Fairness-Privacy Issue in Adaptation of ZEXE Protocol for Exchange Between Untrusted Parties

Victor Languille, David Menga, Gerard Memmi

Fairness-Privacy Issue in Adaptation of ZEXE Protocol for Exchange Between Untrusted Parties

Victor Languille[1,2], David Menga[1], and Gerard Memmi[2]

[1] EDF R&D Saclay
[2] LTCI, Telecom-Paris, Institut polytechnique de Paris

Abstract. We present the main features of ZEXE, a privacy-preserving system extending the Zerocash protocol. Then, through a realistic use case, we show how a slight improvement can make this system more effective. We point out some of its potential vulnerability in term of anonymity followed by methods to overcome them.
Keywords : Blockchain, Privacy, Distributed Ledger Technology (DLT), Zcash, ZEXE, ZKP, NFT.

1 Introduction

The Bitcoin protocol was designed circa 2008 [Nak08] aiming at creating a currency functioning without any central authority. Bitcoin has been intentionally designed in such a way that anyone accessing the underlying public ledger gets knowledge of the content and the whole history of transactions of any participant. Bitcoin is known to be pseudonymous and this is a major drawback with regard to privacy pointed out in many papers (see [KYMM18] for a list of citations). Addressing privacy issues is one of the key recommendations made to the French Administration in [DDJ+21] and is at the core motivation of this research.

The Zerocash protocol was proposed in 2014 [BSCG+14], in order to remediate the lack of privacy in Bitcoin, while also functioning on a decentralized and publicly accessible ledger (known as *Distributed Ledger Technology* (DLT)). Anonymity in Zcash was achieved through a notion of "shielded pool", a mechanism analyzed in [KYMM18]. This protocol recently improved in [HBHW22] is now used at the core of the privacy-preserving cryptocurrency Zcash.

During the same period, the launch of Ethereum blockchain in 2015 [3] integrated the notion of smartcontract [4], a kind of program executed on the decentralised ledger widely generalizing simple money transfers.

In 2018, the ZEXE protocol [BCG+20], combining both innovations, was proposed to build privacy-preserving systems for execution of programs on a public

[3] https://ethereum.org/en/whitepaper/
[4] Notion introduced by N. Szabo circa 1994 [SJZG18].

ledger, hiding not only the program's inputs, but also the kind of computation which constitutes them.

The ZEXE protocol relies on a system of records based on Non-interactive Zero-knowledge Proofs [GMR89]. Roughly speaking, a Zero-Knowledge Proof is a cryptographic tool allowing a Prover to prove to a Verifier, for a given program P and a given output y, that he knows an input x such that $\mathsf{P}(\mathsf{x}) = \mathsf{y}$, without revealing anything about x. Furthermore, it is expected that the verification be performed much more quickly than the computation of P.

1.1 Outline

In this report, we first briefly present the ZEXE protocol and its main features Section 2. Then in Section 3, through the use-case of purchase and sale agreement, we explain how an alteration of the original protocol can overcome some security issues regarding the $weak - liveness$ property. Finally Section 4, we point out some privacy issue that can arise in case where multi-party computation techniques are utilized together with the ZEXE protocol. We conclude by calling for a new specific cryptographic primitive allowing mutually distrustful parties to share an obfuscated append-only list guarantying no repetition.

2 ZEXE

In this section, we present the functioning of ZEXE and a payment scheme built from it that we will use in the following sections. We tried to make it as concise as possible, sometimes at the cost of completeness. For a more in-depth presentation we refer to the original ZEXE paper, which we quote freely and from which we use the figures.

2.1 General Presentation

The protocol Zerocash is not just adding a privacy feature to the bitcoin protocol but works in a fundamentally different way from the bitcoin one. To make a loose analogy, bitcoin works like a digital currency where each user has an account and is able to send and receive tokens with other users' accounts through public transactions. Zerocash works more like a fungible money, where each user owns several real coins or bills and exchanges them in a potentially anonymous manner.

The ZEXE protocol widely extends the Zerocash protocol, building a *decentralised private computation* (DCP) scheme. Such scheme allows different mutually distrustful users to share an append-only ledger used to attest the execution of offline computations in a private way. In such a context, participants individually hold some (partial) states of processes and can update them and transmit their ownership following some specified rules. The different processes are themselves private in the sense that the shared ledger doesn't allow observers telling

what process is performed. Moreover, interaction between different processes is permitted if the rule specifying the allowed state's transitions of all involved processes are compatible.

The Zerocash protocol can then be considered as a particular case of the ZEXE protocol where the process is publicly known: its state corresponds to a certain distribution of coins between users, and its transition rules must check the balance between created and consumed coins.

2.2 ZEXE Functioning

Briefly, the basic idea is that the ownership of a (partial) state of a process by a user u is represented on the ledger by a commitment biding together a public address associated to u and the representation of this state. The transition from one state to another one, potentially owned by a different user, is realised by revealing a function of those information and the private address of u, with a zero-knowledge proof ensuring that those information indeed corresponds to the previous appended commitment, while keeping them private. This scheme prevents the ownership of a state to be used twice to produce another one ; in particular, this solves the double spending issue when ZEXE is used as Zerocash (i.e. as a simple payment scheme).

The specification of the rules allowing to pass from one state to another will be given by a couple of Boolean valued functions: a *birth and a death predicates*, associated to any state, and specifying the rule by which it can be created from other ones and serve for the creation of them.

In addition to a trusted append-only public ledger, which can typically be realised as blockchain or any other kind of distributed ledger, ZEXE protocol needs also *cryptographic building blocks* and associated *public parameters*, which have to be specified and trusted.

Cryptographic Building Blocks and **Public Parameters**:

- *Security parameter* λ, determined by the desired level of security and the supposed computational power of a potential attacker, and on which depends implicitly the characteristics of the three following primitives.
- *Pseudo-Random Function* (PRF) *families* $\mathsf{PRF}^{\mathsf{add}}_{\mathsf{x}}()$ and $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{x}}()$, where x denotes a random seed. $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{x}}()$ is required to be collision resistant.
- *Commitment scheme* $\mathsf{COM}_{\mathsf{x}}()$, where x denotes a random seed.
- *Zk-SNARK*[5], protocol in which, given a program P and an output y, a Prover produces a short proof π convincing a Verifier that he knows an input x such that $\mathsf{P}(x) = y$, without revealing any information about x to the Verifier. We will sometimes use the more general term "Zero-Knowledge Proof" instead of the more specific one "*Zk-SNARK*".
- *Common reference string* CRS, public parameter used to compute *Zk-SNARK* proofs and verify them.

[5] Zero-knowledge Succint Non-interactive Argument of Knowledge

We refer to [Gol01] for the precise definition of pseudo-random function families and commitment scheme, and to [BCCT12] for the precise definition of *zk-SNARKs*.

As in Zerocash, and even as in Bitcoin, allowing different users to manage their asset without any necessity to delegate it to a trusted central authority implies that, as trade-off, each user has to manage a kind of secret information linked to his public identity.

Setup: Each user u generates at least one pair of *public/private addresses* $(\mathsf{apk}_u, \mathsf{ask}_u)$.

The *private address* ask_u is generated by the user by sampling a random number whose size depends on the security parameter λ.

The public address apk_u is derived from this private address via a pseudo-random function $\mathsf{PRF}^{\mathsf{add}}$: u computes his public address from his private one applying $\mathsf{apk}_u = \mathsf{PRF}^{\mathsf{add}}_{\mathsf{ask}_u}(0)$ such that no one can retrieve ask_u from the knowledge of apk_u.[6]

The user u provides his public address to any other users likely to interact with him, and he keeps his private address secret.

A user can generate as many addresses as he wants, if he wants to appear under different identities with regard to the different users with whom he interacts or the nature of its interactions.

To a public address of a user, will be bound some units of data representing the states of processes he can manipulate via is private address.

2.3 Records

Ownership of state of processes is embodied by data sets called *records*. Among the data constituting a record, some are private, and some other will appear on the public ledger at some point. Each user u owns records containing certain states of processes and a specification of the rules by which this state have been attained, and the ones by which that it can change.

Change of state and transfer of ownership from a user u to a user u' is done by spending records, thanks to the knowledge of private data of this record and of the private address ask_u, and creating a new record for u', thanks to his public address apk'_u, as we will explain for the next session. For now, let us specify more precisely what are the data constituting a record.

A record $\mathtt{r} = (\mathsf{payload}, \mathsf{apk}_u, \rho, s, \mathsf{cm}, \mathsf{sn})$ consists in the list of the following data:

- A *payload*: $\mathsf{payload}$ representing the state of a certain process
- A *birth predicate*: Φ_{b} which has to be satisfied when the record is created

[6] Note a first difference with Bitcoin here. User doesn't have couple of public/private signature keys, but only a couple of public/private addresses.

- A *death predicate*: Φ_{d} which has to be satisfied when the record is consumed
- The *public address* of its owner u: apk_u
- A *secret random generator* nonce: ρ
- A *randomness number* used in commitment scheme: s
- A *commitment*: cm computed at the creation of the record, and binding together the different information constituting the records
- A *serial number*: sn computed at the consumption of the record, and only computable with the knowledge of the secret address ask_u of the record's owner.

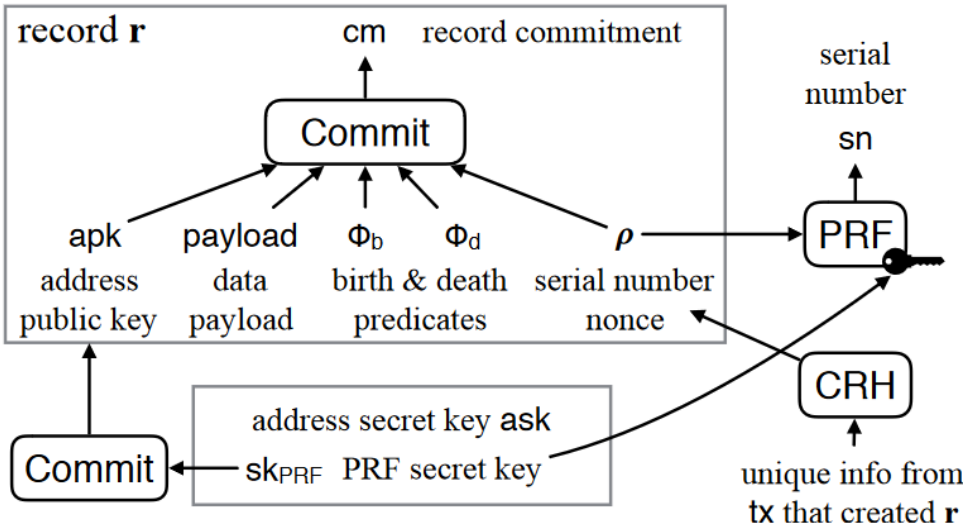

Fig. 1. ZEXE: record commitment constitution, from [BCG+20]

Among these data, only the commitment cm and the serial number sn will appear on the ledger at some point ; payload, owner address, randomness number s and secret generator will be kept private. As mentioned, the commitment cm is added when the record is created, and the serial number sn is added when the record is spent. Thus, the ledger maintains both the list of all added commitments, that we will write ComList, and a list of all added serial numbers SpentList.

For efficiency reasons, the list ComList is maintained as a Merkle tree using a collision resistant hash function. The leaves of the tree are constituted with the added commitments. We note $rt(\mathsf{ComList})$ the root of this Merkle tree. The depth d^{tree} of the Merkle tree is hardcoded and corresponds to the maximum number of records supported by the ledger.

As shown in Figure 1, cm and sn are computed from the private data of the record applying the commitment scheme $\mathsf{COM_x}()$ and the Pseudo-Random Functions $\mathsf{PRF}^{\mathsf{sn}}_x()$ respectively in the following way:

$\mathsf{sn} = \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{ask_u}}(\rho)$

$\mathsf{cm} = \mathsf{COM_s}(\mathsf{payload}||\Phi_\mathsf{b}||\Phi_\mathsf{d}||\mathsf{apk}_u||\rho)$

As we can see, sn and cm of a same record are linked by the fact that they are both computed from the same seed ρ. However, this not directly visible from the reading of the ledger. Moreover, note that computing the commitment cm only requires a public address apk_u, while computing the serial number sn requires the corresponding private address ask_u. This allows one participant u to create a record for another participant u', such that u' will be able to spend it but not u.

Those change of owner's state, as well as the transition between states is done through transactions.

2.4 Transactions

A transaction between m not necessarily distinct (and in practice often identical) senders $[u_i]_1^m$ and n not necessarily distinct (and in practice often identical) receivers $[u'_j]_1^n$ consists in the data list $\mathsf{tx} = ([\mathsf{sn}^{\mathsf{old}}_\mathsf{i}]_1^m, [\mathsf{cm}^{\mathsf{new}}_\mathsf{j}]_1^m, \pi)$ where

- The m serial numbers $\mathsf{sn}^{\mathsf{old}}_\mathsf{i}$ corresponds to m old consumed records $\mathfrak{r}_i^{old}$ owned by u_i
- The n commitments $\mathsf{cm}^{\mathsf{new}}_\mathsf{j}$ corresponds to n new created records $\mathfrak{r}_j^{new}$ owned by u'_j.
- π is a Zero-Knowledge Proof that the records are actually well computed, and that the respective death predicates of each $\mathfrak{r}_i^{old}$ records and birth predicates of each $\mathfrak{r}_j^{new}$ records are verified. Such birth and death predicates are Boolean valued functions taking in argument the transaction's local data constituted by all the records attributes (i.e., $\mathsf{payload}$, Φ_b, Φ_d, apk_u, cm, sn) present in the transaction, and optionally auxiliary inputs.

More precisely, π is a zero-knowledge proof of the NP-statement with the following form:

Instance: $I = (\mathsf{ComList}, [\mathsf{sn}_i^{old}]_1^m, [\mathsf{cm}_j^{new}]_1^n)$

Witness: $([\mathfrak{r}_i^{old}]_1^m, [\mathsf{ask}_{u,i}^{old}]_1^m)$

Statement: 1)Records are well formed, and the spent ones corresponds to previously added ones actually spent by their owners:

$\forall i \in [1, m]$

$\mathsf{cm}^{\mathsf{old}}_\mathsf{i} \in \mathsf{ComList}$

$\wedge\, \mathsf{sn}^{\mathsf{old}}_\mathsf{i} = \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{ask}^{\mathsf{old}}_{\mathsf{u},1}}(\rho_i)$

$\wedge\, \mathsf{cm}^{\mathsf{old}}_\mathsf{i} = \mathsf{COM}_{\mathfrak{r}^{\mathsf{old}}_\mathsf{i}}(\mathsf{payload}_i^{old}||\Phi^{\mathsf{old}}_{\mathsf{b,i}}||\Phi^{\mathsf{old}}_{\mathsf{d,i}}||\mathsf{apk}^{\mathsf{old}}_{\mathsf{u,i}}||\rho_i^{old})$

$\wedge\, \mathsf{apk}_{\mathsf{u,i}}^{\mathsf{old}} = \mathsf{PRF}_{\mathsf{ask}_{\mathsf{u,i}}^{\mathsf{old}}}^{\mathsf{add}}(0)$
$\wedge\, \mathsf{cm}_{\mathsf{j}}^{\mathsf{new}} = \mathsf{COM}_{\mathsf{r}_{\mathsf{j}}^{\mathsf{new}}}(\mathsf{payload}_j^{new}||\Phi_{\mathsf{b,j}}^{\mathsf{new}}||\Phi_{\mathsf{d,j}}^{\mathsf{new}}||\mathsf{apk}_{\mathsf{u,j}}^{\mathsf{new}}||\rho_j^{new})$

2) Death predicates of consumed records and birth predicates of created ones are verified:
$\forall i \in [1,m], \forall j \in [1,n], \quad \Phi_{\mathsf{d,i}} = \Phi_{\mathsf{b,j}} = 1$

Remark: As specified in the ZEXE original paper, transaction's local data *does not* contains the records' randomness generator s nor random serial number generator ρ, so they are not taken in argument by records birth and death predicates. We will explain in later section why it is useful to includes those date in the predicates arguments to avoid some attacks threatening the $weak-liveness$ property of certain exchange protocols.

The Ledger maintainer(s) accepts the transaction tx and append it to the ledger if the following conditions are met:

- The proof π is valid.
- There are no duplicate serial number, both within the transaction itself, that is $\mathsf{sn}_{\mathsf{i}}^{\mathsf{old}} \neq \mathsf{sn}_{\mathsf{j}}^{\mathsf{old}}$ for all $i \neq j$, and regarding the current serial number list, that is $\mathsf{sn}_{\mathsf{i}}^{\mathsf{old}} \notin \mathsf{SpentList}$.

As we can see, a transaction is almost completely anonymous. Reading the register does not allow to determine either its senders or its receivers; nor does it allow to determine the type(s) of processes whose state has been changed. However, since the lists $[\mathsf{cm}_i^{old}]_1^n$ and $[\mathsf{sn}_j^{new}]_1^m$ are part of the transaction, the number of records created and consumed are revealed. To counter this, we add the possibility of including dummy records in transactions, which do not contain a payload and which can be created and consumed from scratch. This gives users some control over the size of the transaction which they can then make arbitrarily large, if not arbitrarily small.

The general ZEXE framework can be used in particular to implement anonymous Payment Scheme, e.g anonymous cryptocurrencies.

2.5 User-Defined Token Payment Scheme

On the same ledger, several anonymous cryptocurrencies can coexist, represented by different tokens. ZEXE allows any user u to create one or more cryptocurrencies, and to use those created by others, by representing in records owned by u the tokens held by him. More precisely, we can use the following architecture, presented in the original ZEXE paper:

The payload $\mathsf{payload} = (\mathsf{id}, \mathbb{V}, v, \mathsf{aux})$ of a record $\mathfrak{r}$ of a user u specifies is constituted by the identifier id of the type of token contained in the record, the maximum value of the number of tokens id initially created on the ledger, v the number of tokens id that the record $\mathfrak{r}$ contains, and, if necessary, auxiliary

information aux.

The record has for birth predicate Φ_{b} "Mint or Conserve" (MoC) described Figure 2 from [BCG+20], which can be used either in the mint mode to create a new type of token, or in the conserve mode to transfer one of the tokens of an already existing type.

"When invoked in mint mode, MoC creates the initial supply $\mathbb{V}$ of the asset in, say, a single output record, by deterministically deriving a fresh unique identifier id for the asset (see below for how), and storing the tuple $(\mathsf{id}, \mathbb{V}, v, \bot)$ in the record's payload. The predicate MoC also ensures that the given transaction contains no input records or other output records (dummy records are allowed). If MoC is invoked in mint mode in other transactions, a different identifier id is created, ensuring that multiple assets can be distinguished even though anyone can use MoC as the birth predicate of a record.

When invoked in conserve mode, MoC inspects all records in a transaction whose birth predicates all equal MoC (i.e., all the transaction's user-defined assets) and whose asset identifiers all equal to the identifier of the current record. For these records it ensures that no new value is created: that is, the sum of the value across all output records is less than or equal to the sum of the value in all input records."

Mint-or-conserve predicate $\mathsf{MoC}(k, \mathsf{ldata}; \mathsf{mode})$ (mode is the private input of the predicate)

1. Parse ldata as $\begin{pmatrix} [\mathsf{cm}_i^{\mathsf{in}}]_1^2 & [\mathsf{apk}_i^{\mathsf{in}}]_1^2 & [\mathsf{payload}_i^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{d},i}^{\mathsf{in}}]_1^2 & [\Phi_{\mathsf{b},i}^{\mathsf{in}}]_1^2 & [\mathsf{sn}_i^{\mathsf{in}}]_1^2 & \mathsf{memo} \\ [\mathsf{cm}_j^{\mathsf{out}}]_1^2 & [\mathsf{apk}_j^{\mathsf{out}}]_1^2 & [\mathsf{payload}_j^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{d},j}^{\mathsf{out}}]_1^2 & [\Phi_{\mathsf{b},j}^{\mathsf{out}}]_1^2 & & \mathsf{aux} \end{pmatrix}$.
2. If $\mathsf{mode} = (\mathtt{mint}, \mathbb{v}, r)$, ensure that the first output record contains the initial supply of the asset:
   (a) the index of the current output record is correct: $k = 1$.
   (b) all other records are dummy: $\mathsf{payload}_1^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{in}}.\mathsf{isDummy} = \mathsf{payload}_2^{\mathsf{out}}.\mathsf{isDummy} = 1$.
   (c) the asset identifier is derived correctly: $\mathsf{id} = \mathsf{CM.Commit}(\mathsf{pp}_{\mathsf{CM}}, \mathsf{sn}_1 \| \mathsf{sn}_2; r)$. (See explanation below.)
   (d) the current output record's payload is correct: $\mathsf{payload}_1^{\mathsf{out}}.\mathsf{isDummy} = 0$ and $\mathsf{payload}_1^{\mathsf{out}} = (\mathsf{id}, \mathbb{v}, \mathbb{v}, \bot)$.
3. If $\mathsf{mode} = \mathtt{conserve}$, check that the value of the current asset is conserved:
   (a) parse the current output record's payload $\mathsf{payload}_k^{\mathsf{out}}$ as $(\mathsf{id}^\star, \mathbb{v}^\star, v^\star, c^\star)$.
   (b) for $i \in \{1, 2\}$, parse the i-th input record's payload $\mathsf{payload}_i^{\mathsf{in}}$ as $(\mathsf{id}_i^{\mathsf{in}}, \mathbb{v}_i^{\mathsf{in}}, v_i^{\mathsf{in}}, c_i^{\mathsf{in}})$.
   (c) for $j \in \{1, 2\}$, parse the i-th output record's payload $\mathsf{payload}_j^{\mathsf{out}}$ as $(\mathsf{id}_j^{\mathsf{out}}, \mathbb{v}_j^{\mathsf{out}}, v_j^{\mathsf{out}}, c_j^{\mathsf{out}})$.
   (d) initialize $v^{\mathsf{in}} := 0$ and $v^{\mathsf{out}} := 0$, representing the value of asset $\mathsf{id}^\star$ consumed and created (respectively).
   (e) for $i \in \{1, 2\}$, if $\Phi_{\mathsf{b},i}^{\mathsf{in}} = \Phi_{\mathsf{b}}^\star$, $\mathsf{id}_i^{\mathsf{in}} = \mathsf{id}^\star$, $\mathsf{payload}_i^{\mathsf{in}}.\mathsf{isDummy} = 0$, set $v^{\mathsf{in}} := v^{\mathsf{in}} + v_i^{\mathsf{in}}$ and check that $\mathbb{v}_i^{\mathsf{in}} = \mathbb{v}^\star$.
   (f) for $j \in \{1, 2\}$, if $\Phi_{\mathsf{b},j}^{\mathsf{out}} = \Phi_{\mathsf{b}}^\star$, $\mathsf{id}_j^{\mathsf{out}} = \mathsf{id}^\star$, $\mathsf{payload}_j^{\mathsf{out}}.\mathsf{isDummy} = 0$, set $v^{\mathsf{out}} := v^{\mathsf{out}} + v_j^{\mathsf{out}}$ and check that $\mathbb{v}_j^{\mathsf{out}} = \mathbb{v}^\star$.
   (g) check that the value of asset $\mathsf{id}^\star$ is conserved: $v^{\mathsf{in}} = v^{\mathsf{out}}$.

Fig. 2. ZEXE: algorithm performed by the ledger maintainers receiving a transaction, from [BCG+20]

"Most of the lines above are self-explanatory, but for the line that derives a fresh unique identifier in the "mint" case (Step 2c), which deserves an explanation. Informally, the idea is to derive the identifier from the (globally unique) serial numbers of records consumed in the minting transaction. In more detail, we set the identifier to be a commitment to the serial numbers of consumed input records"

The point of deriving the identifier in this way rather than letting the user choose it is to prevent a user from recreating tokens for an already existing identifier from scratch.

We notice, in this scheme , that all the tokens of a certain type are available from the creation and initially all owned by their creator. In particular, there is no mining, which is in contrast to most existing cryptocurrencies. That said, adding some form of mining can be done by modifying the birth predicate in an appropriate way.

Non Fungible Token (NFT) [ESES08]: A token whose initial unit value is 1 corresponds to an NFT.

This user defined payment scheme can serve as basis to implement several exchange protocols such as a purchase and sale agreement presented hereunder.

3 Purchase and Sale Agreement (PSA)

Property acquisition can be a complex set of transactions that requires a number of conditions, provisions, or duties to be checked and verified before the closing of the purchase and sale. These conditions are to be listed in a document called Purchase and Sale Agreement (PSA) [7]. In France, they are to be listed in a similar document called "compromis de vente" [8].

3.1 Informal use case description

The transactions happen between two participants: the "*buyer*" and the "*seller*". In this use case for the sake of simplicity, we won't consider other possible roles such as a "notary".

More specifically, let us suppose we have the following situation.

Alice owns a piece of real estate $asset$ which she wants to sell to Bob for a fixed price of $100,000\nu$, ν being a unit of a given currency.

Under French law, Bob will have ten days to retract himself without any penalty. Let's call δ this delay.

Once the purchase and sale agreement is concluded and the sale contract has been signed, Alice must reserve her property for Bob, who in turn, must put at least a percentage of the price of the estate into escrow. Bob then has a time limit δ during which he can choose to take back his funds and cancel the whole transaction with no penalty. Once this time has passed, the exchange cannot be cancelled without penalties described in the purchase and agreement and a reversal is only possible by making another transaction by mutual agreement between the two parties.

[7] http www.bakermckenzie.comfilesUploadsDocumentsGlobal%20EMIar_emi_purchasesaleagreement_jul12.pdf

[8] https://www.majordhom.fr/downloads/Vendeur/Modele-de-compromis-de-vente-majordhom.pdf

3.2 Protocol implemented with ZEXE

The architecture proposed by ZEXE makes it possible to take into account such kind of exchanges[9] either by including a smartcontract overlay of the Hash Time Locked Contracts (HTLC) type [10] , or by simply passing an auxiliary time as an argument to the predicates governing record creation and consumption. We'll be choosing this second method.

We assume that the ledger has a well-defined time stamping mechanism. For example, we can imagine that in case where the ledger is maintained in the form of a blockchain, the current block time stamp can be used to deliver a date.

Alice's *asset* is represented as the payload of some record $\mathfrak{r}_A$ of serial number $\mathsf{sn_A}$. Similarly, Bob's $100,000$ tokens are represented as the payload of some record $\mathfrak{r}_B$ of serial number $\mathsf{sn_B}$. More precisely, we will use the User-Defined Token Payment Scheme, based on the MoC predicate presented in Section 1. Thus the records $\mathfrak{r}_A$ and $\mathfrak{r}_B$ have respectively for payload $(\mathsf{id_A}, 1, 1, \perp)$ and $(\nu, \mathbb{V}, 100,000, \perp)$, for birth predicate MoC, and an empty death predicate. Here ν represents the monetary unit in which the transaction will be carried out, $\mathbb{V}$ the total number of tokens of this currency available on the whole register (defined at the time of the creation of this one), and $\mathsf{id_A}$ the NFT corresponding to Alice's property. It is not relevant to discuss here how Bob can ensure that this randomly generated identifier $\mathsf{id_A}$ indeed represents Alice's property. For example, we can assume that there is a trusted authority that unequivocally links real assets to their identifiers. Furthermore, since the initial value of the number of $\mathsf{id_A}$ tokens is 1, Bob is at least sure that there are no other copies on the register.

It is possible to implement a private purchase and sale agreement on public ledger using only ZEXE without any additional overlay, save for the definition and management of a global time t. To do this, we need to customize the user-defined token payment scheme by imposing conditions on the death predicates of the created records, as proposed in the original paper presenting ZEXE, and by using a temporary address common to the contractors and created by them.

Let us also assume that Alice and Bob have a secure communication channel. One of the possible protocols is the following **Protocol**:

1 Setup 1	Alice creates a new private address - public address pair $(\mathsf{ask}_C, \mathsf{apk}_C)$ and sends it to Bob. Bob checks that the public address apk_C is correctly computed from the private ask_C. Alice and Bob thus share a common address pair that they can both control.
2 Setup 2	Bob randomly generates an integer *rand* and sends the hash $h = \mathsf{H}(rand)$ to Alice.
	All records created afterwards share the same birth predicate MoC.

[9] In fact, for the sake of simplicity, we will not consider penalties in the following, but their is no obstacle to do so if necessary.

[10] For a definition of this mechanism, see for instance: https://corporatefinanceinstitute.com/resources/cryptocurrency/hashed-timelock-contract-htlc/

3 Record 1	Alice creates a first transaction $\mathsf{tx}_1 = (\mathsf{sn}_\mathsf{A}, \mathsf{cm}_\mathsf{C}, \pi_1)$ by spending her record $\mathfrak{r}_A$, and by creating a new record $\mathfrak{r}_C$, for the public address apk_C, of identical payload $(\mathsf{id}_\mathsf{A}, 1, 1, \perp)$, and whose death predicate $\Phi_\mathsf{d}^\mathsf{Lock}$ stipulates that $\mathfrak{r}_C$ can be spent either by being returned to Alice, or by being consumed in a transaction creating two records $\mathfrak{r}_{C,lock}^{asset}$ and $\mathfrak{r}_{C,lock}^{100000}$, where: – $\mathfrak{r}_{C,lock}^{asset}$ has for owner apk_C, for payload $(\mathsf{id}_\mathsf{A}, 1, 1, \perp)$, and for death predicate $\Phi_\mathsf{d}^\mathsf{Exchange}$, stipulating that the record $\mathfrak{r}_{C,lock}^{asset}$ must be spent in a transaction where: 1. Either a record $\mathfrak{r}'_A$, of null death predicate, be created for Alice with whose payload is $(\nu, \mathbb{V}, 100000, \perp)$ and a record $\mathfrak{r}'_B$ is created for Bob whose payload is $(\mathsf{id}_\mathsf{A}, 1, 1, \perp)$ (exchange confirmed) ; or a record $\mathfrak{r}'_A$, of death predicate null, is created for Alice with whose payload is $(\mathsf{id}_\mathsf{A}, 1, 1, \perp)$ and a record r'_B is created for Bob whose payload is $(\nu, \mathbb{V}, 100000, \perp)$ (cancelled exchange). 2. Either the time t is greater than $delta$, or an $rand$ such that $h = \mathsf{H}(rand)$ is revealed. – $\mathfrak{r}_{C,lock}^{100000}$ has for owner apk_C, for payload $(\nu, \mathbb{V}, 100000, \perp)$, and for death predicate the same $\Phi_\mathsf{d}^\mathsf{Exchange}$ Having imposed the nullity of the death predicate of the record $\mathfrak{r}'_A$ intended for Alice in the death predicate $\Phi_\mathsf{d}^\mathsf{Exchange}$ serves to prevent Bob from imposing another death predicate on $\mathfrak{r}'_A$ that would allow him to control its spending.
4 Record 1	Alice sends $\mathfrak{r}_C$ to Bob.
5 Record 2,3	Bob, after receiving $\mathfrak{r}_C$, spends it with his record $\mathfrak{r}_B$ in a transaction $\mathsf{tx}_2 = (\mathsf{sn}_C, \mathsf{sn}_B, \mathsf{cm}_{lock}^{asset}, \mathsf{cm}_{lock}^{100000}, \pi_2)$, creating two records $\mathfrak{r}_{C,lock}^{asset}$ and $\mathfrak{r}_{C,lock}^{100000}$
6 Record 2,3	Bob sends $\mathfrak{r}_{C,lock}^{asset}$ and $\mathfrak{r}_{C,lock}^{100000}$ to Alice.

7 Record 4,5	Bob then has a time t to complete the exchange (or on the contrary break the exchange) by adding to the ledger a transaction $\mathsf{tx}_3 = (sn_{lock}^{asset}, sn_{lock}^{100000}, cm'_A, cm'_B, \pi_3)$, consuming $\mathfrak{r}_{C,lock}^{asset}$ and $\mathfrak{r}_{C,lock}^{100000}$, and creating two new records $\mathfrak{r}'_A$ and $\mathfrak{r}'_B$, respectively for Alice and himself, and containing respectively $100,000\nu$ and $asset$ if he has confirmed the exchange, or conversely if he has broken the exchange.
	After the same time t, Alice also gets this possibility. [11]
8	If Bob (resp. Alice) is the first to have confirmed or broken the exchange, he (she) sends the record $\mathfrak{r}'_A$ (resp $\mathfrak{r}'_B$) to Alice (resp. Bob).

[11] Modifying slightly the death predicate $\Phi_\mathsf{d}^\mathsf{Exchange}$ of $\mathfrak{r}_{C,lock}^{asset}$ and $\mathfrak{r}_{C,lock}^{100000}$, we can also impose that Alice can only confirm the transaction at the end of a time t and not break it.

Communication complexity:In all, in a normal execution, the exchange requires:

- The transmission of 5 messages between Alice and Bob. 2 messages from Alice to Bob and 3 from Bob to Alice.
- The creation of 5 records $(\mathfrak{r}_C, \mathfrak{r}_{C,lock}^{100000}, \mathfrak{r}_{C,lock}^{asset}, \mathfrak{r}'_A, \mathfrak{r}'_B)$. The record $\mathfrak{r}_C$ is created by Alice, the four records $\mathfrak{r}_C^\star, \mathfrak{r}_C^{dagger}$ are created by Bob.
- The addition of 3 transactions to the public register. One by Alice, one by Bob, and the last one by either one of them.

Remark:This PSA protocol works promise to sell. Symmetrically, it also can work as a promise to purchase protocol, simply by having the seller choose *rand* and and transmit $\mathsf{H}(rand)$ to the buyer.

Flaws: This protocol does not satisfy the $weak-liveness$ property as define in [HLS19], i.e. it is possible for Alice to honestly follow the protocol and to find out that funds are indefinitely not available. This is fundamentally due to the fact that the protocol is not indivisible (also let's remember that roll back are not possible in a blockchain). Indeed, Bob can attack the protocol in the following way:

When Alice has performed her first transaction $\mathsf{tx_1} = (\mathsf{sn_A}, \mathsf{cm_C}, \pi_1)$, and has sent the record $\mathfrak{r}_C$ to Bob, Bob performs the transaction $\mathsf{tx_2} = (\mathsf{sn_C}, \mathsf{sn_B}, \mathsf{cm}_{\mathsf{lock}}^{\mathsf{asset}}, \mathsf{cm}_{\mathsf{lock}}^{100000}, \pi_2)$ by spending $\mathfrak{r}_C$ and $\mathfrak{r}_B$. The protocol then requires Bob to send Alice the $\mathfrak{r}_{lock}^{100000}$ and $\mathfrak{r}_{lock}^{asset}$ records thus created. However, Bob may choose not to follow the protocol from this point on, and never send $\mathfrak{r}_{lock}^{100000}$ or $\mathfrak{r}_{lock}^{asset}$ to Alice, thus keeping his own funds and Alice's property permanently locked up.

Indeed, even if the death predicates of $\mathfrak{r}_{lock}^{100000}$ and $\mathfrak{r}_{lock}^{asset}$ authorize Alice to spend the records after a certain time t, and that she possesses the secret key ask_C showing that she is indeed the owner of these records, and that she knows the payloads and the birth and death predicates of the records, this is not sufficient to be able to spend them in practice. In fact, it still lacks the two serials numbers ρ_{lock}^{100000}, ρ_{lock}^{asset} and the two randoms s_{lock}^{100000}, s_{lock}^{asset} generated by Bob to compute the corresponding commitments $\mathsf{cm}_{\mathsf{lock}}^{\mathsf{asset}}, \mathsf{cm}_{\mathsf{lock}}^{100000}$, so that the serial numbers $\mathsf{sn}_{\mathsf{lock}}^{\mathsf{asset}}, \mathsf{sn}_{\mathsf{lock}}^{100000}$ can be computed to spend them.

Exactly the same problem arises for the final record $\mathfrak{r}'_A$. If it is Bob who validates/cancels the transaction, he may choose not to send it to Alice, thus preventing her from getting her property back or accessing it if Bob has broken the exchange (or accessing her funds if Bob has confirmed it).

But here the symmetric problem also arises. If it is Alice who validates/cancels the transaction, she may not send $\mathfrak{r}'_B$ to Bob and prevent him from accessing his funds.

Countermeasure:A countermeasure would be to let Alice choose the serial numbers $\rho_{lock}^{100000}, \rho_{lock}^{asset}$ and the randoms $s_{lock}^{100000}, s_{lock}^{asset}$ to build the records $\mathfrak{r}_{lock}^{100000}, \mathfrak{r}_{lock}^{asset}$, so that she is sure to have control over them, and a serial number

ρ'_A and a random s'_A to be sure to have control over the final $\mathfrak{r}'_A$. Similarly, we should let Bob choose the ρ'_B and s'_B, in case Alice creates the $\mathsf{tx_3}$ transaction.

Specifically, the changes to be made are as follows:

1 Setup 1	In her first message, Alice sends to Bob the serial numbers and randoms $\rho_{lock}^{100000}, \rho_{lock}^{asset}, s_{lock}^{100000}, s_{lock}^{asset}, \rho'_A, s'_A$, in addition to the pair $(\mathsf{apk}_C, \mathsf{ask}_C)$.
2 Setup 2	In his first message, Bob sends Alice the serial number and random ρ'_B and s'_B, in addition to the random *rand*.

When $\mathfrak{r}_C$ is created, the previously created serial numbers and randoms appear as auxiliary data in the payload. The death predicate $\mathfrak{r}_{C,lock}^{100000}, \mathfrak{r}_{C,lock}^{asset}$, in addition to forcing the death predicates of the records $\mathfrak{r}_{C,lock}^{100000}, \mathfrak{r}_{C,lock}^{asset}$ to equal $\mathfrak{r}_{C,lock}$, specifies that their respective serials numbers and random are $\rho_{lock}^{100000}, s_{lock}^{100000}$ and $\rho_{lock}^{asset}, s_{lock}^{asset}$.

The death predicate $\Phi_{\mathsf{d}}^{\mathsf{Exchange}'}$ itself modified takes up the death predicate $\Phi_{\mathsf{d}}^{\mathsf{Exchange}}$, in addition to imposing that the serial numbers and the random of $\mathfrak{r}'_A$ (resp $\mathfrak{r}'_B$) be ρ'_A, s'_A (resp ρ'_B, s'_B).

Such a countermeasure cannot be directly implemented in the current ZEXE protocol, because in the current implementation of the ZEXE protocol, the birth and death predicates of the records take as arguments only the payloads, the death and birth predicates, and the public keys of the records created and consumed in a transaction.

3.3 Modification of ZEXE protocol

We propose a slight alteration of the ZEXE protocol.

First, we propose to remove the condition which imposes that, in a transaction $\mathsf{tx} = ([\mathsf{sn}_i]_1^m, [\mathsf{cm}_j]_1^n, \pi)$, the serial number nonces of created records are derived from serial numbers of spent nonces as $\rho_j = \mathsf{CH}(\mathsf{pp_{CM}}||\mathsf{j}||\mathsf{sn_1}||...||\mathsf{sn_m})$ [12]. We proposed instead that users simply randomly generated them when they create new records.

For each transaction $\mathsf{tx} = ([\mathsf{sn}_i^{old}]_1^m, [\mathsf{cm}_j^{new}]_1^m, \pi)$, recall that the local data taken in argument by the birth and death predicates of each records currently consist in:

- Owners public keys $[\mathsf{apk}_i^{old}]_1^m, [\mathsf{apk}_j^{new}]_1^n$
- Birth predicates $[\Phi_{\mathsf{b,i}}^{\mathsf{old}}]_1^n, [\Phi_{\mathsf{b,j}}^{\mathsf{new}}]_1^m$
- Death predicates $[\Phi_{\mathsf{d,i}}^{\mathsf{old}}]_1^n, [\Phi_{\mathsf{d,j}}^{\mathsf{new}}]_1^m$
- Payloads $[\mathsf{payload}_i^{old}]_1^n, [\mathsf{payload}_j^{new}]_1^m$
- Commitments $[\mathsf{cm}_i^{old}]_1^m, [\mathsf{cm}_j^{new}]_1^n$
- Serial numbers $[\mathsf{sn}_j^{new}]_1^n$

[12] here $\mathsf{pp_{CM}}$ is a public parameter, and j a the position of the record corresponding to ρ_j in the transaction

We propose to add to this list the others records private data, namely:

- Serial number nonces $[\rho_i^{old}]_1^m, [\rho_j^{new}]_1^n$
- Commitment randomness $[s_i^{old}]_1^m, [s_j^{new}]_1^n$

In fact, a similar modification seems to be proposed in the remark 6.1 of the updated ZEXE paper [BCG+20]:

"Remark 6.1 (preventing a denial-of-funds attack). In the [Decentralised Exchange protocol we proposed], the maker M could refuse to provide the taker T with information about its output record, thus denying T the ability to consume 35 its exchanged record. A simple approach to prevent this is to modify the exchange-or-cancel predicate to additionally enforce that the memorandum field of the created transaction contains an encryption of the output record information under a public key specified by T."

But this modification is not mentioned in the general protocol's presentation.

Another point is that the method proposed by the ZEXE paper, given the slight modification we propose is used in a different way that we mention to overcome $weak - liveness$ issue trading the use of transaction memorandum for less offchain communication.

So our method have some advantage and drawback in comparison:

Advantage: A slight privacy issue is that, in case where the ledger supports only a small number of different processes, a transaction memorandum, by its size, can contribute to identify the undergoing process of a transaction. From the performances point of view, it is likely that the proof π of a transaction is harder to generate if it proves encryption for a particular public key than if it proves computation of hash function for a certain input. Abandoning the serial number nonce generation condition also simplifies the statement which must be proven for each transaction, also saving computation time.

Drawback: The drawback of our proposed modification regarding the ZEXE's one is that it requires more offchain communications, which could potentially lead to attacks. Abandoning the serial number nonce derivation from transaction inputs could also potentially lead to attacks by reusing several time the same nonce.

4 Fairness-Privacy Vulnerability

Let consider a simpler case where Alice and Bob wants only to do a direct exchange between the Alice's good *asset*, represented in a records $\mathfrak{r}_A$ with payload $(\mathsf{id_A}, 1, 1, \perp)$ birth predicate MoC and null death predicate, and Bob's 100000ν represented in a record $\mathfrak{r}_B$ with payload $(\nu, \mathbb{V}, 100000, \perp)$ and same predicates as $\mathfrak{r}_A$.

A way to do it is to follow the ZEXE indications, creating a shared common address between Alice and Bob, sending a first transaction with, say, the Alice's

asset to it, in a record $\mathfrak{r}_C$ with the additional condition that $\mathfrak{r}_C$ can be spent in a second transaction, either being re-transmitted to Alice if the transaction is cancelled, or being transmitted to Bob at the condition that 100000ν is sent to Alice if the transaction is confirmed.

The potential $weak - liveness$ issue due to the fact that Bob can confirm the transaction without sending the final record to Alice can be circumvent by one of the two methods described in the preceding section.

Instead of using this method, we can imagine that Alice and Bob use secure multi-party computation to compute together a transaction exchange Alice's *asset* and Bob's 100000ν. Suppose we have apply the slight modification of the preceding section so that the serial number nonces can be derived by simple sampling random numbers, and that records' predicates take in input commitments' randomness and serial number nonces. The protocol to follow would simply be the following:

Protocol:

1 Record	Alice computes a record $\mathfrak{r}'_B$ (including its commitment cm'_B) with payload $(\mathsf{id}_\mathsf{A}, 1, 1, \perp)$, birth predicate MoC, death predicate null, serial number nonce ρ'_B and commitment's randomness s'_B. She sends $\mathfrak{r}_{B'}$ to Bob.
2 Record	Bob computes a record $\mathfrak{r}'_A$ (including its commitment cm'_A) with payload $(\nu, \mathbb{V}, 100000, \perp)$, birth predicate MoC, death predicate null, serial number nonce ρ'_A and commitment's randomness s'_A. He sends $\mathfrak{r}_{A'}$ to Alice.
3 MPC	Alice and Bob engage in a multi-party computation protocol to compute together the proof π attesting the validity of the transaction $\mathsf{tx} = (\mathsf{sn}_\mathsf{A}, \mathsf{sn}_\mathsf{B}, \mathsf{cm}'_\mathsf{A}, \mathsf{cm}'_\mathsf{B}, \pi)$., using their respective private inputs ρ_A, s_A and ρ_B, s_B corresponding to input's records, where $\mathsf{cm}'_\mathsf{A}$and cm'_B are the respective commitments of $\mathfrak{r}'_A$ and $\mathfrak{r}'_B$, and $\mathsf{sn}_\mathsf{A}, \mathsf{sn}_\mathsf{B}$ the serial numbers of the original records $\mathfrak{r}_A, \mathfrak{r}_B$.
4 Share	Alice send her sn_A to Bob.
5 Transaction	Bob, now disposing of all the data needed, appends the transaction tx to the ledger.

Flaw: If Bob is dishonest, he can abort the protocol at the moment he receives sn_A from Alice, never sending the full transaction $\mathsf{tx} = (\mathsf{sn}_\mathsf{A}, \mathsf{sn}_\mathsf{B}, \mathsf{cm}'_\mathsf{A}, \mathsf{cm}'_\mathsf{B}, \pi)$ to the ledger, (nor his own sn_B to Alice, which would allows her to send herself the transaction tx to the ledger).

The interest of Bob of doing so is limited. Indeed, he cannot use the sn_A sent by Alice in any other transaction tx' because he wont be able to compute the corresponding Zero-Knowledge Proof, which requires the knowledge of the whole private information of $\mathfrak{r}_A$ in addition to the private address ask_A of Alice. However, he can now de-anonymize the records $\mathfrak{r}_A$ of Alice, knowing exactly when she spent it by simply scrutinizing the pubic list $\mathsf{SpentList}$ to see when sn_A

is append to it (and potentially share this knowledge to any other parties). We call this issue a *fairness-privacy* vulnerability.

An easy way to circumvent this is simply that Alice and Bob, after doing multi-party computation of the proof π, simply send π with their respective serial numbers to a trusted third party[13] which append full the transaction $\mathsf{tx} = (\mathsf{sn}_A, \mathsf{sn}_B, \mathsf{cm}'_\mathsf{A}, \mathsf{cm}'_\mathsf{B}, \pi)$ after reception of both sn_A and sn_B. However, we doesn't want to suppose the presence of such trusted third party, and so we proposed other possible counter-measures.

4.1 Countermeasure 1

The most obvious and easy way to counter the attack is simply for Alice to resend to herself the assets *asset*. Passed a certain amount of time after sending sn_A to Bob without any publication of the full transaction tx on the ledger (or at least a communication by Bob of sn_B), she considers the transaction is aborted and she can use her serial number sn_A in a new transaction, creating a new record $\mathfrak{r}^\dagger_A$ to her own public address embodying the property of *asset*, whose serial number $\mathsf{sn}^\dagger_\mathsf{A}$ will be unknown to Bob.

However, to do so for a particular record $\mathfrak{r}$, note that it is necessary that its birth and death predicates allows such manipulation, which is not guaranteed a priori. Moreover, it is not possible to enforce that at the level of the ledger[14], because those birth and death predicates are completely unknown to it, so this countermeasure have to be taken at each user level.

We didn't found concrete case were adding the possibility for a user to re-anonymize his asset by re-sending it to himself would be infeasible. However it could be desirable to have another type of countermeasure in case such case arises.

4.2 Countermeasure 2

The vulnerability one which is based the *fairness-privacy* attack is the public character of the list of SpentList. Let us reminds that the public character of this spending list itself serves to avoid double spending : receiving a new serial number, the ledger verifies that it does not appears in SpentList, if it doesn't the ledger accepts the transaction an adds it the spending list. In particular, theoretically, this not absolutely mandatory that the spending list appears in clear on the ledger, we just need a way to ensure that serial number have not been use several times, a thing that we could do if we dispose the following primitive.

[13] The "trust" put in such third party would be quite limited, it can know nothing about the content of the transaction.

[14] Excepts if we enforce each transaction's proof to prove that each created record it involves has birth/death predicates allowing the cut-and-paste operation, which would be tedious

Cryptographic Waste Basket with Collision Detection (CWBCD) A CWBCD is a cryptographic scheme allowing several mutually distrustful participants to share an obfuscated list of elements such that every participant can add element to the list, but no one can extract the list of added elements from the obfuscated one, and no one can add an element already present in the list.

Definition: A CWBCD consists in a tuple of polynomial time algorithms $\Pi = (\mathsf{Setup}, \mathsf{Add}, \mathsf{Verify})$

$\mathfrak{L}_{init} \leftarrow \mathsf{Setup}(\lambda)$ takes as input a security parameter λ and output an initial state $\mathfrak{L}_{init}$ of a CWBCD, containing no value at all.

$\mathfrak{L}_{i+1} \leftarrow \mathsf{Add}(\mathfrak{L}_i, v)$ takes as input a CWBCD at state i and a value v, and output a CWBCD at state $i+1$ containing the value of $\mathfrak{L}_i$ plus the value v.

$\mathsf{b} \leftarrow \mathsf{Verify}(\mathfrak{L}_i)$ takes as input a CWBCD at state i, and output 1 if $\mathfrak{L}_i$ is a valid state, in particular containing no repetition and 0 elsewhere.

Security: A CWBCD is secure if, for all real-world adversary $\mathcal{A}$, there exist an efficient ideal-world simulator $\mathcal{S}_{\mathcal{A}}$ such that, for every efficient environment $\mathcal{E}$, the outputs of $\mathcal{E}$ interacting with the adversary $\mathcal{A}$ in real-world execution and the outputs of $\mathcal{E}$ interacting with the simulator $\mathcal{S}_A$ are indistinguishable.

Ideal Functionality: The ideal functionality $\mathcal{F}_{CWBCD}$ that a CWBCD is designed to simulate is given by the following protocol:

- Receive an input x.
- If $\mathsf{x} \notin \mathsf{FuncList}$ then add x to $\mathsf{FuncList}$, and send the message $\mathsf{input\ added}$ to all the parties.
- Else send the message $\mathsf{input\ rejected}$ to all the parties.

Tracks for the CWBCD Based on the discrete logarithm problem and pairing, it is possible to design dynamical accumulator with proof of non repetition [DT08]. We can try to draw inspiration from these methods, but we must keep in mind that our problem is quite different. Indeed, in the case of an accumulator, the prover knows the whole list of accumulated values, while in our case, a participant didn't know any accumulated value except the ones he added himself.

The general idea would be to associate to any serial number e_i a unique prime number p_i, select a group where the discrete logarithm problem is hard and one base point g (or several base points $(g_1, g_2, .., g_n)$) then consider a CWBCD of the form $g^{p_1 p_2 \cdots p_m}$ (or $(g_1^{p_1 p_2 \cdots p_m}, g_2^{p_1 p_2 \cdots p_m}, ..., g_n^{p_1 p_2 \cdots p_m})$. Addition of a new element p_i would simply be given by exponentiation of the previous state by p_i. However, such protocol would need a detection of square in exponent which we don't currently know how to do [15]. Another difficulty to face is the fact that during an addition procedure, it would potentially be easy for the adder to delete

[15] Wouldn't having an oracle that, given g^x, answers whether or not there is a square in the factorization of x allowing solving the discrete logarithm problem?

a previously appended value p_{old}, exponentiating by $\frac{1}{p_{old}}$.

A variant would be to use a trusted set up $[g^{s^i}]_1^N$ (or $([g_1^{s^i}]_1^N, [g_2^{s^i}]_1^N, ..., [g_n^{s^i}]_1^N)$ where s is a secret exponent, and to associate to each e_i not a prime number p_i, but a monomial $(s - e_i)$.

The element $g^{(s-e_i)}$ can then simply be computed by a participant as $\frac{g^s}{g^{e_i}}$. The CWBCD would then take the form $g^{(s-e_1)(s-e_2)...(s-e_i)}$ and detecting a square $(s - e_j)^2$ in the exponent might be easier, though we don't know how to do that either.

Moreover, several problems would arise: the first one being that we would have to find a way to add $(s-e_i)$ to the CWBCD, and that simply exponentiating the previous state is impossible since the value $(s - e_i)$ is not itself known by the adder. The second is that the trusted setup should have a very large size N, equal to the maximum number of elements that can be added to the list (and thus, for the application we are interested in, to the number of records supported by the system, which is itself greater than or equal to the number of possible transactions).

5 Conclusion

In the use-case of purchase and sale agreement, slight modifications of the ZEXE protocol can be applied to guarantee $weak - liveness$ properties. There is alternative to the one presented in the ZEXE paper which are less costly in computation time and can improve function privacy, but is potentially less secure since it requires more offchain communications and random nonces.

We have shown that the use of MPC together with the ZEXE protocol to build transaction with distinct senders can lead to *fairness-privacy* issue, in case on of the party aborts the protocol. To circumvent this, either we need additional specifications to ensure that a user can always copy a state for himself, or we need a new specific cryptographic primitive, yet to be invented, allowing mutually distrustful parties to share an obfuscated append-only list with no repetition.

References

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 326–349, New York, NY, USA, 2012. Association for Computing Machinery.

[BCG+20] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. ZEXE: Enabling Decentralized Private Computation. In *2020 IEEE Symp. on Security and Privacy (SP)*, pages 947–964, San Francisco, CA, USA, May 2020. IEEE.

[BSCG+14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. *Zerocash: Decentralized Anonymous Payments from Bitcoin.* Cryptology ePrint Archive, Paper 2014/349, 2014.

[DDJ+21] S. Dalmas, P. Duvaut, G. Jacovetti, S. Tucci, A. Lanusse, G. Gonthier, and G. Memmi. Les verrous technologiques des blockchains. Direction générale des Entreprises, Numéro du rapport: 978-2-11-162212-8, 2021.

[DT08] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 01 2008.

[ESES08] W. Entriken, D. Shirley, J. Evans, and N. Sachs. Erc-721 non-fungible token standard., 2108.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. *The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol01] Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[HBHW22] D. Hopwwod, S. Bowe, T. Hornby, and N. Wilcox. *Zcash Protocol Specification*. Electric Coin Co, 2022.

[HLS19] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *Proc. VLDB Endow.*, 13(2):100–113, oct 2019.

[KYMM18] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. *An Empirical Analysis of Anonymity in Zcash*. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 463–477, Baltimore, MD, August 2018. USENIX Association.

[Nak08] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*, 2008.

[SJZG18] Alan T. Sherman, Farid Javani, Haibin Zhang, and Enis Golaszewski. On the origins and variations of blockchain technologies. *CoRR*, abs/1810.06130, 2018.