymize transactions since users could simply deposit funds in a single pool and withdraw them from it later: because of the fungible property of ERC-20 tokens, an observer could not tell who a retrieved token belonged to in the first place. However, such an implementation can neither prevent double-spendings nor ensure users withdraw the funds they are entitled to, at least without publicly disclosing identifying data. Since we cannot store a list of withdrawals and token recipients, the non-identifying, proving-verifying framework offered by zk-SNARKs comes in handy.

a. Depositing

Let a sender S and a receiver R two individuals that use Tornado to exchange funds without revealing they interacted with each other. Let N be an integer representing the amount of tokens S wants to send to R. To deposit these funds, S needs to interact with the N pool. Before depositing the N tokens in the pool, S generates a secret which is called a deposit note. Once the deposit note has been generated and carefully saved by S, S sends the hash of the note h and the N tokens to the pool. Each hashed deposit note that is sent to the contract's storage is called a coin.

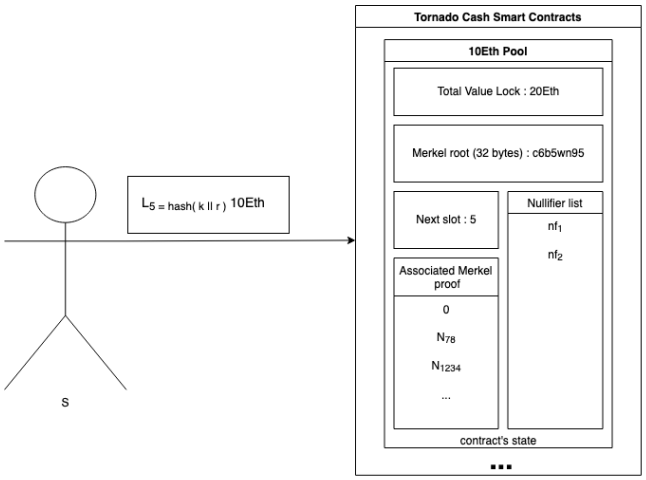

Fig: sender S sends 10 ETH to the 10 ETH pool contract along with the hash of his deposit note.

The deposit note is the concatenation of (k, r), two random numbers respectively called the nullifier and the random. These random numbers are computed client-side, in the web browser of S, so that they are not sent over the network. Indeed, contrary to its hash which is sent to the pool contract, the deposit note must absolutely remain secret.

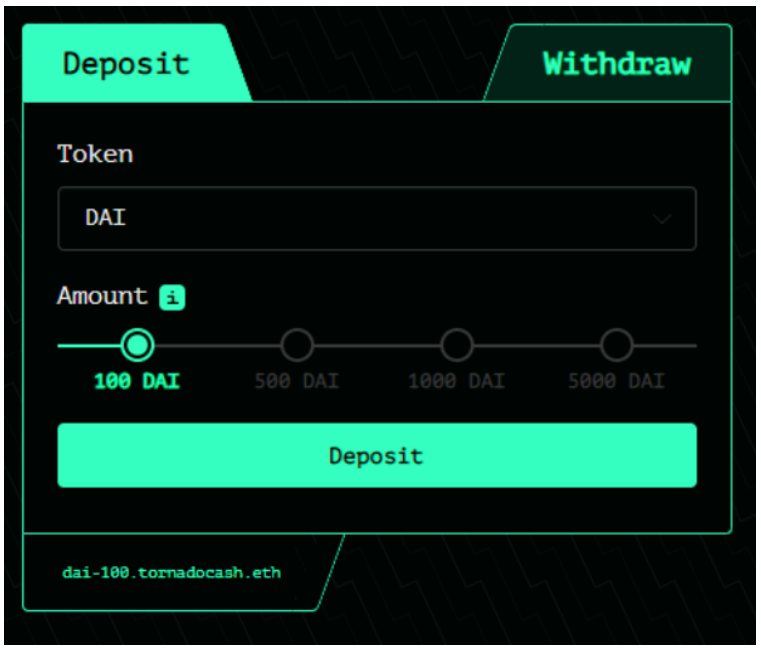

Fig: Tornado's user interface to deposit 100 DAI

b. Withdrawing

Let's assume the sender S and the receiver R have exchanged the secret (k, r) using a confidential setup. Besides, let's assume R has waited for other funds to be deposited in the N pool before redeeming the funds. Tornado's user interface provides some insights on the protocol's statistics to enforce this cautious behavior.

Fig: Tornado's statistics for the 100 DAI pool

When R wants to withdraw the N funds deposited by S, the pool contract expects to be provided a wallet address A to send the funds to and a proof P that the N funds have effectively been deposited in the contract. However, neither the hash h nor the clear text of the secret (k, r) can constitute valid proofs, since the hash is public (and could be provided by anybody reading the contract's storage) and the secret has to remain secret, which otherwise would link S and R or would allow any user to

front-run the transaction with the correct argument and steal the funds. Thereby, the proof P is computed using a zk-SNARK protocol (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) off-chain (using the proof-generating snark.js library[7]) while its verification is done by the pool contract (thanks to a Solidity implementation of a proof verifier[8]).

To prevent double spending, R sends to the contract the hash of the nullifier h(k) which will be stored in the contract. Thereby, before each withdrawal, the contract checks whether a specific coin has already been redeemed or not. R also sends the Merkle proof (a clear definition of Merkle proof is provided in the next chapter) to prove the coin he's willing to redeem has actually been deposited.

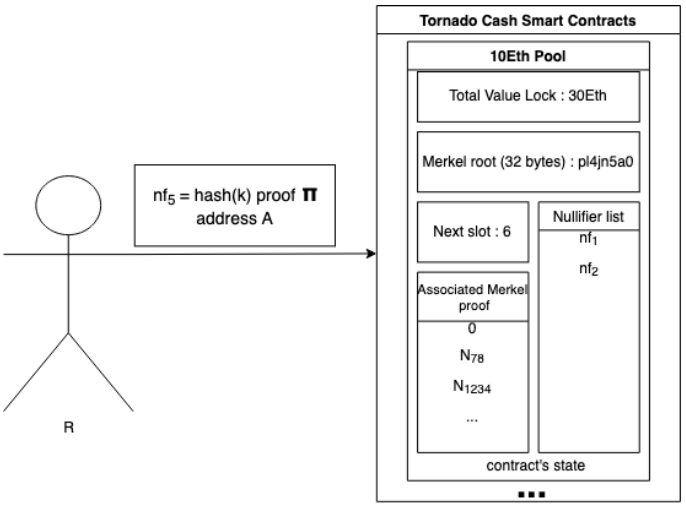

Fig: receiver R redeems funds from the 10 ETH pool contract providing the hash of the nullifier nf_5, the Merkle proof associated with the coin he's redeeming and an address A to send the funds to.

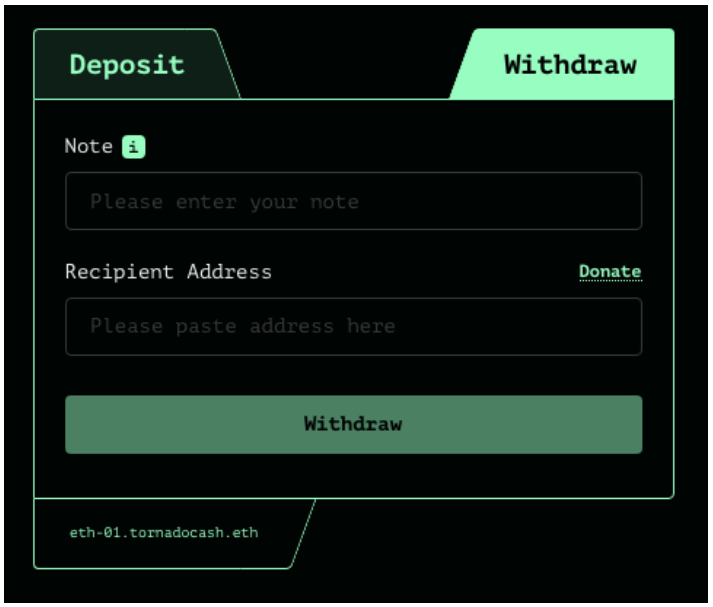

Fig: Tornado's withdrawal user interface. Note that the 100 DAI amount does not need to be mentioned.

[7] https://github.com/iden3/snarkjs
[8] https://github.com/tornadocash/tornado-core/blob/master/contracts/Verifier.sol

c. Relayers

The main problem with the withdrawal process is the transaction fees payment. Indeed, when receiver R interacts with the contract to redeem the sender's N tokens, he needs to pay for the gas the contract will consume to verify the proof P. Actually, any kind of interaction, as lightweight as it may be in terms of computation, requires gas. However, paying for gas fees implies for the receiver R to buy tokens, which is hard to do without revealing his identity (buying tokens on a central exchange requires passing a KYC procedure) or being linked with the sender S. We cannot ask the smart contract to pay for the transaction fees either, simply because a smart contract cannot pay for its own execution.

To tackle this issue, Tornado has created an off-chain relayer network. A relayer is a wallet address that calls the pool contract and pays for the transaction fees F on behalf of the receiver R. As a reward, the relayer takes a fee $f \geq F$ for each withdrawal. The amount (N - f) is then sent to R.

3. Security claims

According to Tornado's white paper, the overall security properties of the protocol are the following:

- Only coins deposited into the contract can be withdrawn;
- No coin can be withdrawn twice;
- Any coin can be withdrawn once if its parameters (k, r) are known unless a coin with the same k has been already deposited and withdrawn;
- If k or r is unknown, a coin can not be withdrawn. If k is unknown to the attacker, he can not prevent the one who knows (k, r) from withdrawing the coin (this includes all cases of front-running a transaction);
- The proof is binding: one can not use the same proof with a different nullifier hash, another recipient address, or a new fee amount;
- The cryptographic primitives used by Tornado have at least 126-bit security (except for the BN254 curve where the discrete logarithm problem has something like 100-bit security), and the security does not degrade because of their composition;
- The withdrawal procedure of a given coin does not reveal which coin is being redeemed.

4. Merkle tree architecture

Tornado leverages the Merkle tree data structure for the needs of their proving-verifying mechanism. In cryptography and computer science, a hash tree or Merkle tree is a tree in which every "leaf" (node) is labeled with the cryptographic hash of a data block, and every node that is not a leaf (called a branch, inner node, or inode) is labeled with the cryptographic hash of the labels of its child nodes. A hash tree allows efficient and secure verification of the contents of a large data structure. A hash tree is a generalization of a hash list and a hash chain.

Demonstrating that a leaf node is a part of a given binary hash tree requires computing a number of hashes proportional to the logarithm of the number of leaf nodes in the tree. Conversely, in a hash list, the number is proportional to the number of leaf nodes itself. A Merkle tree is therefore an efficient

example of a cryptographic commitment scheme, in which the root of the tree is seen as a commitment and leaf nodes may be revealed and proven to be part of the original commitment[9].

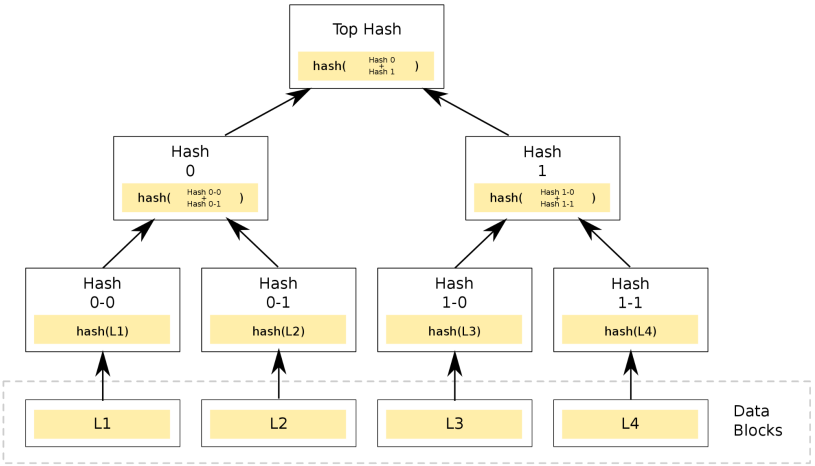

Fig: An example of a binary hash tree. Hashes 0-0 and 0-1 are the hash values of data blocks L1 and L2, respectively, and hash 0 is the hash of the concatenation of hashes 0-0 and 0-1.

Tornado does not store the whole Merkle tree in their pool contracts, instead it only keeps track of the current leaves' values (an array that is appended after each deposit) and the current root's value, which is overwritten after each deposit. Since each pool contract computes a 20-level Merkle tree to store coins (i.e. the hash of deposit notes), which means a pool contract can process 1048576 (2^20) deposits before becoming obsolete. Leaves cannot either be deleted without exposing which coins have been redeemed.

Besides, a pool contract stores other variables, such as:

- The next slot (the leaf index in which the hash of the next deposit note should be written);
- A nullifier[10] list to prevent double spending;
- A Merkle proofs[11] array, each Merkle proof corresponding to a deposit;

[9] Source: https://en.wikipedia.org/wiki/Merkle_tree

[10] As previously mentioned, a sender S generates a deposit note consisting of (k, r), two random numbers called respectively nullifier and random, right before depositing. Storing the nullifier in the contract's storage is a way to know if a coin has already been redeemed using a specific deposit note without revealing neither the secret nor which coin is redeemed.

[11] A Merkle proof is nothing more than an array consisting of the nodes hashes from the ground up to the root. A Merkle proof is unique because it relies on a specific leaf's value.

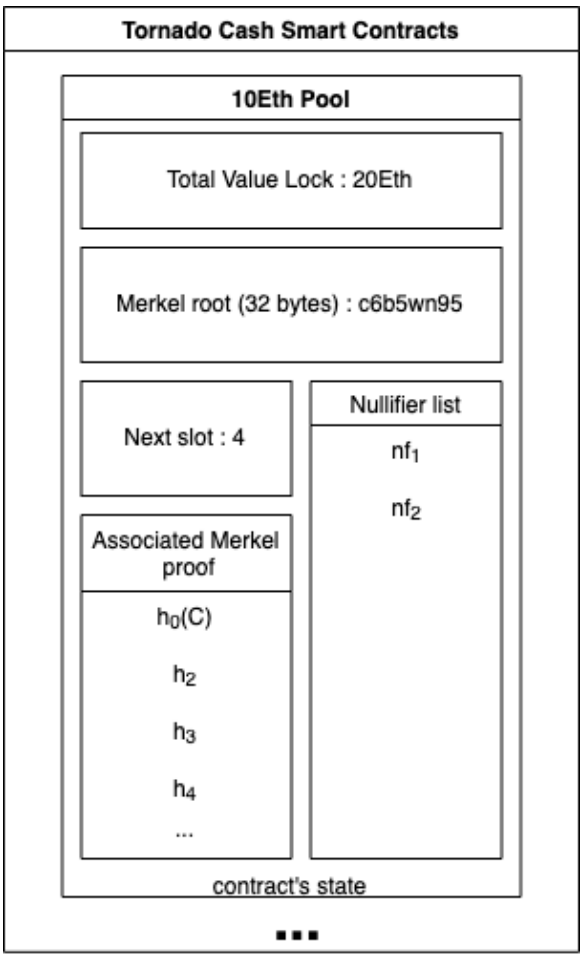

Fig: representation of a pool contract's storage

5. Drawbacks

More than privacy, several aspects of Tornado's beta version undermine the user experience. Since most on-chain mixers are Tornado's doppelgängers - based on other blockchains than Ethereum, the identified drawbacks are not Tornado-unique.

a. Issues with current implementations

Usage-based obfuscation

One of the main drawbacks with on-chain mixers is their low throughput, which itself is due to their overall lack of usage. Since mixing transactions remains a niche practice in the blockchain ecosystem, there are few interactions with the contracts - especially transactions with high-amount pools. However, the mixers' guidelines advise users to space their deposits and withdrawals to obfuscate the bond between a sending and a redeeming wallet. Indeed, the fewer the transactions in between, the easier it is for observers to understand the link between two wallets: if each user redeemed their tokens right after their deposit, there would be no point in using zk-SNARKs because of time correlations an external observer could make. For instance, let a protocol that is used three times a year in total; a depositing wallet could easily be linked to a redeeming one given they both interact

with the contract in a two-minute time window. This means for users that an effective on-chain mixing may be an extremely long process depending on the protocol's usage.

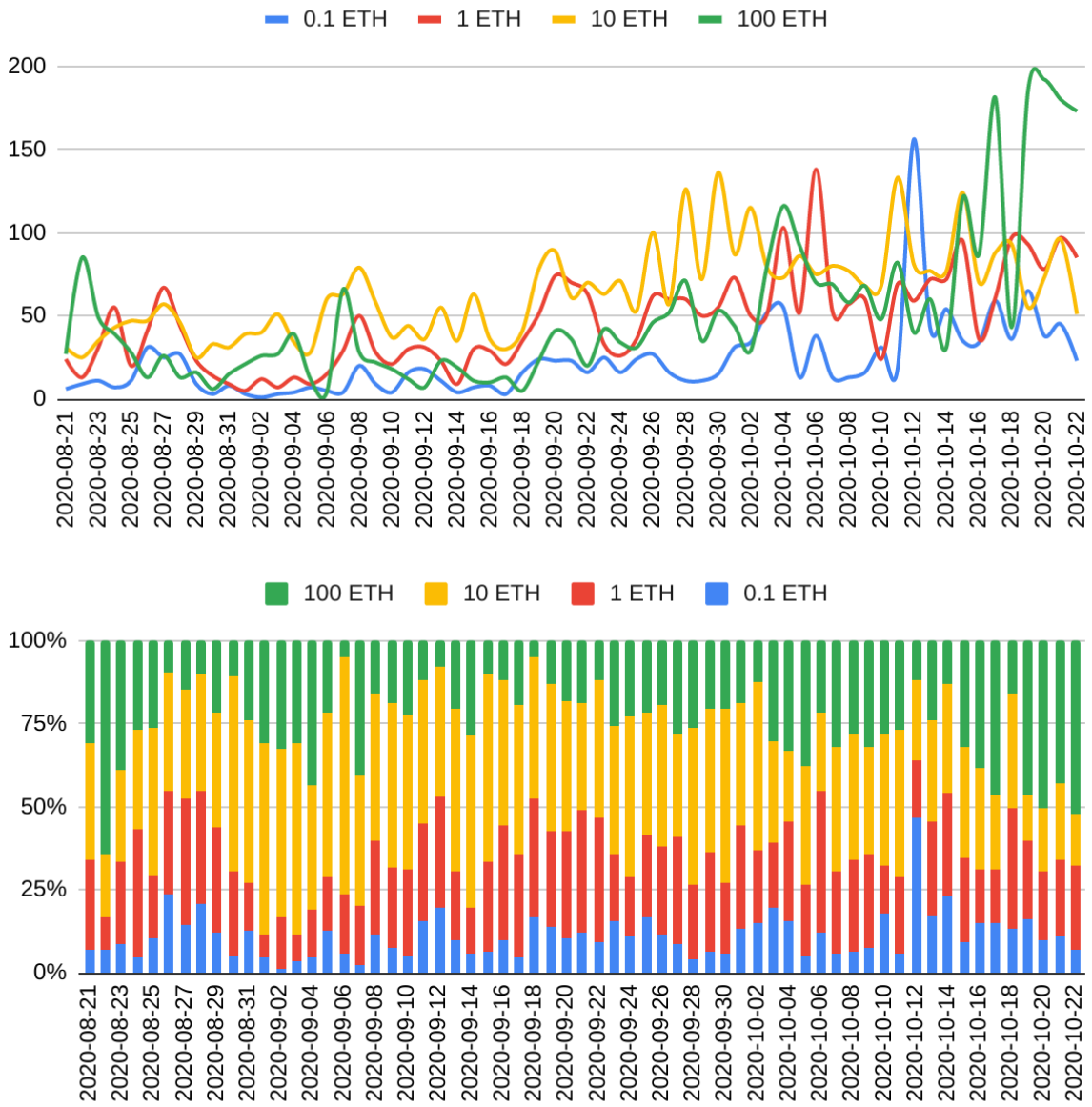

Fig. a & b: pool usage between 2020, August the 21st and October the 22nd

The above graphs show that the different pools are not used equally.

Fixed deposit and withdrawal amounts

Another disadvantage with Tornado's design is that users can only deposit and withdraw fixed amounts. For instance, let a protocol having three different pools to mix respectively 0,1 token, 1 token and 10 tokens. A user willing to mix 15,3 tokens using this protocol will have to go through the deposit-withdrawal process nine times in total (one time to mix 10 tokens, five times to mix 5 tokens and 3 times to mix the 0,3 tokens remainder).

In terms of privacy, this is a great mechanism to prevent users from transferring identifying amounts: if the user deposited 15,3 tokens and withdrew the same amount in a single process, it would be

extremely easy for an external observer to correlate the depositing and withdrawing wallets. However, due to the aforementioned usage-based obfuscation and the logarithmic relationship between pool-authorized amounts, mixing such an "exotic" amount could take a tremendous amount of time, especially when using the least active pools. In late 2021[12] Tornado introduced the alpha version of Nova which is a pool-free version of the protocol allowing users to deposit arbitrary amounts. Nonetheless, the protocol's guidelines encourage withdrawers to redeem fixed amounts ranging from 0.1 ETH to 1 ETH instead of their original deposit.

One-to-one relationship between sending and receiving wallets

With Tornado's current implementation, users need to split their deposit in separate transactions if they want to send their funds to different recipient addresses. Indeed, users could be tempted to retrieve funds using several unrelated stealth wallets to enhance privacy. There are two issues with this mechanism: the first is that the sender and the receiver need to agree beforehand on the number N of withdrawing wallets so that the sender splits the total amount in N deposits. The second issue is that splitting funds into separate deposits is a financially costly operation for both depositing and withdrawing parties.

b. Possible mitigations

Transaction fuzzing

A method to solve the aforementioned issues with usage-based obfuscation consists in flooding the protocol with mixing transactions to artificially increase traffic. A high throughput would significantly shorten the mixing process for users, since they would not have to wait for other users to transfer their funds. Since we do not want this mechanism to rely on human interactions, the perfect candidate for transaction fuzzing is a bots swarm. However, bots need funds to both mix and pay for the associated gas fees. This concern could be answered by financially incentivizing users to lend funds to the fuzzing bots, which would increase the protocol's popularity at the same time. These users are referred to as liquidity providers (or LPs) in the following sections.

At first sight, this method raises a concern: how could liquidity providers be rewarded so that an external observer cannot make a difference between artificial and genuine transactions? Indeed, the whole fuzzing mechanism would be pointless if we could see who is being paid for flooding the protocol with transactions. An easy way to address this issue consists in hiding the relationship between a liquidity provider's wallet and the fuzzing bots'. While the liquidity providers' wallets list would be stored on-chain, the fuzzing wallets - created on-the-go by the bots - would behave exactly like actual users'.

Besides, it should be ensured bots do not send transactions homogeneously across time, otherwise an external observer could spot a time-based pattern and identify whether a transaction is made by a bot or an actual user. Instead, bots should fuzz transactions randomly. While it is not possible to trigger a pub/sub scheduled function randomly, a convenient way to mimic randomness could consist in having

[12]Source:
https://tornado-cash.medium.com/tornado-cash-introduces-arbitrary-amounts-shielded-transfers-8df92d93c37c

a predicate evaluating the value of a pseudo-random number (which is either 0 or 1). If the random number equals one, the bot sends a transaction to the contract, otherwise it does nothing.

Batched withdrawals

Another method to mitigate time-correlation risks consists in batching withdrawals together. Instead of having users withdraw their funds one by one when they want - which can lead users to identify themselves if they do not comply with the guidelines -, funds could be withdrawn altogether by fixed-size transaction packs. This method should be implemented in addition to transaction fuzzing to avoid withdrawers waiting for too long. Once again, protocol users could be financially incentivized to release transactions once a pack is full.

The main issue with batched withdrawals consists in finding the proper way to store the memory pool (the pending withdrawals waiting zone). Such a queue cannot be on-chain: because we do not want withdrawers to reveal their stealth addresses before they are sent their funds, they cannot interact publicly with a smart contract. This memory pool should thus be stored off-chain, which means users should trust the storing entity.

Multiple recipient addresses

To withdraw funds, a user needs to provide a receiving address which lies in the public statement of the zk-SNARK proof. Embedding this address within the proof ensures non-malleability i.e. preventing the recipient address from being changed, funds cannot be diverted. Instead of providing a single address in the public statement, users could pass an array containing different multiple wallet addresses and ensure non-malleability on this set of addresses. This feature would make optional the one-to-one relationship between sending and receiving wallets. However, the feasibility of such an improvement is yet to be assessed. Maelstrom.fi, a Tezos-based on-chain mixer, has a feature to split withdrawals into separate transactions. However, one of the core team developers has stated that such a functionality could not be achieved using zk-SNARKs[13].

[13] Source: https://www.reddit.com/r/tezos/comments/ln87b9/comment/gw0si9o/?utm_source=share&utm_medium=web2x&context=3