University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Cybersecurity

FairDrop: a Confidential Fair Exchange Protocol for Media Workers

Supervisor
Prof. Mauro Conti
University of Padova

Co-supervisor
Prof. Zekeriya Erkin
Delft University of Technology

Master Candidate
Egon Galvani

Student ID
2028932

Academic Year
2021-2022

Acknowledgments

I would like to express my gratitude to Prof. Mauro Conti, the supervisor of my thesis, for his help and support during the writing of this dissertation. It was a pleasure to meet a person who loves his work so much and shares his passion for the world of research with it. I would also like to thank Prof. Zekeriya Erkin and Dr. Oğuzhan Ersoy for all the feedback and meetings we had over the past few months, and for the good conversations and advice you gave me in Delft.
I would like to sincerely thank my family for their support and great help. Starting with Luna, my twin, my ge, and my biggest supporter, always ready to help me in difficult times and always being able to make me feel proud. Asia, for her confidence and for making me see things from a more relaxed point of view. I want to thank my parents, who never set limits on me and always left me the freedom to decide what to do on my own.
Next, I have the desire to thank my friends for the 4 wonderful years we spent together and for all the experiences, ideas, and thoughts we shared. I want to thank Aslı, for having started our conversations on the 6th floor, for the thoughts shared in the library, and for all the toppers of pasta unlocked together. Vitto for all the ideas and dreams we talked about over the years, for all the failed projects and jokes that filled our time together. Massi for being the best partner in crime during our months in Delft; I will always remember the dinners, drinks, conversations, and rides we shared. I want to thank Tommaso, for all the years he put up with me, and for continuing to do so, and Loris and Gian for the very long walks and nights spent together. Ludovica, Francesco, and Filippo for being the best friends I could hope to find during a course of stochastic processes. Mirko and Elton for all the lunches we survived in the canteen, for the talking and laughing we did together. Osi, Cezza, Dema, Jack, and Valton for all the sushi we ate together and for the time we spent in the front rows of the P200.

Having arrived at the end of this journey, I also want to say thank you to myself, for the effort and passion I have put in over the years. And for all the decisions I have made, the results of which I am very proud.

To all of you,
Thank you ;)

Abstract

In recent years, the asymmetry between open societies and regimes that control their media has increased, leading to the number of murdered journalists more than doubling worldwide. Even in countries in which freedom of the press is publicly recognized, the number of journalists jailed, assaulted, or criminally charged is relevant and growing. These attacks on media workers usually want to limit or control information regarding critical topics.
In this context, the necessity of a system that allows reporters to publish their works without risking their own life is evident. Some systems to share information with newspapers while keeping the source anonymous exist. An example is SecureDrop, developed and maintained by the Freedom of the Press Foundation, and widely adopted by all major international newspapers. What limits them from extensively using this type of system is the lack of credibility in the information exchanged, which represents the main problem for the publisher's reputation.
In this thesis, we present FairDrop, a system that allows the exchange of information between two untrusted parties and proposes a tradeoff between the anonymity of the source and the credibility of the information exchanged. We present a fair exchange protocol based on blockchain that allows sharing of a digital good fairly and confidentially. We also define the guidelines for a system based on ring signatures to measure the credibility of the exchanged information.
All our design decisions are made taking into account the requirements of a journalist-newspaper communication, and the guidelines for anonymous sources applied by major newspapers around the world. We test the system in a real-world blockchain testnet, considering multi-seller and buyer situations, and introducing economic incentives for sources to use the system.

Contents

Listing of figures

Listing of tables

Listing of acronyms

CPJ ..... Committee to Protect Journalists

SPJ ..... The Society of Professional Journalists

AP ..... The Associated Press

NYT ..... The New York Times

ROM ..... Random Oracle Model

PPT ..... Probabilistic Polynomial Time

IND-CPA ..... Indistinguishable under Chosen Plaintext Attacks

EVM ..... Ethereum Virtual Machine

TTP ..... Trusted Third Party

PoM ..... Proof Of Misbehaviour

VRF ..... Verifiable Random Function

PoW ..... Proof of Work

PoS ..... Proof of Stake

1 Introduction

1.1 Introduction

In recent years, the asymmetry between open societies and despotic regimes that control their media and online platforms has increased [1]. The World Press Freedom Index 2022 shows that the polarization of press freedom has been subject to a two-fold increase, leading to deeper divisions within countries, but also among countries at the international level [1]. This phenomenon is closely linked to the deterioration of security conditions of reporters, who in some cases have to risk their own lives to pass on information that has not been censored.
According to the Committee to Protect Journalists (CPJ), the number of murders of journalists has more than doubled worldwide in 2020 compared to the previous year, passing from 10 in 2019 to 22 in 2020 [2]. In 2021 this number didn't change, having 22 media workers singled out for murder in retaliation for their reporting and registering India and Mexico as the countries with the most media workers killed during the year [3]. Even in countries where this number is lower, a high percentage of journalists is imprisoned: 2020 was the fifth consecutive year during which repressive governments imprisoned at least 250 journalists, limiting the amount of shared information regarding critical topics. For two years in a row, China was the first nation for the number of jailed media workers, followed by Turkey, Egypt, and Saudi Arabia [4]. Also in democracies, this type of problem is still actual: in 2020, 452 journalists were assaulted and 143 arrested or criminally charged [4].

Considering the previous data, it is evident how a system that allows journalists to publish their work without risking their life would have a strong impact on this field: allowing them to overcome censorship and achieve a free flow of information. A naive solution to this problem is to allow entities to publish information anonymously: data can be shared by sources who are unwilling or unable to reveal their identities publicly. From a practical point of view, newspapers tend to avoid this approach [5, 6, 7] since their reputation could be affected if the published news turns out to be false. This credibility problem also affects the audience, which can be subject to disinformation, since it can be difficult to determine whether information attributed to an unnamed source is reliable, simple rumor or untrue [8, 9, 10, 11]. The problem of allowing sources or insiders to deliver anonymous information is called anonymous sourcing or whistleblowing [12]. The main solution currently adopted by most of the main newspapers and media organizations is SecureDrop [13, 14]. It is an open-source whistleblower submission system, that allows news organizations to receive documents and exchange messages with the source anonymously. It does not use third parties, but allows direct communication between source and organization, minimizing the shared metadata and encrypting the communication channel [13]. It allows each interested media company to install its instance of it, forcing the usage of security best practices for journalists and guaranteeing compliance for high-risk environments. Even if SecureDrop allows anonymity of the source, the media organization usually has still the necessity to identify it to guarantee the credibility of the exchanged information [5, 6, 7]. The general description of the source in the final article leads to a loss of trust by the audience [8, 15], which does not have any way to check directly the credibility of the source itself. Moreover, SecureDrop does not provide the source any incentive to share its information and take this risk.

1.2 Newspaper guidelines

Over the years, the journalism literature regarding anonymous sourcing has increased, leading to two main schools of thought. Many scholars defend the use of anonymous sources as reporting method when: there is no conflict of interest by the source, the sharing is preventing either physical or emotional harm to the source [16] or the information compels public interest [17]. In any case, most of them still require the identification of the source to determine its goal or interest in sharing the information.
On the other side, many scholars [18, 19, 20, 21, 22] criticize an excessive usage of this reporting method, since it can lead the press to publish and grant anonymity too easily.

Starting from the literature, each main newspaper and journalist society defined a code of ethics that covers, among other things, how anonymous sources are managed by the newspaper itself. A brief overview of some of them follows.

SPJ Code of Ethics The Society of Professional Journalists (SPJ) in its Code of Ethics [5] contains two main statements regarding anonymous sourcing:

- *identify sources whenever feasible*: the public is entitled to as much information as possible on sources' reliability;
- *always question sources' motives before promising anonymity*: clarify conditions attached to any promise made in exchange for information.

The SPJ focuses mainly on the consumer's faith in the newspaper and the credibility of the published information. For this reason, the guidelines refer to identifying the source as clearly as possible without pointing a figure at the person who has been granted anonymity [5]. The usage of anonymous sources is more tolerated when it is the only way to publish a story that is of importance to the audience. Regarding the second statement of the code of ethics, the identification of the source is often required to understand if the information sharing is based on a conflict of interest. SPJ wants to avoid a source sharing information to undermine someone else, to even the score with a rival, to attack an opponent, or to push a personal agenda [5]. In general, verification is required by multiple actors, and publishing without verification is considered a dangerous practice.

The Associated Press The Associated Press (AP), define its policy regarding anonymous sources highlighting the importance of transparency and credibility with the public and their subscribers. They define the usage of anonymous sources as possible only when [6]:

- the material is information and not opinion or speculation, and is vital to the report;
- the information is not available except under the conditions of anonymity imposed by the source;
- the source is reliable, and in a position to have direct knowledge of the information.

In its guidelines [6] the AP refers to the requirements of having more than one source, with an exception in cases in which "the material comes from an authoritative figure who provides information so detailed that there is no question of its accuracy". They also specify how it is necessary to provide the public with a clear description of the sources, maintain their anonymity, and describe the motivation for sharing their information.

New York Times The New York Times (NYT) provides a guideline on how their reporters manage anonymous sources, and the company guidelines about them [7]. Under their guidelines, this reporting tool is used only when necessary, that is when the news is newsworthy and credible, but it is not possible to report it in any other way. They also recognize how "many important stories in sensitive areas like politics, national security and business could never be reported if we banned anonymous sourcing" [7]. NYT recognizes how people in sensitive positions would just state the official line if recorded, and for this reason, they are open to the possibility of anonymous sources, always keeping a critical and skeptical approach.

1.3 Relevant historical examples

One of the most relevant political scandals related to anonymous sourcing is Watergate, which produced one of the most famous unnamed sources of all time, called with the name: Deep Throat. His identity was not revealed until several years later, in 2005 [23]. Before Watergate, confidential sources were rare, while after that the usage of anonymous sources raised, with many reporters considering them more appealing than having named sources [24].
Bob Woodward, who with Carl Bernstein did much of the reporting for the Watergate scandal, highlighted how the Watergate coverage, that led to President Richard Nixon's resignation in 1974, would have been impossible without unnamed attribution [24].
With the growing interest in anonymous sources, problems didn't take long to arrive: famous is the article "Jimmy's World" [9] by the Washington Post reporter Janet Cooke, published in 1980. Cooke published a story about "Jimmy", an 8-year-old heroin addict. With the motivation of protecting its sources, Cooke convinced her editors to publish the story without naming them at all. The story was discovered to be invented only in 1981 [9] after Cooke won the Pulitzer Prize and reminded the media community of the credibility problems related to anonymous sources.
Another case of an inaccurate report using anonymous sources was published by the Newsweek newspaper in 2005 [10]. The magazine claimed that government sources had confirmed that U.S. personnel at the Guantanamo Bay detention camp had intentionally damaged a Quaran and led a detainee around with a collar and dog leash. The news provoked anti-US demonstrations in the Islamic world, that resulted in at least 15 deaths [25].
The magazine later retracted the article, saying that further verification of the report was needed and that the source changed its mind about what had happened [11].
In recent years, other newspaper scandals have been linked to anonymous sources: on several

occasions, journalists used the pretext of unnamed sources to fake major stories [26, 27] and to get more engagement from the public. In summary, some of the most important stories could never have been told without relying on sources unwilling to reveal their names (e.g., Watergate, the Pentagon Papers, Guantánamo). However, the presence of misleading episodes leads the audience to perceive articles with anonymous sources as less credible than reports with named sources [28, 29].

1.4 Contribution

In this paper, we present `FairDrop`: a fair exchange protocol that allows anonymous sourcing and introduces an approach to manage the credibility of the information shared anonymously. The protocol considers two main parties: a *source* or *reporter*, that is willing to share his information, and a *newspaper*, interested in obtaining and publishing it. As usual in fair exchange protocols, the two parties do not trust each other, but they want to exchange a digital good, i.e. a file. Usually, the file is exchanged by the use of a description, such as its hash, that the two parties agree about. In our scenario, this type of description can't be used, since it can't be trusted by itself. Thus, we use a form of sampling-based [30, 31, 32, 33, 34, 35] fair exchange. The fair exchange that we propose takes most of its ideas by FairDex [30] since it allows us to achieve low on-chain and off-chain costs using symmetric encryption and a logarithmic-size proof of misbehavior, based on the sampling of sub-keys instead of sub-files [36, 37]. However, FairDex is missing confidentiality of the shared key, management of multiple files, and re-selling of the same product, which avoid it to be used in practical contexts.
For the credibility aspect, we leverage ring signatures to allow each reporter to share information and hide his identity behind a group of other members, without the necessity for them to actively participate in the procedure. Depending on the composition of the group we propose some ideas on how a measure of the credibility of the shared information can be derived. The main goal of our design is to define a solution that allows finding a trade-off between anonymity and credibility of the sources, guaranteeing fairness for all the involved parties.

`FairDrop` considers mainly two different sides:

1. *Credibility*: a way to allow each newspaper interested in the information to derive some measure of the credibility of the source itself;

2. *Fair Exchange*: a fair exchange protocol that allows managing in a fair, confidential, and anonymous way the exchange of the file shared by the source.

The relevant aspects of our protocol can be summarized as follows:

- *Confidentiality of fair exchange*: if both parties behave correctly, the protocol allows only the receiver of each exchange to learn the information, while in many protocols proposed up to now [30, 36] the key would be published in plain text inside the blockchain.
- *Proof of credibility*: using ring signatures the newspaper receives a signature of the file and can share it with its customers, allowing them to verify it and check the credibility of the ring.
- *No direct communication*: unlike other protocols [30], an authentic and confidential communication channel between these two parties is not required: all the operations that do not use the blockchain, can be executed by sharing data in a public channel, e.g., a website or public directory.
- *Protection against grieving attacks*: the protocol considers and avoids situations where an adversary attempts to violate the financial fairness of the protocol by forcing the honest party to pay expensive transaction fees [37].
- *Practical implementation*: a working and fully implemented version of the designed fair exchange is available publicly. We executed the protocol on multiple testnet, i.e., Ethereum Ropsten and Goerli, comparing our results with the solutions proposed in the literature.
- *Low on-chain and off-chain cost*: the fair exchange protocol is based on symmetric key encryption which requires low off-chain computation and storage cost, and it requires publication of only the master key on the blockchain.

1.5 Organization of the document

Before describing the proposed protocol, we provide all the necessary elements to understand it, starting from its building blocks, like blockchain, fair exchange protocols, and ring signatures, and then considering the main related works. All these information are provided in Ch. 2.
In Chapter 3, the problem that the protocol wants to solve is formalized, and our proposed solution for media workers is introduced and described in detail. The protocol implementation and performance evaluation are described in Chapter 4.
Finally, in Chapter 5 we discuss the conclusions and possible future research directions of our work.

2
Preliminaries

In this chapter, we introduce the fundamental building blocks for our work, starting from the necessary cryptographic primitives used, for then focusing more on the blockchain and fair exchange protocols. We present and analyze the principal aspects introduced by FairSwap [36] and FairDex [30], which represent the main reference point for our fair exchange design.

2.1 Cryptographic Primitives

In this section, we introduce the building blocks and the main cryptographic primitives used to design our protocol.

Hash Functions It represents the main element of the protocol. A hash function $H : X \rightarrow Y, X = \{0,1\}^*, Y = \{0,1\}^m$, maps a binary string of arbitrary length to a binary string of fixed length m. We consider H to be a cryptographic hash function, which means that the following properties hold:

- *Preimage resistance*: given an hash value y, it is difficult to find any message x s.t. $y = H(x)$.
- *Second pre-image resistance*: given an input x_1, it is difficult to find an input $x_2 \neq x_1$, s.t. $H(x_1) = H(x_2)$.

- *Collision Resistance*: it is difficult to find any pair $(x_1, x_2) \in X^2, x_1 \neq x_2$ s.t. $H(x_1) = H(x_2)$.

For model H, we use the Random Oracle Model (ROM) [38]. It considers an oracle (i.e., black box), that given an input returns a random response chosen uniformly from its output domain. If the same query is repeated, then the corresponding response doesn't change.

Symmetric Encryption We make use of a symmetric encryption scheme consisting of three PPT (probabilistic polynomial-time) algorithms: `Sym-KeyGen`, `Sym-Enc`, and `Sym-Dec`. Assuming key space K, plaintext space M and cipher space X, then:

- `Sym-KeyGen` algorithm is used to generate the secret encryption key $k \in K$;
- `Sym-Enc` algorithm outputs a ciphertext x, given a key k and a plain message m;
- `Sym-Dec` algorithm outputs the message m for a given ciphertext c and key k.

We assume a symmetric encryption scheme to be indistinguishable under chosen-plaintext attacks secure (IND-CPA). This means that every probabilistic polynomial time adversary has only a negligible advantage over random guessing [38].

Asymmetric Encryption We make use of an asymmetric encryption scheme consisting of three PPTs: `Asym-KeyGen`, `Asym-Enc`, and `Asym-Dec`. In this case, we consider two keys: a public key pk and a private one sk, belonging to the corresponding public key space P and private key space S. We consider $sk \sim U(S)$ and a function $f : S \to P$ s.t. $pk = f(sk)$. Regarding the `Asym-Enc` algorithm, it outputs a ciphertext x starting from a message m and a private key sk; while the `Asym-Dec` algorithm allows retrieving the plaintext message by starting from the ciphertext x and the public key pk. We assume that:

- it is hard to derive the private key sk from pk, i.e., f is a one-way function;
- it is hard to derive the message m from (pk, x);
- it is hard to derive the private key sk from (m, x).

Ring Signatures We consider a ring (i.e., a group) of n entities, each characterized by a public/private key pair (sk, pk), with $sk \in S$ private key space, and $pk \in P$ public key space. Considering $\mathcal{M}$ as message space, we can identify the following main PPT algorithms [39]:

- `Ring-Sign`: given a message m, the list of public keys $pk_1, \ldots, pk_n$ of the members of the group and the private key sk_s of the s-th member of the group, i.e., the signer, it produces a ring signature σ;
- `Ring-Verify`: it accepts in input a message m and a signature σ (containing also all the $pk_1, \ldots, pk_n$ public keys of the ring), and outputs *true* or *false* depending on the validity of the signature.

Unlike group signatures [40], ring signatures are *set-up free*: the signer does not need the knowledge or consent of the other ring members to put them in the ring. Also, the signature has to be *signer-ambiguous*: it has not leaked information about the signer [39].

Merkle Trees Considering an alphabet $A = \{0, 1\}$, and a list of n elements $x_1, \ldots, x_n \in A^*$, we can define a Merkle Tree $\mathcal{M}$:

- as a binary tree having as leaves the values $x_1, \ldots, x_n$;
- considering a hash function H over alphabet A, then each non-leaf node v_j of $\mathcal{M}$ is obtained computing the hash of the value of its two child nodes, i.e., $v_j = H(v_j^l, v_j^r)$.

For simplicity in this thesis we consider complete Merkle trees, i.e., trees in which n is an integer power of 2. A complete description of algorithms over Merkle Trees is presented in Appendix A.

2.2 Blockchain

A blockchain is a growing list of records, called blocks, that are securely linked together using cryptography [41]. Each block contains some data, a timestamp, and a reference to the previous block. The form in which the data is represented can vary depending on the specific design choices of the considered blockchain, however, generally, it is represented as the root of the Merkle tree having the transactions as its leaves.
Each block contains a reference to the previous one: usually it's a hash. In this way, each block is connected to the previous one shaping the actual list of records. This guarantee that any

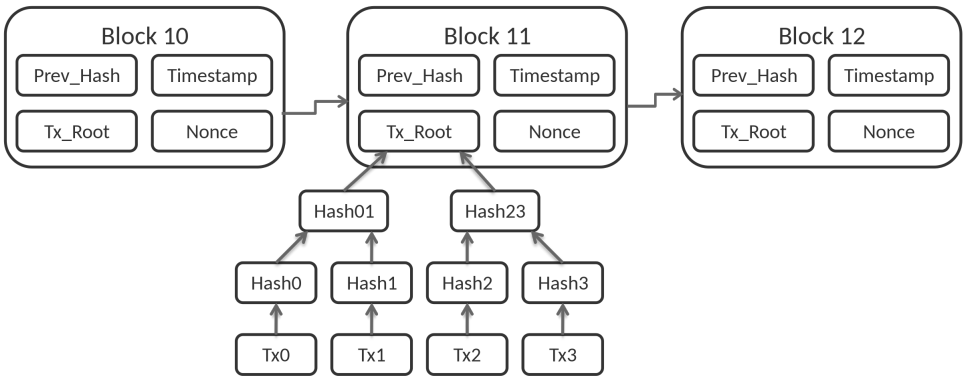

Figure 2.1: Bitcoin blockchain structure

block inside the blockchain can't be altered retroactively without the alteration of all subsequent blocks. Consequently, given a block, the higher the number of blocks introduced after him inside the chain, the more difficult will be for a possible attacker to modify it. An illustrated visualization of a typical blockchain structure is given in Fig. 2.1.

Blockchain is integrated into multiple fields, but up to now, its main usage is as a distributed ledger for cryptocurrencies such as Bitcoin [42]. The principal property of blockchain is avoiding infinite reproducibility for a digital asset, solving the double-spending problem [43].

2.2.1 Consensus

As explained before, the blockchain can be represented as a list of records, where every single unit is called a block, and where blocks are connected through a cryptographic link. For this reason, the way the members of the network decide which block to introduce inside the blockchain represent a crucial point for the whole system.
A consensus mechanism is a set of rules and incentives that allow nodes to agree on the state of the network. In general, the consensus problem within the blockchain is closely related to the order of transactions, i.e., participants must agree on a single history of the order in which they were entered [42].
Typically, network participants working to introduce new nodes within the blockchain itself are referred to as *miners*.

Proof of work A first approach to solve the aforementioned problem was proposed by Santoshi Nakamoto in his Bitcoin whitepaper [42]. As stated in the whitepaper, the consensus

problem to be solved corresponds to the design of a distributed timestamp server in a peer-to-peer network.
The proposed consensus mechanism is called *proof-of-work*. It consists of looking for a value that, when hashed, starts with a certain number of zero bits. As the number of zeros increases, the complexity of finding the value increases exponentially [42], while the proof always consists of a single hash. This allows the protocol to increase the difficulty of finding a correct block, with the increase of the computational power provided by the miners. The found value is introduced inside the block in a field called *nonce*.
The miner wishing to introduce new transactions in the chain must:

1. create a new empty block;
2. fill in the data field and hash of the previous block;
3. begins brute-forcing the value of the nonce to satisfy the condition on the number of zeros at the beginning of the hash.

Consequently, only the block that was created and transmitted faster will be added to the chain. If two miners broadcast different versions of the next block at the same time, or if the broadcasting of one block is significantly slow, then different nodes could consider as valid different chains.
In this case, each participant in the network will continue working on the last received node, and the tie will be broken when the next block is created. As a general rule, the chain that is considered valid is always the longer one. This means that the transactions included inside the shorter chains but not inserted inside the longer ones will not be contained inside the final ledger.
To be sure that a transaction has been executed correctly, and so introduced inside the blockchain, is appropriate to wait for a *confirmation time*, that is a certain number of blocks successive to the one to which the transaction belongs. In this way, the probability of a tie between chains is lowered.
To convince miners to support the network and participate in the solving of this computational problem, an economic incentive is included in every new block mined. The issue of the incentive allows also for solving the coin distribution problem inside the network: coins are mined and obtained as rewards by miners, without the need for any central authority.
The incentive is not the only economic reward miners receive, they also earn a fee on each transaction included inside blocks mined by them. The coins minted at each block are progressively

reduced, following a process called *halving*, up to the point where transaction fees will represent the only reward for miners.
Every user inside the network can use its computing power to race to solve the next block. If a user or group of users is able to control more than 50% of the computational power of the whole network, then we talk about a 51% attack. This group can lead to an altered blockchain which is theoretically accepted by the network since the attackers would own most of it [44]. Successful attackers can mine blocks faster than the other parties inside the network and so gain the ability to block new transactions from being confirmed, change the ordering of new transactions, and partially rewrite recent blockchain transactions.

PROOF OF STAKE Proof-of-stake is another type of consensus mechanism used inside blockchains. While in proof-of-work miners risk their computational power trying to identify the next block of the chain, in this case, validators, i.e., miners, explicitly stake capital. This capital acts as collateral since it can be lost by the validator if he behaves in the wrong way or does not participate actively inside the network [45].
Each validator has to check that new blocks are being propagated over the network, check their validity and occasionally create and propagate new blocks himself.
Proof-of-stake comes with several improvements to the proof-of-work system [45]:

- better energy efficiency, since the number of heavy computational operations is considerably lower than on proof-of-work [46];
- lower barriers to entry, which lead to a higher number of nodes and reduced centralization risk. Since there is no need for elite hardware to stand a chance of creating new blocks then more users of the network can participate in the new consensus mechanism;
- economic penalties for misbehavior make 51% style attacks exponentially more costly for an attacker compared to proof-of-work.

This type of consensus mechanism is already used by various blockchains, such as Ethereum [47].

SMART CONTRACT A smart contract is a computer program or a transaction protocol that is intended to automatically execute, control, or document legally relevant events and actions according to the terms of a contract or an agreement [48]. Smart contracts allow the execution of programs which state is stored inside the blockchain. They also allow to bind money transfers to program code. Each smart contract is characterized by an internal state and a set of public

functions that are callable by users. Each function call is associated with a transaction, which can also be used to send money to the smart contract itself. The economic incentive provided by the consensus mechanism is at the base of the execution of smart contract transactions by miners. Smart contract execution is characterized by fees proportional to the number and type of instructions executed.
Depending on the considered blockchain, the structure and possibilities achievable by smart contracts change. The usual point of reference for this technology is the Ethereum blockchain, which uses an internal unit called *gas* for fee management, and *Solidity* as a scripting language. Smart contracts are executed inside the Ethereum Virtual Machine (EVM) [47], but before that, they are compiled to low-level EVM bytecode. Before the bytecode, the Solidity smart contract is translated in OP_CODES, each of which refers to an instruction inside the EVM. The number of instructions inside the EVM is limited, and each of them has a gas cost associated with it.

2.2.2 Fair Exchange

In fair exchange protocols, we identify two main parties: a receiver and a sender, who do not trust each other and want to exchange a digital good, e.g., a file, for a monetary value. The sender wants to sell the file, while the receiver wants to pay for it. The result of such a protocol is that either both parties receive their expected output or none do. It has already been proved that fair exchange is not possible without the usage of a Trusted Third Party (TTP) [49], which can be very costly or might not be available at all. In this thesis, we consider as TTP a blockchain, and leverage technologies like smart contracts to achieve and implement this type of protocol.
In much fair exchange protocols [30, 36] the main idea is to let the two parties agree on information that can subsequently be used in case of misbehavior by one of them. This information is called *description*, and it is usually derived from the content of the file to be exchanged. Depending on the context of use, such a description can be not useful since the receiver has no idea about the actual content of the file. In these cases, sampling of files can be used for description [30, 31, 32, 33, 34, 35], more specifically the description is composed as a subset of these sub-files. We refer to this type of fair exchange protocol as *sampling-based*, and it usually requires the two parties to jointly define the description together. Initially, the sender shares an encrypted version of the file. Then, the description is jointly chosen: sender and receiver agree on a small subset of the sub-files. Each of the items composing that description is subsequently shared in

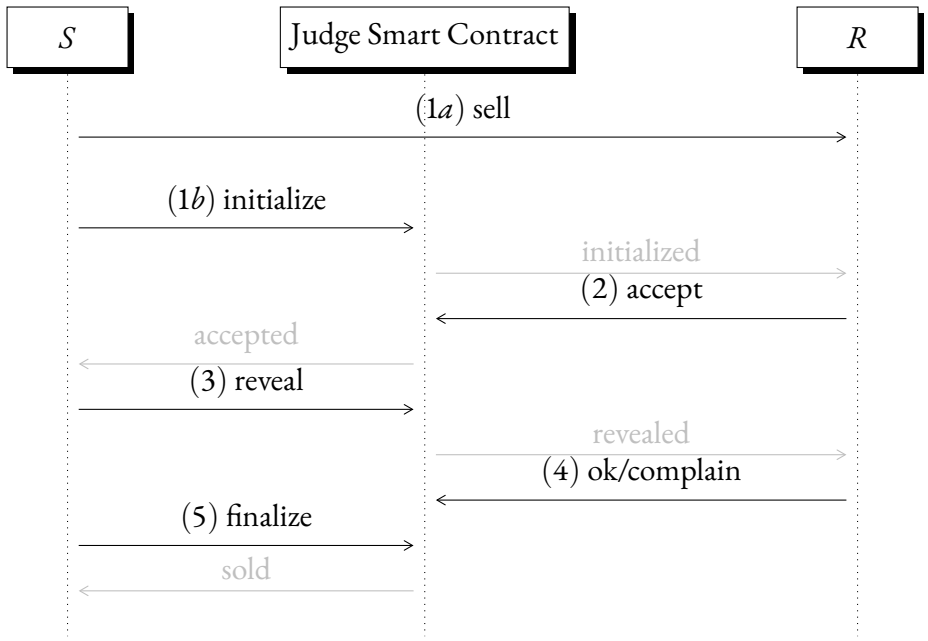

Figure 2.2: FairSwap: global model

clear text by the sender. During the next phase, the receiver sends the payment to the system in exchange for the key used for the encryption of the file. If the shared key is wrong, then the agreed description is used to identify the party that misbehaved and economically penalizes it. We briefly treat FairSwap [36] and FairDex [30] fair exchange protocols, with a specific focus on the misbehavior detection ideas introduced by them. Both protocols are *optimistic* fair exchanges [50], which means that the TTP, i.e., the blockchain, is heavily used only if one of the two parties misbehaves. At the beginning of the protocol, the two parties agree on a description: common knowledge representing the file's content. If one of the two parties behaves in a malicious way, the other one can provide the TTP a proof of it, called proof of misbehavior (PoM). Usually, this proof is cryptographically linked with the description of the file, allowing the TTP to check its validity.

FairSwap This protocol assumes that both parties know in advance the hash of the digital good x, which is represented as a set of units called sub-goods. As visible in Fig. 2.2, the seller S has to initialize the exchange, by first sending the encoding z of the good to the receiver (step $1a$), and then sending a commitment to the judge smart contract (step $1b$).

The encoding z is composed using the Alg. 1 by the encryption of all the nodes of the Merkle tree generated by $Mtree(x)$ by using a symmetric encryption scheme.

Algorithm 1 Encode

Input: $x = (x_1, ..., x_n), k$
  $M \leftarrow MTree(x)$ ▷ Compute all the Merkle tree nodes
  for $m_i \in M$ **do** ▷ Encrypt all the nodes
    $k_i \leftarrow H(k||i)$ ▷ Sub-key as a hash of master key and index of the node
    $z_i \leftarrow E(k_i, m_i)$
  end for
Output: $z = (z_1, ..., z_m)$

The judge smart contract is provided with a cryptographic commitment of the master key, i.e., $H(k)$, and with the description of the good, i.e., $root(MTree(x))$ and $root(MTree(z))$.
Once the buyer transfers the agreed amount of money to the smart contract (step 2), the seller publishes the key (step 3), which integrity is checked using the previously published key commitment. If both parties behaved correctly then the sender can withdraw the money and the protocol ends. Otherwise, if the sender behaves maliciously, the receiver has a certain amount of time to provide proof of the misbehavior and receive his money back. Passing the encoding Z to the algorithm 2, the receiver is able to identify at least one node in the Merkle tree that is wrong. The proof of misbehavior sent to the smart contract is composed of the wrong node, its sibling, and its expected parent. The sender is also required to attach the proof that the considered node and its sibling belong to the encoding z; this is done by providing the corresponding Merkle tree paths. The smart contract then executes the algorithm 3 to verify the provided proof of misbehavior.

The protocol achieves constant on-chain complexity in the optimistic case. In the pessimistic case, the time complexity depends on the logarithm of n and on the size of each sub-good z_i, since the PoM requires two of them. Regarding the off-chain performances, the encoding z adds a certain communication overhead, since other than the n encrypted sub-goods it is necessary to send also the internal nodes of the Merkle tree. Even assuming the encrypted file is to be exchanged using a private communication channel between the two parties, in the pessimistic case at least two sub-files are shared with the smart contract, and so published. If the procedure is repeated multiple times, an eavesdropper could learn a relevant portion of the file.

FairDex It is a sampling-based fair exchange protocol, which considers the file x as a set of sub-files. Instead of using the hash of the file, a jointly agreed subset of the sub-files is used to define the description. Its idea is similar to FairSwap [36], but with some changes that allow

Algorithm 2 Extract

Input: $z = (z_1, ..., z_m), k, h = Root(Mtree(x))$
  for $i \in [n]$ **do** ▷ Decrypt the leaf nodes
    $x_i \leftarrow D(k_i, z_i)$
  end for
  $Z \leftarrow MTree(z)$
  for $i \in [n+1, ..., m]$ **do**
    $x_i \leftarrow H(x_i^l, x_i^r)$
    if $(x_i \neq D(z_i, k_i)) \vee (i = m \wedge x_i \neq h)$ **then** ▷ Wrong decrypted node or root
      $l \leftarrow index(z_i^l)$ ▷ Index of left child of z_i
      $r \leftarrow index(z_i^r)$ ▷ Index of right child of z_i
      $i_{proof} \leftarrow Mproof(Z, i)$ ▷ Proof that z_i belongs to Merkle tree Z
      $l_{proof} \leftarrow Mproof(Z, l)$ ▷ Proof that z_l belongs to Merkle tree Z
      $\pi \leftarrow (i, l, z_i, z_l, z_r, i_{proof}, l_{proof})$
      Output: $((x_1, ..., x_n), \pi)$
    end if
  end for
Output: $((x_1, ..., x_n), \perp)$

Algorithm 3 Judge

Input: $\pi = (i, l, z_i, z_l, z_r, i_{proof}, l_{proof})$
  if $Mverify(i, z_i, i_{proof}, Root(Z)) \neq 1$ **then**
    Output: *false* ▷ Mverify of i_{proof} returns false
  end if
  if $(Mverify(l, z_l, l_{proof}, Root(Z)) \neq 1) \vee l_{proof}[0] \neq z_r)$ **then**
    Output: *false* ▷ Mverify of l_{proof} returns false or it does not start with z_r
  end if
  $x_i \leftarrow D(z_i, k_i)$
  $x_l \leftarrow D(z_l, k_l)$
  $x_r \leftarrow D(z_r, k_r)$
Output: $x_i = H(x_l, x_r)$ ▷ Check if s_i has the right value

reducing the execution cost in the pessimistic case.

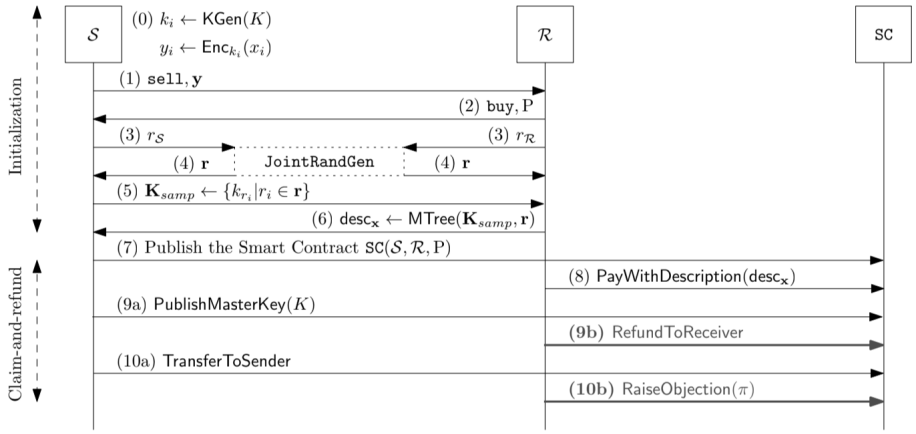

Figure 2.3: Illustration of FairDex protocol. Honest and malicious executions are represented with (9a)-(9b) and (10a)-(10b).

Even in this protocol, the file is encrypted using the *Encode* algorithm (Alg. 1) used in FairSwap. The first part of the protocol is executed off-chain: the sender S shares with the receiver R an encrypted version of the file (*step* 1). The two parties agree on the description to use, and S shares with R the sub-keys to decrypt each sub-file contained in the description (*step* 2 up to 6). In this way, R can check them and decide if to continue the exchange process or not. Unlike FairSwap, the Merkle tree commitment shared with the smart contract is not generated starting from the sub-files, but from the sub-keys used for the encryption process. Considering the set r of indexes of sub-keys representing the description, then the Merkle tree has $[k_{r_1}, r_1, \ldots, k_{r_s}, r_s]$ as leaves, as visible in Alg. 4. This allows to achieve lighter proofs of misbehavior, which size does not depend on the sub-files, but only on the sub-keys.

After sharing the master key, if both parties behave correctly then the protocol stops, otherwise, the pessimistic case is considered. In particular, knowing the master key, R is able to generate all the sub-keys and decrypt the sub-files.
If R identifies a wrong sub-key during this process, he can share the proof of misbehavior with the smart contract. The goal of R during this phase is to prove that one of the sub-keys used for the description can't be generated using the shared master key k.

Algorithm 4 DescGen

Input: $k = (k_1, ..., k_n), r, s$ ▷ Set of keys k, random seed r, size of description s

$r \leftarrow GenIndex(r, n, s)$ ▷ Set of s distinct indexes in $[1, n]$

$K_{samp} \leftarrow \{K_{r_i} | r_i \in r\}$ ▷ Set of selected sub-keys

if $Verify(K_{samp})$ **then**

$desc_x \leftarrow MTree(k_{r_1}, r_1, ..., k_{r_s}, r_s)$ ▷ Merkle Tree description

end if

Output: $(desc_x, K_{samp}, r)$

The proof of misbehavior is then composed of:

- the index of the wrong sub-key
- the right sub-key used for the description
- the Merkle path that proves that the sub-key belongs to the description, generated with the Alg. 2.

The verification of the proof of misbehavior is similar to Alg. 3 but considers that the leaf of the tree is composed of sub-keys and corresponding indexes.
This protocol allows achieving a logarithmic complexity in the pessimistic case, avoiding having proof of misbehavior depending on the size of the sub-files and removing the communication overhead that characterizes FairSwap. No assumption is done regarding the knowledge by the two parties of the hash of the file.

Security Properties The main security properties of a fair exchange protocol [30, 36, 51, 52, 37] are:

- *Correctness*: if both parties behave correctly, then R learns the file content, while S receives the requested amount of money p.
- *Termination* (or *Timeliness*): if at least one party is honest, then the protocol terminates in a finite number of rounds and unlocks all the coins from the contract.
- *Sender Fairness*: an honest sender S is guaranteed that no information about x is shared, if he doesn't receive the requested amount of money.
- *Receiver Fairness*: an honest receiver R is guaranteed to pay the price of the file iif he is able to learn correctly the content of the file.

- *Security against grieving*: in case of a dispute, the cheating party always compensates the cheated party for any transaction fees paid during the dispute process.

A probabilistic definition of fairness can be introduced by defining an arbitrary lower bound on the probability to violate fairness [53, 54, 55].

3 Protocol

Our proposal aims to allow reporters to share and sell information to newspapers, in a confidential and anonymous way, providing a measure of the credibility of the shared information. We take some ideas from existing sampling-based fair exchange protocols [36, 30], to extend them to manage multiple users, sale operations, and confidentiality between seller and buyer. We design the system in order to cover in a complete way the whole sharing process, not assuming any private or secure direct communication channel between the two parties.
Since this work represents the starting point of a longer research, this section starts with all the considered assumptions, and among them some definitions that we leave to better formalize in future works. We present the considered threat models and properties that we want to achieve, for then describing the actual protocol, followed by the corresponding security and privacy proofs.

3.1 Assumptions

Let's define A as the media worker's space, where each $a \in A$ is characterized by a public/private key pair $(pk_a, sk_a) \in P \times S$. Let's call T the set of all possible newspaper article topics. We assume to have a mapping $I : T \times P \to [0, 1]$, defining for each topic $t \in T$ and media worker a, identified by its public key pk_a, the interest and expertise level of a w.r.t. t. Starting from I, we can design a function to compute, given a set of n media workers, each with its public key, a measure of its correlation with a specific topic, i.e., $f : T \times P^n \times I \to O$, with O output space.

The definition and design of T, I, f are out of the scope of this thesis, and we leave it as future work.
The symbol || is used in the thesis to indicate the concatenation of the value of two variables. Table 3.1 contains the main notations used for the protocol.

Table 3.1: Notation table

Symbol	Definition
A	set of media workers
B	set of newspapers
T	set of topics
I	mapping between topic and media worker
f	correlation function
x	file to exchange
z	encrypted file
$desc_x$	description of file x
K_{samp}	sample of sub-keys
s	number of samples to consider, i.e., $\lvert K_{samp} \rvert$
I_{samp}	indexes of the sampled sub-keys
r	randomness
H	hash function
E_s, D_s	symmetric encryption and decryption functions
E_as, D_as	asymmetric encryption and decryption schemes
$Sign, Verify$	ring signature sign and verification functions
S, P	private and public key spaces
$B_{channel}, P_{channel}$	blockchain and public communication channels

3.1.1 Communication model

We consider a synchronous communication model [56, 57, 58], where the time unit is defined as round and each party is aware of the current round. The protocol is executed following these rounds, and, assuming a synchronous model, only one party can operate during each of them. We assume the presence of a communication channel $B_{channel}$ between the blockchain and each party, used to interact with the blockchain itself. Regarding the communication between the reporter and newspaper, no direct, confidential, or authenticated channel is required, but we

assume the presence of a public channel $P_{channel}$ where each party has reading and writing privileges, e.g., a website or shared directory.

3.2 Threat Model

In this work, we examine multiple threat scenarios, depending on the behavior of the involved parties. Fixing a specific reporter-newspaper information exchange then we can observe the following scenarios.

Malicious reporter, honest newspaper The reporter shares the sampling-based description of the file but then he sends the wrong encrypted file or decryption key.

Honest reporter, malicious newspaper The reporter follows the protocol providing the correct encrypted file and decryption key to the newspaper. The newspaper behaves in a malicious way, trying to create a wrong PoM and initiating the refund routine even after having received the correct information.

Eavesdropper *Passive* attacker, which gets information by observing the protocol execution between the reporter and newspaper. The eavesdropper wants to get additional information about the shared data or about one of the parties.

3.2.1 Properties

The proposed protocol satisfies some *anonimity* and *fair exchange* properties. We consider a single execution of the protocol, so a single exchange between a specific reporter and newspaper. The fair exchange properties that we consider are the ones defined in Sec. 2.2.2, that using the notation presented in Table 3.1, become:

- *Correctness*: if both parties behave honestly, then the receiver obtains a file o s.t. $desc_o = desc_x$, while the sender earns the agreed amount of money.
- *Probabilistic Receiver Fairness*: if the receiver does not obtain the correct file, then we can set an arbitrary lower bound on the probability of the sender earning the agreed amount of money.
- *Sender Fairness*: if the sender does not earn the pre-agreed amount of money, then the receiver cannot obtain any useful information about x, except for the samples related to its description, i.e., x_i s.t. $k_i \in K_{samp}$.

- *Timeliness*: every party that behaves honestly terminates the protocol in a finite and capped amount of time.
- *Grieving Security*: in case of a dispute, the cheating party must always compensate the cheated party for any transaction fees paid during the dispute process;

Additional properties are:

- *Confidentiality*: if at least the sender behaves honestly, then no information but the samples used for the description, has to be visible to an external observer;
- *K-anonimity*: given a group of k reporters and a specific information exchange, the probability of a specific reporter to have shared the information has to be uniform in the size k of the group.

3.3 FairDrop

We can define FairDrop as a multi-buyer fair exchange protocol with anonymity properties. FairDrop is a sampling-based fair exchange, and for this reason, the information to be exchanged is seen as a set of samples. We refer to it as a *file*.
A smart contract *Judge* is used to mediate the two parties during the process, providing functionalities like escrow [59] and misbehavior verification, allowing them to fairly resolve disputes when they disagree. Since *Judge* is able to manage sales executed by multiple reporters and newspapers, then we assume *Judge* to be already deployed on the blockchain. We do not require any user of the service to manage the deployment operations and costs while executing the protocol. Each exchange is composed of two main phases:

- *initialization phase*: executed by the reporter to define the description of the file he wants to sell, and publish the required metadata.
- *sale phase*: handles the exchange and the possible misbehavior by one of the two parties.

The protocol considers two different behavior models by its participants: an *optimistic* one, when both parties follow the protocol, and a *pessimistic* one, when at least one of them misbehaves.

3.3.1 Initialization Phase

For simplicity of explanation, we start fixing a reporter, or in general media worker, $a \in A$. We assume a to hold some information x that he wants to sell for a fixed price p. We consider x as a set of $|x|$ sub-files, i.e., $x = \{x_i\}_{i=1}^{|x|}$.
Since a possible buyer does not know the content of x in a detailed way, then the two parties have to agree on some common knowledge, called *description* of x. Being the protocol a sampling-based one, then the description of the file is represented by a subset of its sub-files.
During this phase, as shown in Fig. 3.1, the reporter a interacts with the smart contract *Judge* and the public communication channel $P_{channel}$ to share some information regarding the file x and allow any interested newspaper to request to buy the file itself. Initially, a generates a master key k, that he uses to compute a sub-key for each sub-file, each sub-key k_i is defined as:

$$k_i = H(K||i) \quad \forall i \in \{1, ..., n\}. \tag{3.1}$$

Then a encodes x by encrypting each sub-file x_i with the corresponding sub-key k_i and a symmetric encryption scheme:

$$z_i = E_s(k_i, x_i) \quad \forall i \in \{1, ..., n\}. \tag{3.2}$$

In this way a obtains $z = [z_1, ..., z_n]$, representing the encrypted version of x.
Since each sub-key k_i is computed starting from the master key k and its index i and being E_s symmetric, then a possible buyer would just need k to completely decrypt z. For this reason, k is called *master key*.
In a sampling-based fair exchange protocol, the description is usually derived from the actual content of the file, and so by a subset of its sub-files. As already presented in other protocols [30], instead of using a subset of sub-files, it is useful to use a subset of sub-keys K_{samp}, in order to reduce the cost of execution in the blockchain. While usually [30] the randomness to select the samples in K_{samp} is given by the interaction between seller and buyer, in our protocol we assume it to be generated directly from the blockchain. This allows the reporter a to execute the initialization phase even in absence of a ready buyer. To identify the indexes of keys to be sampled, starting from the randomness r we use the algorithm 1, which is a modified version of Fisher and Yates' shuffle algorithm [60]. The number of indexes to generate, i.e., $|K_{samp}|$, represents the portion of the file that the reporter wants to reveal for free. This value is decided by the reporter itself during this phase.

Algorithm 1 GenIndexes

Input: s = $|K_{samp}|$, n, r ▷ s: #samples to take, n: # of sub-files, r: randomness
Rand.seed(r) ▷ Init a PRNG using the obtained randomness r
indexes = [0, 1, ..., n-1] ▷ Array of all possible key indexes
i = n-1, $I_{samp} = \{\}$
while i > n-s-1 **do** ▷ Fisher and Yates' shuffle algorithm, to get a random permutation
$j \leftarrow$ Rand.nextInt(i) ▷ Random integer j s.t. $0 \leq j \leq i$
$I_{samp} \leftarrow I_{samp} \cup \{indexes[j]\}$ ▷ Save selected value
swap(indexes[i], indexes[j]) ▷ Swap indexes[j] with indexes[i] to avoid repetition
$i \leftarrow i - 1$
end while
Output: I_{samp}

In this way, in $O(s)$ time, and $O(n)$ space, we are able to obtain a sequence of size s, containing the indexes to be used for the sub-keys sampling. At this point K_{samp} is defined in the following way:

$$K_{samp} = \{k_i\} \quad i \in I_{samp}. \tag{3.3}$$

Each party knowing K_{samp} is able to obtain:

$$x_i = D_s(k_i, z_i) \quad \forall k_i \in K_{samp}. \tag{3.4}$$

Starting from K_{samp}, we can define the description *desc* of x as the root of the Merkle tree having as leaves $(k_i, i) \in K_{samp}$:

$$desc = Root(MTree(k_{i_1}, i_1, ..., k_{i_{|K_{samp}|}}, i_{|k_{samp}|})). \tag{3.5}$$

With the term *depth* we refer to the height of that Merkle tree. Assuming the number of leaves to be a power of 2:

$$\#leaves = 2 * |K_{samp}| = 2^h, \quad h \in \mathbb{N}, \tag{3.6}$$

then we can consider $depth = h$.

SMART CONTRACT INTERACTION During the initialization phase, the reporter a has to interact with the *Judge* smart contract, following the scheme shown in Fig. 3.1.

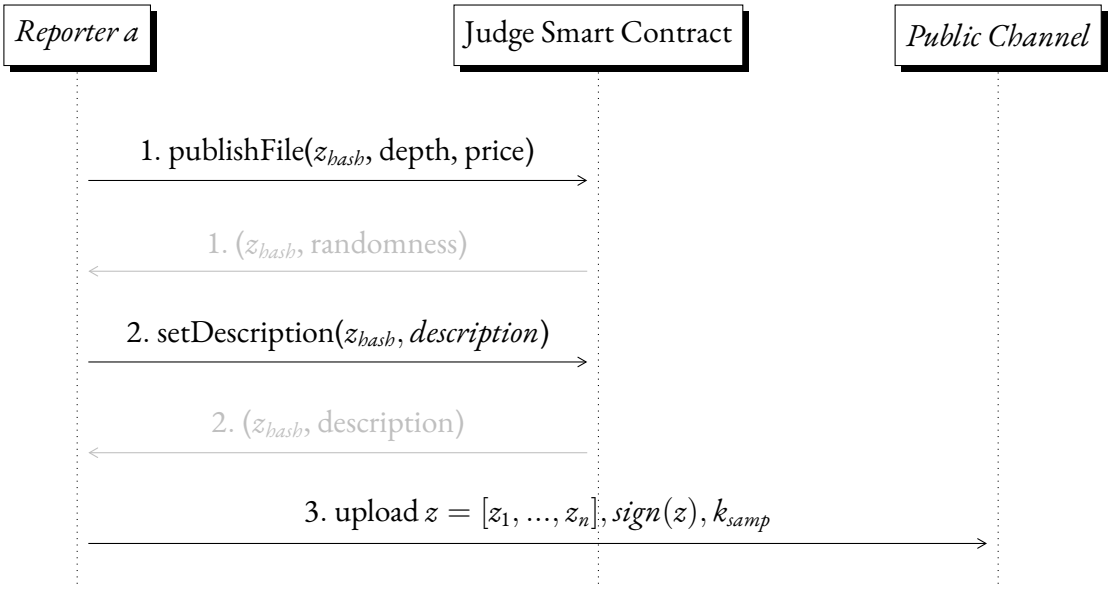

Figure 3.1: FairDrop: initialization phase scheme

After computing z, the reporter a has to share:

- a cryptographic commitment of the encrypted file z, represented by $H(z)$ in our case;
- the *depth* of the expected description Markle tree, strictly related to the $|K_{samp}|$ decided by the reporter;
- the *price* the reporter wants to share its information for.

The previous information is shared with the smart contract *Judge* by calling the method `publishFile`. During the execution of this method, the smart contract generates and publishes a random value r that can be used as input for Alg. 1 to identify the index of the sub-keys that will compose K_{samp}. The description *desc* is then computed following Eq. 3.5, and shared with the smart contract calling the method `setDescription`.
To conclude the initialization phase the reporter a has to share through the public communication channel P:

- the encoded file z;
- the sampled sub-keys K_{samp} with the corresponding indexes;
- a public key pk, selected by a, and different from pk_a;

- the ring signature of the encrypted file z, obtained by providing to the algorithm *Sign*, the private key sk_a, the public keys of the other members of the group, and the encoded file z;
- some metadata regarding x, such that a description of its content, a title, and a topic $t \in T$ associated with it.

As mentioned, among the information that the reporter publishes, there is the signature of z. Being it a ring signature, the reporter a has to first choose a group of members from the set A. The choice of that group, containing also a, has to be done in order to maximize the value of f w.r.t. the group and the topic $t \in T$ of x. This would incentive a possible newspaper $b \in B$ to proceed with the *sale phase* since the information would seem more credible.

3.3.2 Sale Phase

Each newspaper $b \in B$, interested in buying and obtaining the file x, has to execute the *sale phase*. During this phase, the newspaper checks the data shared by the reporter in the *initialization phase*, both in the blockchain, i.e., through $B_{channel}$ and in the public one, i.e., $P_{channel}$. Examining this data, b actually decides if to continue with the information exchange or stop the protocol.
If the exchange continues, then b has to interact with smart contract *Judge*, to record his interest in the file, and to perform the actual exchange. During the whole phase, no direct communication involves a and b, which communicate only through the smart contract. At the end of this phase:

- in the *optimistic* case, the reporter a receives the payment, while the newspaper b pays the agreed amount of money and he has full access to x;
- in the *pessimistic* case, the reporter a doesn't receive the payment while the newspaper b receives back his money without having the possibility to learn more about x.

The sale phase represents the interactive part of the protocol since requires both parties to execute computations and interactions with the smart contract. For this reason, we can split the phase in *rounds* and define a maximum round time duration R_{time}. If a party doesn't share with the smart contract the data required for the considered round in time, i.e., before R_{time} is reached, then we assume the party to abort the protocol. Defining in this way R_{time}, allows to achieve *timeliness* property in the protocol, as explained later in Sec. 3.3.3.
In the following paragraphs, we analyze in a more detailed way all the steps that compose this phase.

Initial checks As introduced previously, before continuing with the actual exchange, the newspaper b examines the data shared by the reporter during the initialization phase, to check everything is correct. In particular, b has to:

- compute the hash of z shared in $P_{channel}$, and verify that a corresponding randomness value has been shared by the smart contract, as result of the `publishFile` method call;
- the newspaper b computes the Merkle tree having as leaves the element in K_{samp} shared by the reporter a in $P_{channel}$ and compares its root with the description published using the `setDescription` method of the smart contract;
- verify the correctness of the ring signature shared by a, i.e, $Sign(z)$, by using the *Verify* algorithm. We assume the signature to contain the list of public keys of the members of the group.
- use the shared sampled sub-keys, i.e., k_{samp}, to decrypt the corresponding sub-files and check that they respect the expectations of b. To do so, we can assume b to consider the shared topic t of x, and the other metadata, such as the title and description of the file. As described in Sec. 3.3.3, the size of K_{samp} influences the *probabilistic receiver fairness* property, and for this reason, has to be considered by b before continuing with the exchange.

If all the previous checks are successfully executed, then before proceeding b verifies the composition of the ring signature. By the properties of ring signatures, b cannot understand which particular member of the ring signed the file, but he knows the composition of the ring, and in particular the list of the public keys associated with the members of it. At this point, using f, function introduced in 3.1, b can obtain a value representing the correlation between the topic of the file x and the ring. This value can be used also as a measure of the credibility of the information exchanged, and as a parameter by b to decide if to continue with the actual exchange or not.

Optimistic case If the newspaper b decides to proceed with the exchange, he formalizes his decision by calling the `buy` method of the smart contract and paying its price. In order to allow the reporter a to subsequently share the master key in a confidential way, b randomly select a value *secret*, and provides to the `buy` method:

- $H(secret)$: the hash of the secret, which represents a cryptographic commitment of it;
- $E_{as}(pk, secret)$: the ciphertext obtained by encrypting the secret using the public key pk, shared during the initialization phase.

As shown in Fig. 3.2, the reporter a, using its private key sk_a, can decrypt the ciphertext and obtain the *secret*. The reporter also checks the correct behavior of b, by comparing the received hash of the secret, and directly computing it with the decrypted data. If one of these two steps fails, then a aborts the protocol by letting pass R_{time}.

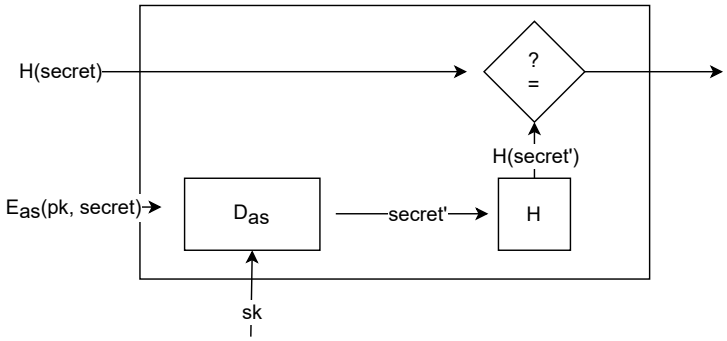

Figure 3.2: Sale phase, optimistic case - secret verification

Once called, the method `buy` publishes an `id` associated to the current file sale between a and b. The id is necessary for the smart contract to manage multiple file exchanges between different parties, and it is required to provide in each subsequent call of a method of the smart contract. If the reporter decides to continue with the protocol, he can compute:

$$\hat{k} = k \oplus secret, \tag{3.7}$$

and share it by calling the method `publishKey` of the smart contract. To avoid grieving attacks on the receiver, a has also to send a certain amount of money, called *collateral*, while calling this method. Since the newspaper b knows *secret*, he can compute the master key as:

$$k = \hat{k} \oplus secret, \tag{3.8}$$

and derive the sub-keys as defined in Eq. 3.1, to fully decrypt z.
Assuming the shared key to be correct, the protocol ends after a certain timeout, i.e, R_{time}. After it, the reporter calls the `withdraw` method and receives the price of x plus the collateral back.
A representation of the on-chain part of this phase is shown in Fig. 3.3. In particular, while the call 1. is always done by the newspaper, depending on the data provided by him, i.e., hash and encryption of the secret, two different execution flows are identified, in the figure: $(2a, 3a)$ and $2b$.

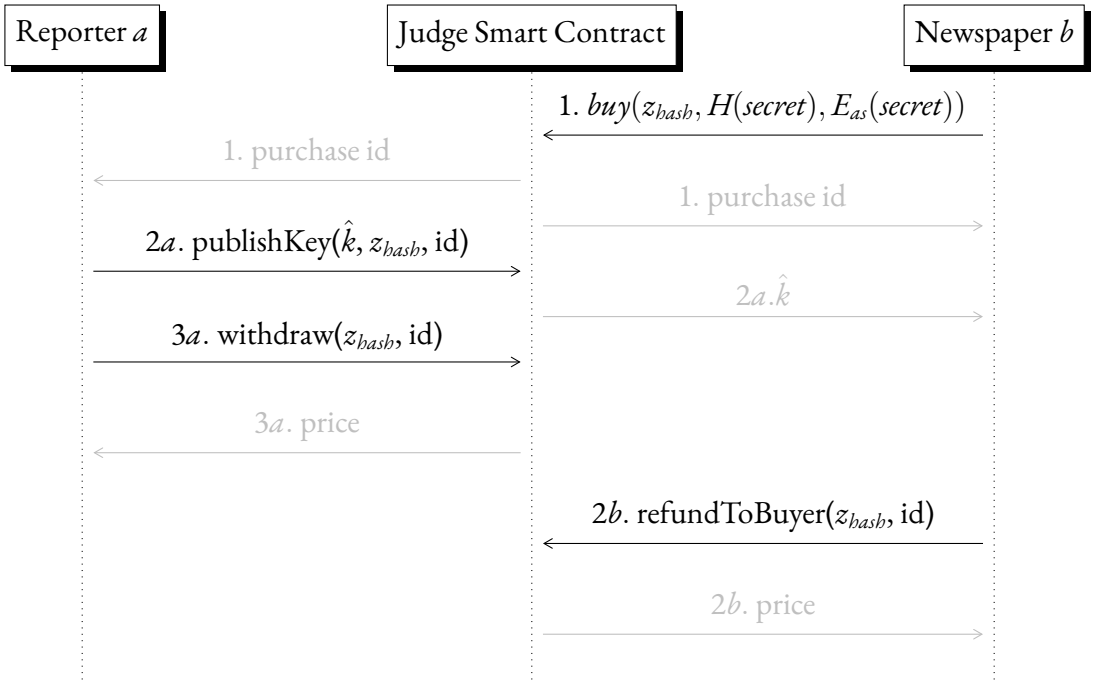

Figure 3.3: FairDrop, sale phase: optimistic case

PESSIMISTIC CASE If the reporter a shares a wrong encrypted master key $\hat{k}$, then b can prove it by using the previously published information. In particular to define a PoM, b has to find a sub-key k_i belonging to K_{samp} that it is not possible to generate using $\hat{k}$. It means that for k_i it holds:

$$k_i \neq H((\hat{k} \oplus secret) \mid\mid i), \qquad k_i \in K_{samp}. \tag{3.9}$$

Fixing such k_i, a valid PoM would be composed by:

- the index i of the sub-key to consider;
- $\pi_i = Mproof(Mtree(k_{samp}), i)$: used to prove that the sub-key belongs to K_{samp}, and that it was used to compute the description *desc*;
- the *secret* previously chosen by b, in cleartext.

Once created the PoM, the newspaper b can publish it by calling the `raiseObjection` method of the *Judge* smart contract. The `raiseObjection` logic is shown in Alg. 2: if the PoM is valid, then b receives back his money plus the *collateral*, in order to refund b for the fee costs paid to call the `raiseObjection` method.

Algorithm 2 raiseObjection

Smart Contract State: $secret_h$, $desc$, $\hat{k}$ ▷ $secret_h$: hash of secret, $\hat{k}$: $k \oplus secret$
Input: i, k_i, π_i, secret ▷ i: index, k_i: sub-key, π_i: Merkle path of k_i in K_{samp}

```
if H(secret) ≠ then                                   ▷ Check secret commitment
    Output: ⊥
end if
k ← secret ⊕ k̂                                        ▷ Extract the master key
if H(k||i) ≠ k_i then          ▷ Verify that the supplied k_i does not derive from k
    if Mverify(i, k_i, π_i, desc) then                ▷ Verify the correctness of π_i
        send(price + collateral, address)             ▷ Refund the newspaper (price + collateral)
    else
        Output: ⊥ ▷ k_i does not belong to MTree(K_samp), and so the PoM is not valid
    end if
else
    Output: ⊥                  ▷ k_i is derivable from k, and so the PoM is not valid
end if
Output: ⊤
```

To sum up, a representation of the on-chain interaction during this part of the protocol is shown in Fig. 3.4. Inside the diagram two different paths are reported depending on the validity of the submitted PoM:

- $(3a)$: in the case the PoM is evaluated as correct by the *Judge* smart contract, a refund is sent to the newspaper b, that receives both the *price* and the *collateral*, used to cover the expenses needed for the `raiseObjection` call.

- $(4b)$: if no correct PoM is submitted, after R_{time}, the reporter a can call the `withdraw` method and receives back the *collateral* plus the *price* payed by b for the file x.

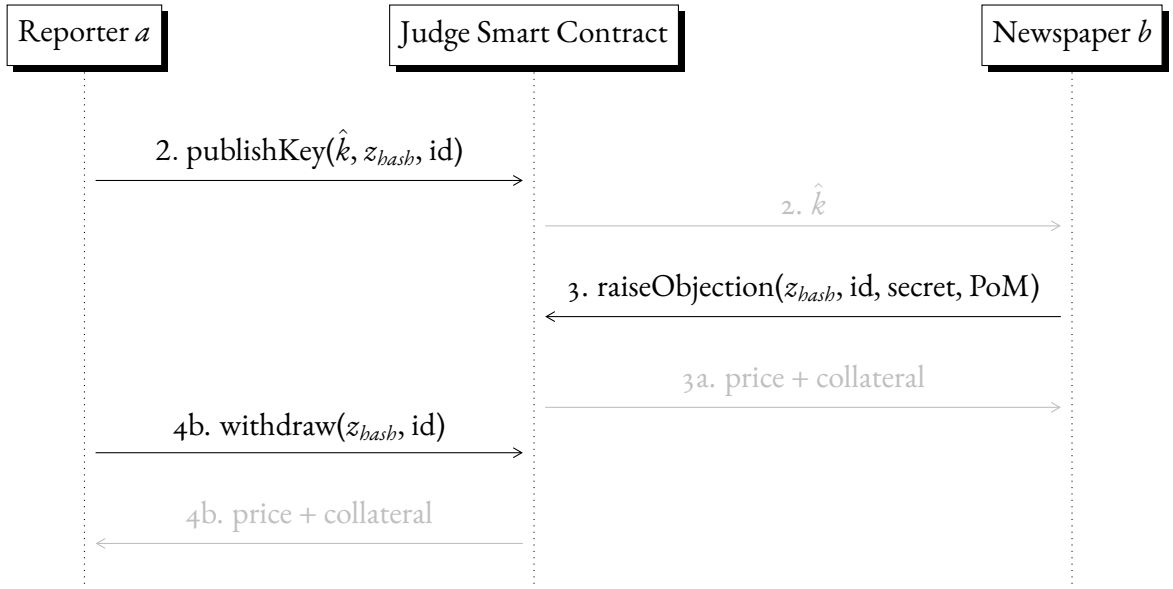

Figure 3.4: FairDrop, sale phase: pessimistic case

3.3.3 Security & Privacy Analysis

In this section, we provide the proofs for both security and privacy properties mentioned in Sec. 3.2.1.

Correctness Protocol *correctness* implies that if both parties behave correctly, and so only the optimistic case is executed, i.e., no PoM is considered, then both of them will receive what they expect: the reporter a obtains the payment, while the newspaper b obtains x.

Proof Sketch. First, the reporter a executes the initialization phase, without any interaction with the other party b. During this phase, the reporter samples the sub-keys to share and generates a Merkle tree having them as leaves. In this way, the description of x, i.e., the root of the tree, is identified. The reporter then shares some information with both the smart contract *Judge* and the public channel $P_{channel}$. Among these a public key pk, K_{samp}, and the description $desc$. During the sale phase, the newspaper b uses pk to share a secret with the reporter, which commitment, i.e. $H(secret)$, is stored inside the smart contract. Assuming both the reporter a and the newspaper b to behave in a correct way: a shares the $\hat{k} = k \oplus secret$, b successfully recovers k, derives all the sub-keys and fully decrypt z.
Since the shared master key is correct, the newspaper b doesn't provide the smart contract with

any PoM, and after a time R_{time}, the reporter a can call the `withdraw` method and receive its payment. □

Lemma 3.3.1. *FairDrop satisfies the correctness property.*

TIMELINESS Protocol *timeliness* implies that each party that behaves honestly terminates the protocol in a finite and capped amount of time.

Proof Sketch. Considering the communication model used in the protocol, we assume each round to have a maximum duration of R_{time}. We analyze and prove the timeliness property only on the sale phase since it is the only one that requires actions by both parties. We split the discussion considering separately an honest reporter a and an honest newspaper b.
Starting with the honest sender a, we can identify the following scenarios:

- b behaves in a malicious way, calling the `buy` method of the smart contract and sharing an encrypted secret that does not correspond to the shared hash. In this case, the reporter a can just wait until the end of the round without taking any action, this would abort the protocol for both parties.

- b does not provides a PoM after the reporter a shared the master key. In this case, the reporter can just wait the end of the round, i.e. R_{time}, and call the `withdraw` function, receiving the payment and ending the protocol.

On the other side, considering an honest receiver b, then:

- if the reporter a does not proceed with the sharing of the master key after the request of b to buy the file, then b can just wait for R_{time} and then call the `refundToBuyer` function, to receive back his money and terminate the protocol;

- if the reporter a shares a wrong key, then b can just provide a valid PoM to the smart contract, in maximum R_{time} time, and terminate the protocol.

□

Lemma 3.3.2. *FairDrop satisfies the timeliness property.*

Probabilistic Receiver Fairness Protocol *probabilistic receiver fairness* implies that if the receiver b does not obtain the correct file, then we can set an arbitrary lower bound on the probability of the sender a earning the agreed amount of money.

Proof Sketch. The sender, i.e. reporter a, can behave in a malicious way and try to break receiver fairness by:

1. during the *initialization phase*, providing a value z_{hash} to the smart contract, and upload in $P_{channel}$ an encrypted file z, with different hash;
2. during the *initialization phase*, providing a description *desc* to the smart contract, and upload in $P_{channel}$ a set K_{samp} that leads to a different value;
3. during the *sale phase*, share a wrong master key.

Considering first the *intialization phase*, scenarios 1. and 2. can be avoided by the receiver b, by executing the *initial checks*, explained in Sec. 3.3.2. In particular, the sharing of information in the $P_{channel}$ requires the reporter a to already have interacted with the smart contract and committed to it the value of the hash and the description. For scenario 1., b can easily compute the hash of the shared file z and compare it to the one saved in the smart contract. While for scenario 2., since Alg. 1 returns always the same set of indexes, given the same value of *randomness*, b can first check the indexes of the shared K_{samp}, and then compute the description and compare it to the one saved in the smart contract.
Considering only scenarios 1. and 2., we are able to achieve deterministic receiver fairness, while for scenario 3. we need the introduction of a measure of probability. In the case of the reporter sharing a wrong master key, b has to identify a key in K_{samp} that is not possible to derive from the shared master key. A probabilistic attack by the reporter a would be to consider $x = [x_1, ..., x_n]$, and so $K = [k_1, ..., k_n]$, with the probability of a sub-key k_i to be wrong equal to p. Since Alg. 1 identifies a set of size $s = |K_{samp}|$ from K, then the probability of success by the attacker corresponds to select s correct elements out of a set composed by $p \cdot n$ wrong elements, and $(1-p)n$ correct ones. The probability of success is:

$$\begin{aligned} \frac{\binom{(1-p)n}{s}}{\binom{n}{s}} = \prod_{i=0}^{s-1} \frac{(1-p)n-i}{n-i} &= \frac{n(1-p)(1-p-\frac{1}{n})...(1-p-\frac{s-1}{n})}{n(n-1)...(n-s+1)} \\ &< \frac{(1-p)^s}{(n-1)...(n-s+1)} \ll (1-p)^s \end{aligned} \tag{3.10}$$

□

Lemma 3.3.3. *In FairDrop, no adversarial sender a can violate the receiver fairness property with a probability more than* $(1-p)^s$ *where p is the probability of a not following the protocol for a sub-key.*

Sender Fairness Protocol *sender fairness* implies that if the sender does not earn the pre-agreed amount of money, then the receiver cannot obtain any useful information about x, except for the samples related to its description, i.e., $x_i \; s.t. \; k_i \in K_{samp}$.

Proof Sketch. For this property, we analyze the amount of information obtainable by the receiver b both during the initialization and sale phases.
Starting from the initialization phase, each receiver b can derive $x_i \; s.t. \; k_i \in K_{samp}$, the *sender fairness* in this case is guaranteed by:

- the *one-way* property of the used hash function, i.e., from $k_i = H(k||i)$, it is computationally infeasible to derive k;
- the usage of a symmetric encryption scheme that is indistinguishable under chosen-plaintext, that guarantees that $\{(x_i, z_i), \; i \in I_{samp}\}$ does not leak any additional information about z.

Regarding the sale phase, considering the pessimistic scenario, no additional information is shared by the sender a or by the *Judge* smart contract, during the verification of the PoM. □

Lemma 3.3.4. *FairDrop satisfies the sender fairness property.*

Grieving Security Protocol *grieving security* implies that in case of a dispute, the cheating party must always compensate the cheated party for any transaction fees paid during the dispute process (fee fairness).

Proof Sketch. Without grieving security, a malicious reporter a, could deliberately share a wrong key, forcing the receiver b to execute the PoM verification, and so to pay the corresponding transaction fees. To avoid this scenario the protocol requires the sender of the exchange to provide a *collateral*. This amount of money is sent to b if he is able to provide a correct PoM, otherwise it is refunded to the reporter a. Since *correctness* and *sender fairness* properties hold (Lemma 3.3.1 and 3.3.4), then the described mechanism leads to *grieving security*. □

Lemma 3.3.5. *FairDrop satisfies the fee-fairness (grieving security) property.*

CONFIDENTIALITY Protocol *confidentiality* implies that if at least the sender behaves honestly, then no information but the samples used for the description, i.e., $x_i \ s.t. \ k_i \in K_{samp}$, has to be visible to an external observer.

Proof Sketch. By sender fairness, Lemma 3.3.4, if the sender does not earn the pre-agreed amount of money, then the receiver cannot obtain any useful information about x, except for the samples related to its description, i.e., $x_i \ s.t. \ k_i \in K_{samp}$. We can consider confidentiality as a relaxation of sender fairness, where each external user is a receiver that does not complete or provides a payment to the sender. For *confidentiality*, we have only to prove that only the receiver that actually completes a correct payment can recover the master key.
This is guaranteed by the secrecy of *secret* value that is shared by the receiver during the sale phase. In particular, for an external user is computationally difficult to identify the *secret*, since:

- by the usage of a way-one hash function: it is not computationally feasible to identify the value of *secret*, just knowing $H(secret)$;
- by the asymmetric encryption scheme properties, we assume it is hard to derive the *secret*, only knowing $E_{as}(secret, pk)$.

□

Lemma 3.3.6. *FairDrop satisfies the confidentiality property.*

K-ANONYMITY By k-anonymity, we ensure that an external observer of the system has no significant advantage of guessing the reporter's identity than random guessing over the members of the ring.

Proof Sketch. Considering a single file exchange, the observable information for an external attacker are:

- the metadata regarding the file to be exchanged;
- the wallet associated with the reporter;
- the encrypted file signature done by the reporter.

We assume the reporter to take appropriate measures to secure his wallet, and we focus mainly on the other shared data. While the file metadata is not linked in any way with the reporter, the encrypted file signature is. By the assumptions on the ring signature scheme, it

provides unconditional protection of the signer identity [39]. That leads the attacker to don't have any non-negligible advantage in guessing the identity of the signer, w.r.t. random guessing. □

Lemma 3.3.7. *FairDrop satisfies the k-anonymity property, with k being the number of members in the considered ring.*

4

Performance Evaluation

In this section, we analyze the implementation and performance of the proposed protocol. We first consider the optimistic case, where both participants behave correctly, and then focus on the pessimistic one, characterized by the misbehavior of the sender, the reporter *a*, and the rise of an objection by the receiver, the newspaper *b*. We discuss and motivate the obtained results, comparing them with the proposed fair exchange protocol, to identify in which situations our protocol proves to be more effective.

4.1 Implementation

The implementation of the protocol has covered mainly the fair exchange part since the computation of a credibility measure starting from the ring signature still needs some research work to do. The implemented system consists of three components:

- the *Judge* smart contract deployed on the blockchain;
- a client to allow the reporter *a* to interact with the system, i.e., the blockchain and the smart contract *Judge*, as shown in Fig. 4.2;
- a client to allow the newspaper *b* to easily proceed with the execution of the protocol by interacting with the smart contract, as shown in Fig. 4.1.

The client scripts are written in Python, using the Web3.py [61] library. They provide an easy way for the reporter and newspaper to interact with the system and so execute the protocol.

The usage of a very intuitive and easy user interface, as visible in Fig. 4.1 and 4.2, reduces the knowledge entry barriers required for a possible user to actively operate through our platform.

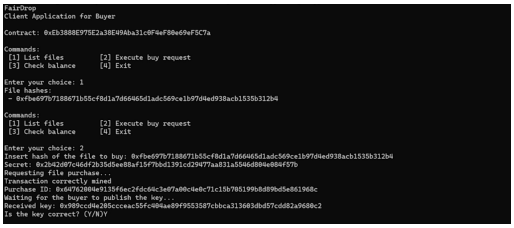

Figure 4.1: Receiver user interface, example of buying a file

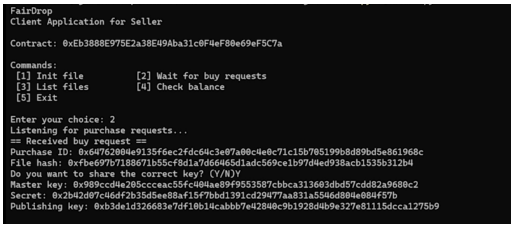

Figure 4.2: Sender user interface, example of buying a file

The smart contract is implemented in Solidity and runs on Ethereum [47]. For test purposes, we considered the Ropsten network, a PoW testnet of Ethereum. Later, following the changes in the consensus mechanism of Ethereum [62], we moved to the Goerli network, a PoS testnet of Ethereum. Inside the smart contract, we use the following Solidity functions as cryptographic primitives:

- the `keccak256` solidity interface for hashing: it receives in input an arbitrary long bytes value, and outputs hashes of 32 bytes;
- the `encodePacked` solidity method: it allows to provide an arbitrary number of parameters and outputs their concatenation;

Since Solidity does not provide built-in cryptographic primitives, except for the one previously mentioned, in the following paragraphs we treat in detail how we achieved on-chain operations, like encryption and randomness generation.

4.1.1 Symmetric encryption scheme

A symmetric encryption scheme is vital for the correct execution of the protocol since both the encoding of the file and the PoM mechanism are based on it. Due to the currently limited capabilities of the Solidity scripting language, we implemented the encryption as XOR between each plaintext x_i and sub-key k_i as:

$$z_i = E(k_i, x_i) = x_i \oplus k_i = x_i \oplus H(k||i). \tag{4.1}$$

This allows us to implement the decryption by easily computing:

$$x_i = D(k_i, z_i) = z_i \oplus k_i = z_i \oplus H(k||i). \tag{4.2}$$

This approach is currently used in most of the other blockchain-based fair exchange protocols [36, 37, 30] since it allows to perform and verify symmetric encryption and decryption on-chain. From our protocol perspective, since the `keccak256` hash function has a 32 bytes output space, then each sub-file x_i has to be of that size to allow efficient encryption and decryption. For the same reason, also the considered sub-keys and the secret used in the *sale phase*, are 32 bytes long.

4.1.2 Asymmetric encryption scheme

An asymmetric encryption scheme is necessary during the sale phase since it is at the base of the confidential exchange of the secret between the two parties. This requirement could be removed by assuming a confidential and secure communication channel between the two parties, as done in other paper [30, 37], but it does not apply to our application scenario.
Since the ciphertext is just shared through the smart contract, and no computation or verification is directly done on-chain, then there is no limitation related to the Solidity language to be considered. For our specific implementation, we used `NaCl` box, but since no strong constraint is identified, other asymmetric encryption schemes could be considered in other implementations.

4.1.3 Id generation

During the *sale phase*, to manage multiple exchanges, each of them is associated with a unique *id*. To generate them on-chain we use an *id generation strategy*. In our specific implementation we considered:

```
id ← bytes32(keccak256(abi.encodePacked(msg.sender, fileHash))),
```

that is the hash of the concatenation of the sender address, i.e., *msg.sender*, and the hash of the exchanged file, i.e. *fileHash*.
Using this id generation strategy, we can fix an *id* to a specific pair of file x and newspaper b. This allows a newspaper to execute the sale phase for a specific file only one time. We took this decision considering that:

- in the optimistic case, if the newspaper b successfully buys the file x, it is a viable option to assume he will not buy it again;

- in the pessimistic case, if the newspaper *b* is forced to provide a correct PoM, it is common sense to assume he will not try to buy the same file again from the same sender *a*.

Depending on the specific requirements of the system, the considered id generation strategy could change, and since the proposed protocol does not heavily rely on it, this is entirely achievable.

4.1.4 Randomness

During the initialization phase, before executing the sub-key sampling, the smart contract must publish a *randomness* value. Since this value heavily influences the sampling, and so the whole information exchange, the way it is generated is critical for the protocol.
In our implementation, we rely on information related to the current block, which is:

- `block.difficulty`: that is a measure of how difficult it is to mine the current block, i.e., to find a hash below a set target;
- `block.timestamp`: the time at which the block is generated.

More specifically we use the following formula to obtain the randomness value:

```
randomness ← uint(keccak256(abi.encodePacked(block.difficulty,
                                block.timestamp)))
```

In this way we can obtain a value for *randomness* that is not enforceable by the reporter *a*, ensuring that he is not able to actively decide which sub-keys to publicly share.
A possible security flow using this approach appears if we don't consider miners as trusted parties. In particular, the *block* information can be directly manipulated by the block miner, forcing a specific value for the randomness. For our implementation, we assumed miners as trusted parties so this scenario is not considered a valid attacking one.
For implementations that do not trust miners, Verifiable Random Function (VRF) [63] can be used. As the name suggests, they are functions that generate an entirely unpredictable (uniformly distributed) value and proof that demonstrates its validity. From the implementation point of view, Chainlink VRF [64], represents the standard and most used solution, in particular, it allows:

- *Unpredictability*: no one can predict the randomness since the used block data is unknown at the time the request is done;

- *Fairness*: the random number is drawn from a uniform distribution, meaning that all numbers in the considered range have the same probability of being extracted;

- *Randomness*: the randomness computation relies on block-hashes that are unknown ahead of time, and on a seed that is not publicly known;

- *Tamper-proof* : neither the oracle nor external entities can force the value to be drawn.

Summing up, the usage of VRF solutions, like Chainlink VRF, leads to higher randomness and then security for the whole protocol, at the cost of more complex implementation and higher usage fees for the final users.

4.2 Performance

In this section, we provide the measurement and performances related to FairDrop. We first provide a theoretical analysis, focused on the complexity of the algorithm used and on the security measures achieved, and then a more practical examination, reporting on-chain and off-chain execution costs.

4.2.1 Complexity analysis

Fixing a reporter a and a file x, during the initialization phase the reporter has to call the `publishFile` and `publishDescription` methods, that are necessary to share correspondingly the hash and the description of the considered file. Both the methods execute only data transfers and memory assignments, and for this reason, their complexity is constant, i.e., $\mathcal{O}(1)$ both in time and memory. The initialization phase also requires some off-chain computation by the reporter a, who has to use the randomness obtained by calling the `publishFile` method to sample from all the sub-keys. The proposed sampling mechanism has a $\mathcal{O}(n)$ space complexity, and $\mathcal{O}(s)$ time complexity, with s being the number of sampled elements, and n the size of the initial file. After that, a uses the sampled sub-keys to generate the description of the file, this process has $\mathcal{O}(s)$ space and time complexity. All the previous complexity data is summarized in Table 4.1, where *SKE* refers to the complexity of symmetric key encryption.

Table 4.1: Initialization phase complexity analysis overview

	On-chain		Off-chain		
	`publishFile`	`publishDescription`	Encoding	Sampling	Description
Time	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n \cdot SKE)$	$\mathcal{O}(s)$	$\mathcal{O}(s)$
Memory	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(x)$	$\mathcal{O}(n)$	$\mathcal{O}(s)$

If we fix a newspaper b, then we can analyze the computational complexity of the sale phase. More in particular, during this phase, b has first to use the shared sub-keys by the reporter a to decrypt the corresponding sub-files, which can be done in $\mathcal{O}(s \cdot SKE)$ time. After that, if he decides to proceed with the protocol, he has to call the `buy` method, which does not execute any particular computation and has a $\mathcal{O}(1)$ time and space complexity. If the reporter a decides to continue with the protocol, he shares the key calling the `publishKey` method of the smart contract, which also has constant space and time complexity. Considering the optimistic case, the protocol ends at this moment. A summary of the previously mentioned complexity measures is available in Table 4.2.

Table 4.2: Sale phase, optimistic case, complexity analysis overview

	On-chain				Off-chain
	`buy`	`publishKey`	`withdraw`	`refundToBuyer`	Decoding
Time	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(s \cdot SKE)$
Memory	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(s)$

If we consider the pessimistic case, then the newspaper b has to: first, compute off-chain a PoM, and then share it with the smart contract invoking the `raiseObjection` method. The time and space complexity of this procedure is shown in Table 4.3.

Table 4.3: Sale phase, pessimistic case, complexity analysis overview

	On-chain	Off-chain
	`raiseObjection`	PoM Generation
Time	$\mathcal{O}(\log s)$	$\mathcal{O}(s)$
Memory	$\mathcal{O}(1)$	$\mathcal{O}(s)$

4.2.2 Execution costs

As for the implementation, also the performances are focused on the fair exchange part of the protocol. We first describe the metrics we use for the performance evaluations and then examine in detail each phase of the protocol, comparing it with the closely related works. During this discussion, we make use of the results to make explicit if the design choices that we took led to the expected results or not.

Metrics The cost of running a protocol in the blockchain is expressed as the amount of *gas* [65] used. The *gas* represents a unit of measure for the fees to pay for the execution of operations in the blockchain. Some examples of operations executed by our protocol are byte manipulations, hash of strings, and event emission. Other elements that lead to gas usage are memory storage inside the contract and the size and the number of each method parameters. As mentioned before, the benchmarks are executed in the Ethereum blockchain, specifically in the Ethereum Ropsten and Goerli Testnet. Starting with the deployment of the contract, it requires `1936780` units of gas to be completed.

Initialization Phase As explained before, this phase is composed of two rounds, that correspond to two smart contract method calls: `publishFile` and `publishDescription`, both executed by the sender a. The on-chain execution cost of this phase is reported in Table 4.4.

Table 4.4: Initialization phase execution gas cost

Method	`publishFile`	`publishDescription`	Total
Gas	93728	48890	142618

Being part of the initialization phase, the sender a has to cope with these costs only one time for each file, independent of the number of successive sales.

Sale Phase Since the smart contract is designed to manage multiple buyers and senders, each call of a method during the sale phase is characterized by a certain overhead, related to all the checks that are necessary to provide this feature. In particular, the checks introduce an execution cost equal to:

$$C_H + 3 \cdot C_{EQ},$$

where C_H represents the cost for the computation of a hash, while C_{EQ} is the cost for the comparison of two values.
The on-chain execution costs of this phase are reported in Table 4.5.

Table 4.5: Sale phase method execution cost

Method	`buy`	`publishKey`	`refundToBuyer`	`withdraw`
Gas	107645	80015	59660	44601

The method `buy` request the caller to provide the ciphertext of the secret he has to share. The size of the ciphertext can change depending on the asymmetric encryption scheme used. In any case, the plaintext, i.e., the secret to share, is always 32 bytes long. So assuming to fix the encryption scheme, also the ciphertext size would be constant.
The execution cost of the `raiseObjection` method depends on the size of the provided PoM, and so on the logarithm of s. In the table 4.6 we show how the price the newspaper b has to pay to execute this method changes at the variation of the number of sampled sub-keys s.

Table 4.6: RaiseObjection method cost execution

Sub-keys	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
Gas	64605	66125	67657	69140	70684	72204	73712	75244	76776

To have a better idea of the costs associated with the sale phase, we consider fixing the number of sampled sub-keys s to 2^{10}, and show in Table 4.7 the overall costs of this phase, distinguishing between the sender and the receiver.

Table 4.7: Sale phase overall execution costs

	Optimistic case		Pessimistic case	
	Sender	Receiver	Sender	Receiver
Gas	124616	107645	80015	178329

As visible in Table 4.7, the execution in the pessimistic case for the sender is cheaper than the one in the optimistic case. This is because in the pessimistic case the sender doesn't have to call the `withdraw` method. At the same time, even if the execution is cheaper, the sender will result in a greater economic loss in the pessimistic case, since he would not earn anything

with the exchange, and he will lose the *collateral* sent to the smart contract, due to *probabilistic receiver fairness* property. As proven in Sec. 3.3.3, the probability by a malicious reporter to break this property is upper bounded by $(1-p)^s$, with p being the probability of a sub-key k_i to be wrong, and s the number of sub-keys to sample for the description. In Fig. 4.3, we show how this upper bound changes for different values of s and p. The probability of a successful attack decreases very fast: with a low number of sub-keys sampled, e.g., 2^5, and a probability p quite low, e.g., 10%, the attack will succeed only the 0.3% of times.

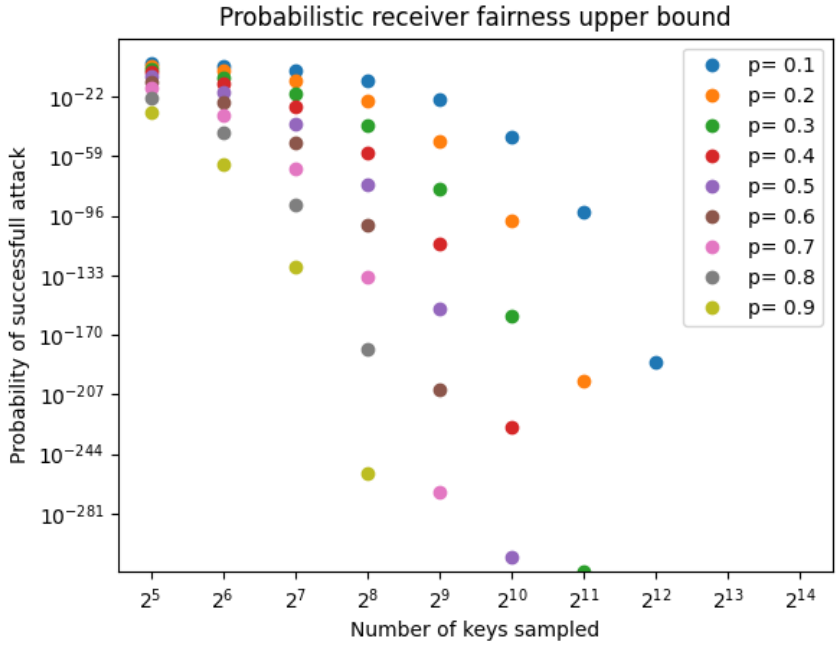

Figure 4.3: Probability of success of an attack against receiver fairness

4.2.3 Performance Comparison with Closely Related Work

We designed this protocol to provide features that state-of-the-art fair exchange protocols were not considering, trying to improve their performances when multiple exchanges have to be executed. As the core of our protocol, we used the idea proposed by FairDex [30], i.e., the creation of a Merkle tree over the sub-keys, and not over the sub-files, as done in other works [36, 37]. As visible in Fig. 4.4 the price that the sender of the file has to pay is considerably lower than the one required by FairDex. FairDrop achieves this by moving the duty of deployment of the smart contract from the sender to the initial provider of the system, e.g., a non-profit

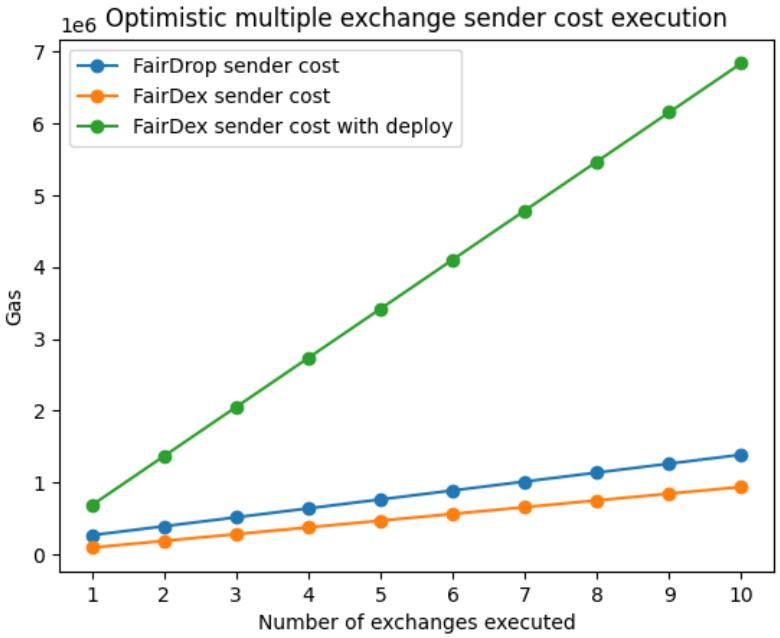

Figure 4.4: Optimistic case: price cost for sender considering multiple exchanges

organization or a journalist company. The deployer of the system doesn't gain any specific right on the smart contract, but it allows any other reporter and newspaper to use it, removing the initial deployment cost from them. Once deployed, the system can be used an unlimited number of times, while the other currently proposed solutions [36, 30, 37] require the sender to deploy a new smart contract for every exchange.

Regarding FairSwap [36] and OptiSwap [37], both FairDex [30] and our protocol result in a lower cost in the pessimistic case, due to the usage of the Merkle tree based on sub-keys. Considering the optimistic case, all the protocols are characterized by a constant time complexity, which difference is only related to the specific operations executed on the blockchain.
Comparing directly FairDex and FairDrop, and considering a single data exchange, in the optimistic case our protocol results in an execution that is `17.75%` more expensive for the newspaper and `32.58%` for the reporter. While in the pessimistic case, `26.98%` more expensive for the newspaper and between `35%` and `40%` for the reporter, depending on the size of the description. This overhead is related to the computation FairDrop has to do to manage the features it introduces, but as visible in Fig. 4.4 for the reporter, our protocol is cheaper if when don't focus just on the single exchange, but if we consider the overall costs.

5

Conclusion

The exchange of information in a confidential and anonymous way represents in its general version the main topic of the current historical period. If we focus on the media world, then we can notice how many of the most relevant and significant articles have been written and published thanks to reporters and sources who, in exchange for anonymity, decided to share some information. Many times sources are limited in what they can report and witness by the policies and decisions taken by the company they work for or the country they live in. To help reporters and sources and to allow an easier and more free flow of information we introduced FairDrop.

FairDrop is a system designed considering the state-of-the-art of fair exchange protocols, and keeping in mind the importance of the protection and anonymity of all the involved parties. For FairDrop we focused mainly on the media world, basing most of the design choices on how real communications between reporters and news organizations actually work. To achieve this goal, during the research work of this thesis we collaborated also with the Oskam Foundation a non-profit organization with the purpose to protect the digital rights of journalists.

We presented a system to allow the confidential exchange of information between a reporter and a newspaper, maintaining the anonymity of the former one. We prioritized the definition of a solution that could actually follow the real necessities of both the considered parties. We finally introduced the formalization and the basic ideas for the design of a more complex system to obtain a numerical measure of the credibility of the information shared.

5.1 Discussion

In this section, we highlight what we consider the most important aspects and features introduced by our protocol, and the results we were able to achieve through the proof of concept that we developed.

Multi-buyer and seller fair exchange FairDrop is able to maintain all the properties provided by already existing fair exchange protocols based on blockchain [36, 37, 30], while adding, among the others, the confidentiality of the shared key, multi-buyer, and multi-seller possibilities. These features that we introduced allow the system to be used in real scenarios, like the media workers one.

Runtime performance An overhead in the runtime performance is added w.r.t. other protocols by the features FairDrop introduces. Fixing a specific piece of information shared by a reporter, and considering a multiple-buyer scenario, for which FairDrop has explicitly been designed, we obtain better performances than other existing protocols, even introducing property, like confidentiality, that they don't provide. Also, FairDrop has to be deployed only one time, and then any user can actively utilize it without the necessity to deploy it on his own. This allows the system to become cheaper than the other proposed solutions, which require a new smart contract deployment for each exchange.

No direct communication Inside the protocol, no assumption regarding the presence of a direct and secure communication channel between the two parties that participate in the exchange is done. The usage of such a channel would probably reduce the cost and complexity, in the number of rounds, of the protocol, but would also be a strong assumption to do. We then preferred to consider a more realistic scenario, eliminating any direct communication between the involved parties.

Shared proof-of-credibility The published ring signature can be verified by anyone, independently of his participation in the system. The signature can be then attached to a final article by the newspaper, to provide a verifiable proof of the credibility of the sources.

5.2 Limitations

During the design and subsequent implementation of the proposed solution, we did some assumptions and simplifications, which we discuss in this section.

Identification by public keys During the whole protocol, the identification of reporters is based on their public key, which availability in a normal scenario is by no means taken for granted. In the future, we could assume this possibility to be more concrete in limited scenarios, like the media and journalism world. The protocol seems for this reason applicable to reporters, but a generalization to any source, such as company insiders, would require an availability of public keys that, at least at the moment, is not realistic to consider.

Credibility We defined from a general point of view the next steps in the definition of a credibility measure starting from the usage of a ring signature, but an actual implementation of it has yet to be done.

5.3 Future Work

Improvement for the specific use case Focusing on the reporter and newspaper scenario possible specific improvements can be done. For example, by paying a higher amount of money, the newspaper could request exclusive rights on the piece of information, and limit or block the number of exchanges of it.

Credibility implementation The definition of a minimum set of information that allows explaining the level of expertise of a reporter without revealing its identity has yet to be defined. Also, the design of a function that uses that data and a ring signature to define a measure of the credibility of the ring and of the shared information remains a future research work. Considering the ring signature, and the problem of the availability of public keys, the composition and the size of the ring represent a necessary research direction to consider since it directly affects the credibility of the information.

Perfect randomness As treated in Sec. 4.1 the strategy we used for randomness generation depends on parameters that could be directly influenced by a malicious blockchain miner. To avoid this problem, the usage of a blockchain-based verifiable randomness function has been

proposed. We leave as future work the actual implementation and test of it, and the discussion about the trade-off between the security achieved in this way and the increase in user cost for using the system.

Existing systems integration In our work we just introduced the existing systems already used by newspapers to accept information and communicate with anonymous sources. Further research could determine if the integration of our protocol with the existing solutions can be feasible. This would allow the introduction of all the features that FairDrop provides to strongly adopted systems.

Secret sharing During the sale phase the secret sharing mechanism requires the usage of an asymmetric encryption scheme and the publication through the blockchain of the corresponding ciphertext. Further research could determine if this information can be shared without the usage of the blockchain, leading to less gas usage by the involved parties.

A

Merkle Trees Algorithms

As already stated in Chapter 2, a Merkle Tree of elements $x_1, ..., x_n$ in $\{0,1\}^*$ is a binary tree $M = MTree(x_1, ..., x_n)$, with MTree defined as Alg. 1. For simplicity, we consider a complete binary tree, i.e., $n = 2^m$, for some m.
We can define a Merkle Tree in a recursive way: each non-leaf node V_j is defined as the hash of its right and left children V_j^l and V_j^r, i.e., $V_j = H(V_j^l, V_j^r)$.
We refer to V_j^l and V_j^r as *siblings*, while V_J as their *parent*.

Algorithm 1 Merkle tree hash *Mtree*

Input: $(x_1, ..., x_n)$
  set V ▷ V will be the root node
  if $N = 1$ **then** ▷ Base Case: the input is a single value
    $label(V) \leftarrow x_1$
  else
    $v_0^l \leftarrow MTree(x_1, ..., x_{\lceil n/2 \rceil})$ ▷ Call the function on the left subtree
    $v_0^r \leftarrow MTree(x_{\lceil n/2 \rceil + 1}, ..., x_n)$ ▷ Call the function on the right subtree
    $label(V) \leftarrow H(root(v_0^l) || root(v_0^r))$ ▷ Compute the label for the current root
  end if
Output: Merkle tree M with root V

To prove that a leaf node is part of the tree, we use the output of the Alg. 2. The algorithm returns a vector of size $log_2(n)$, containing all the siblings of elements on a path from the considered node to the root of the tree.

Algorithm 2 Merkle tree proof *Mproof*

Input: Merkle tree M, index i
  $V = M[i]$ ▷ let V be the i-th leaf node of M
  for $j \in [log_2(n)]$ **do**
    $l_j \leftarrow$ label(sibling of v)
    $v \leftarrow$ parent of v
  end for
Output: Merkle Proof $p = (l_1, ..., l_d)$

To verify the proof obtained in Alg. 2, we can use the *Mverify* algorithm. In particular, given a Merkle Tree M, its root *root(M)*, a leaf node x_i and a proof ρ obtained by using *Mproof(M, i)*, we define the algorithm *Mverify* that returns *true* if x_i belongs to M, and *false* otherwise.

Algorithm 3 Merkle tree proof verification *Mverify*

Input: $i \in [n], x \in \{0,1\}*, \rho = (l_1, ..., l_d), h \in \{0,1\}^{\mu}$
  for $l_j \in \rho$ **do** ▷ We walk across the provided path.
    $x \leftarrow H(x||l_j)$ ▷ Compute x as the parent, up to the root of the tree.
  end for
Output: $x = h$ ▷ The proof is correct, if at the end x corresponds with the root.

The time and space complexity of the previously described algorithms is reported in Table A.1. The logarithmic complexity of *Mproof* and *Mverify* is related to the fact that they execute an operation for each level of the tree. Since we consider the binary tree to be balanced, then the complexity is $\mathcal{O}(\log n)$, while in a more generic case we could consider as complexity $\mathcal{O}(h)$, with h being the height of the tree. The *Mtree* algorithm is $\mathcal{O}(n)$ instead, since it needs to compute the value for each node of the tree.

Table A.1: Space and time complexity of Markle tree algorithms

	Mtree	*Mproof*	*Mverify*
Time	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Memory	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$

Considering the space complexity, for *Mtree*, it is necessary to create and store the whole tree, that is $\mathcal{O}(n)$. Regarding *Mproof*, it creates a vector of $\log n$ elements to prove that the selected one belongs to the tree. Finally, *Mverify*, takes in input a proof and a tree, and using constant memory, i.e., $\mathcal{O}(1)$ space, verifies the validity of the proof.

B

Smart Contract

The code of the smart contract that we designed is provided in this appendix.

```
contract ConfidentialFairExchange {

    uint constant public MAX_INTERVAL = 2 minutes; // 2 mins  timemax inteval
        between operations
    uint constant public COLLATERAL = 100 wei;

    /******** INIT PHASE *******/
    event FileRandomness(bytes32 indexed fileHash, uint randomness);
    event FilePublished(bytes32 indexed fileHash, bytes32 description);

    // struct containing info about the file to sell
    struct FileInfo {
        address seller;
        uint depth;  // description depth
        uint price;
        bytes32 description;
    }

    // mapping associating to each hash the corrisponding FileInfo struct
    mapping(bytes32 => FileInfo) public fileInfos;

    function publishFile(bytes32 _fileHash, uint _depth, uint _price) public {
```

```
22          require(fileInfos[_fileHash].seller == address(0), "The file has already
                been published");
23
24          fileInfos[_fileHash] = FileInfo(msg.sender, _depth, _price, 0);
25          emit FileRandomness(_fileHash, uint(keccak256(abi.encodePacked(block.
                difficulty, block.timestamp))));
26      }
27
28      function publishDescription(bytes32 _fileHash, bytes32 _description) public {
29          require(fileInfos[_fileHash].seller == msg.sender, "Only the seller can set
                the description");
30          require(fileInfos[_fileHash].description == 0, "The description has already
                been set");
31
32          fileInfos[_fileHash].description = _description;
33          emit FilePublished(_fileHash, _description);
34      }
35
36      /**** BUYING PHASE *****/
37      event PurchaseRequested(bytes32 indexed fileHash, bytes32 indexed purchaseID,
            bytes32 secretHash, bytes encryptedSecret);
38      event EncryptedKeyPublished(bytes32 indexed purchaseID, bytes32 encryptedKey);
39
40      enum State { Requested, EncryptedKeyShared, Completed, Invalid, Timeout}
41      struct Purchase {
42          address buyer;
43          uint lastOperationTime;
44          State state;
45          bytes32 encryptedKey;
46          bytes32 secretHash;
47      }
48
49      mapping(bytes32 => Purchase) purchases;
50      function buy(bytes32 _fileHash, bytes32 _secretHash, bytes calldata
            _encryptedSecret)
51          public payable {
52
53          require(fileInfos[_fileHash].seller != address(0), "No file with requested
                hash is present inside the system");
54          require(fileInfos[_fileHash].description != 0, "The considered file has not
                description set yet");
```

```
55          require(msg.value == fileInfos[_fileHash].price, "The sent amount of money
                is lower than the file price");
56
57          bytes32 purchaseID = bytes32(keccak256(abi.encodePacked(msg.sender,
                _fileHash)));
58          require(purchases[purchaseID].state == State.Requested, "Purchase already
                started");
59          purchases[purchaseID] = Purchase(msg.sender, block.timestamp, State.
                Requested, 0, _secretHash);
60
61          emit PurchaseRequested(_fileHash, purchaseID, _secretHash, _encryptedSecret)
62      }
63
64      /* buyer calls the buy function -> seller checks H(D(_encryptedSecret)) ==
            _secretHash:
65          - if false: wait MAX_INTERVAL and call withdraw
66          - if true: call publishKey (with collateral)
67      */
68      function publishKey(bytes32 _encryptedKey, bytes32 _fileHash, bytes32
            _purchaseID)
69          Only(_fileHash, _purchaseID, State.Requested, fileInfos[_fileHash].seller)
70          InTime(_purchaseID)
71          public payable {
72
73          purchases[_purchaseID].encryptedKey = _encryptedKey;
74          purchases[_purchaseID].state = State.EncryptedKeyShared;
75          purchases[_purchaseID].lastOperationTime = block.timestamp;
76          emit EncryptedKeyPublished(_purchaseID, _encryptedKey);
77      }
78
79      // Complain about the goods by proving that:
80      //  1. a committed subkey is different from the subkey derived from the master
            key; and
81      // 2. and this committed subkey is part of the description.
82      struct POM {
83          uint _committed_ri;
84          bytes32 _committedSubKey;
85          bytes32[] _merkleTreePath;
86      }
87      function raiseObjection(bytes32 _fileHash, bytes32 _purchaseID, bytes32 _secret,
88          POM memory pom)
```

```
89          Only(_fileHash, _purchaseID, State.EncryptedKeyShared, purchases[_purchaseID
                ].buyer)
90          InTime(_purchaseID)
91          public
92          payable
93      {
94          require(keccak256(abi.encodePacked(_secret)) == purchases[_purchaseID].
                secretHash, "Provided secret is wrong");
95          bytes32 key = purchases[_purchaseID].encryptedKey ^ _secret;
96
97          bytes32 computedSubkey = keccak256(abi.encodePacked(key, pom._committed_ri))
                ;
98          // Check if the subkey supplied by buyer is different from the subkey
                derived from masterKey.
99          if (computedSubkey != pom._committedSubKey) {
100             bytes32 committedNode = keccak256(abi.encodePacked(pom._committedSubKey,
                     pom._committed_ri));
101
102             // When the loop exits, committedNode will hold the root of Merkle Tree.
103             for (uint i = 0; i < fileInfos[_fileHash].depth; i++)
104                 committedNode = keccak256(abi.encodePacked(committedNode, pom.
                        _merkleTreePath[i]));
105
106             // Check if the root equals description.
107             if (committedNode == fileInfos[_fileHash].description) {
108                 // If so, the buyer is right and gets the deposit + collateral back.
109                 purchases[_purchaseID].state = State.Invalid;
110                 payable(msg.sender).transfer(COLLATERAL + fileInfos[_fileHash].price
                        );
111             }
112         }
113     }
114
115     // Refund to buyer if seller does not publish master key in time.
116     function refundToBuyer(bytes32 _fileHash, bytes32 _purchaseID)
117         Only(_fileHash, _purchaseID, State.Requested, purchases[_purchaseID].buyer)
118         public payable {
119
120         require(purchases[_purchaseID].lastOperationTime + MAX_INTERVAL < block.
                timestamp, "The seller has still time to share the encrypted key");
121         purchases[_purchaseID].state = State.Timeout;
```

```
122         payable(msg.sender).transfer(fileInfos[_fileHash].price);
123     }
124 
125     function withdraw(bytes32 _fileHash, bytes32 _purchaseID)
126         Only(_fileHash, _purchaseID, State.EncryptedKeyShared, fileInfos[_fileHash].
                seller)
127         public payable
128     {
129         require(purchases[_purchaseID].lastOperationTime + MAX_INTERVAL < block.
                timestamp, "The buyer has still time to share a POM");
130         purchases[_purchaseID].state = State.Completed;
131         payable(msg.sender).transfer(fileInfos[_fileHash].price + COLLATERAL);
132     }
133 
134     modifier InTime(bytes32 _purchaseID) {
135         require(block.timestamp <= purchases[_purchaseID].lastOperationTime +
                MAX_INTERVAL, "Max timeout reached");
136         _;
137     }
138 
139     modifier Only(bytes32 _fileHash, bytes32 _purchaseID, State _state, address
            caller) {
140         require(bytes32(keccak256(abi.encodePacked(purchases[_purchaseID].buyer,
                _fileHash))) == _purchaseID, "Wrong purchase ID");
141         require(purchases[_purchaseID].state == _state, "Purchase in the wrong state
                ");
142         require(msg.sender == caller, "Wrong function caller");
143         _;
144     }
145 }
```

References

[1] "World press freedom index - 2022," 2022. [Online]. Available: https://rsf.org/en/rsfs-2022-world-press-freedom-index-new-era-polarisation

[2] (2020) Murders of journalists more than double worldwide. [Online]. Available: https://cpj.org/reports/2020/12/murders-journalists-more-than-doubled-killed/

[3] (2022) Attacks on the press: The deadliest countries in 2021. [Online]. Available: https://cpj.org/reports/2022/01/attacks-on-the-press-the-deadliest-countries-in-2021/

[4] (2020) Record number of journalists jailed worldwide. [Online]. Available: https://cpj.org/reports/2020/12/record-number-journalists-jailed-imprisoned/

[5] Spj ethics committee position papers, anonymous sources. [Online]. Available: https://www.spj.org/ethics-papers-anonymity.asp

[6] Anonymous sources. [Online]. Available: https://www.ap.org/about/news-values-and-principles/telling-the-story/anonymous-sources

[7] Understanding the times, how the times uses anonymous sources. [Online]. Available: https://www.nytimes.com/2018/06/14/reader-center/how-the-times-uses-anonymous-sources.html

[8] D. Ohlmeyer. (2010) Root of all evil? [Online]. Available: http://sports.espn.go.com/espn/columns/story?columnist=ohlmeyer_don&id=5220492

[9] (1980) Jimmy's world, janet cooke. [Online]. Available: https://www.washingtonpost.com/archive/politics/1980/09/28/jimmys-world/605f237a-7330-4a69-8433-b6da4c519120/

[10] J. Barry. (2005) Gitmo: Southcom showdown. [Online]. Available: https://www.newsweek.com/gitmo-southcom-showdown-119001

[11] (2005) Newsweek retracts quran story. [Online]. Available: https://edition.cnn.com/2005/WORLD/asiapcf/05/16/newsweek.quran/

[12] S. Dasgupta and A. Kesharwani, "Whistleblowing: A survey of literature," pp. 57–70, 2011.

[13] P. Di Salvo, "Securing Whistleblowing in the Digital Age: SecureDrop and the Changing Journalistic Practices for Source Protection," *Digital Journalism*, vol. 9, no. 4, pp. 443–460, Apr. 2021. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/21670811.2021.1889384

[14] Securedrop: secure communication between journalists and sources. [Online]. Available: https://securedrop.org/overview/

[15] J. M. Shepard, *Anonymous Sources and Source Confidentiality*. John Wiley & Sons, Ltd, 2019, pp. 1–5. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118841570.iejs0258

[16] L. Wilkins, "Anonymous sources," *Journalism Ethics*, p. 117–122, 1997.

[17] A. Quinn, "271Respecting Sources' Confidentiality: Critical but Not Absolute," in *Journalism Ethics: A Philosophical Approach*. Oxford University Press, 02 2010. [Online]. Available: https://doi.org/10.1093/acprof:oso/9780195370805.003.0018

[18] D. E. Boeyink, "Anonymous Sources in News Stories: Justifying Exceptions and Limiting Abuses," *Journal of Mass Media Ethics*, vol. 5, no. 4, pp. 233–246, Dec. 1990. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1207/s15327728jmme0504_2

[19] F. Brown, "Anonymity hurts reporters and politicians." p. 38–39, 2003.

[20] B. M. Gassaway, "Are Secret Sources in the News Media Really Necessary?" *Newspaper Research Journal*, vol. 9, no. 3, pp. 69–77, Mar. 1988. [Online]. Available: http://journals.sagepub.com/doi/10.1177/073953298800900307

[21] T. Son, "Leaks: How Do Codes of Ethics Address Them?" *Journal of Mass Media Ethics*, vol. 17, no. 2, pp. 155–173, Jun. 2002. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1207/S15327728JMME1702_05

[22] J. Strupp, "Losing confidence," vol. 138, pp. 32–39, 07 2005.

[23] (2005) I'm the guy they called deep throat. [Online]. Available: https://www.vanityfair.com/news/politics/2005/07/deepthroat200507

[24] A. Shepard. (1994) Anonymous sources. american journalism review. [Online]. Available: https://ajrarchive.org/Article.asp?id=1596

[25] (2005) Karzai condemns anti-us protests. [Online]. Available: http://news.bbc.co.uk/2/hi/south_asia/4547413.stm

[26] (2003) Times reporter who resigned leaves long trail of deception. [Online]. Available: https://www.nytimes.com/2003/05/11/us/correcting-the-record-times-reporter-who-resigned-leaves-long-trail-of-deception.html

[27] (2004) Ex-usa today reporter faked major stories. [Online]. Available: https://usatoday30.usatoday.com/news/2004-03-18-2004-03-18_kelleymain_x.htm

[28] R. F. Smith, "Impact of Unnamed Sources on Credibility Not Certain," *Newspaper Research Journal*, vol. 28, no. 3, pp. 8–19, Jun. 2007. [Online]. Available: http://journals.sagepub.com/doi/10.1177/073953290702800302

[29] M. M. Sternadori and E. Thorson, "Anonymous Sources Harm Credibility of All Stories," *Newspaper Research Journal*, vol. 30, no. 4, pp. 54–66, Sep. 2009. [Online]. Available: http://journals.sagepub.com/doi/10.1177/073953290903000405

[30] O. Ersoy, Z. A. Genc, Z. Erkin, and M. Conti, "Practical Exchange for Unique Digital Goods," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. United Kingdom: IEEE, Aug. 2021, pp. 49–58. [Online]. Available: https://ieeexplore.ieee.org/document/9566170/

[31] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on Bitcoin transactions," *Future Generation Computer Systems*, vol. 107, pp. 832–840, Jun. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X17318344

[32] J. Gao, T. Wu, and X. Li, "Secure, fair and instant data trading scheme based on bitcoin," *Journal of Information Security and Applications*, vol. 53, p. 102511, Aug. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2214212619309688

[33] Y. Chen, J. Guo, C. Li, and W. Ren, "FaDe: A Blockchain-Based Fair Data Exchange Scheme for Big Data Sharing," *Future Internet*, vol. 11, no. 11, p. 225, Oct. 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/11/225

[34] H. Yu, J. Gao, T. Wu, and X. Li, "A novel fair and verifiable data trading scheme," in *Frontiers in Cyber Security*, B. Shen, B. Wang, J. Han, and Y. Yu, Eds. Singapore: Springer Singapore, 2019, pp. 308–326.

[35] Y. Zhao, Y. Yu, Y. Li, G. Han, and X. Du, "Machine learning based privacy-preserving fair data trading in big data market," *Information Sciences*, vol. 478, pp. 449–460, Apr. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025518309174

[36] S. Dziembowski, L. Eckey, and S. Faust, "FairSwap: How To Fairly Exchange Digital Goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Toronto Canada: ACM, Oct. 2018, pp. 967–984. [Online]. Available: https://dl.acm.org/doi/10.1145/3243734.3243857

[37] L. Eckey, S. Faust, and B. Schlosser, "OptiSwap: Fast Optimistic Fair Exchange," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. Taipei Taiwan: ACM, Oct. 2020, pp. 543–557. [Online]. Available: https://dl.acm.org/doi/10.1145/3320269.3384749

[38] D. R. Stinson and M. B. Paterson, *Cryptography: theory and practice*, fourth edition ed. Boca Raton: CRC Press, Taylor & Francis Group, 2019.

[39] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[40] D. Chaum and E. v. Heyst, "Group signatures," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 257–265.

[41] A. Narayanan, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton: Princeton University Press, 2016.

[42] C. S. Wright, "Bitcoin: A Peer-to-Peer Electronic Cash System," *SSRN Electronic Journal*, 2008. [Online]. Available: https://www.ssrn.com/abstract=3440802

[43] U. W. Chohan, "The Double Spending Problem and Cryptocurrencies," *SSRN Electronic Journal*, 2017. [Online]. Available: https://www.ssrn.com/abstract=3090174

[44] A. M. Antonopoulos, *Mastering Bitcoin: programming the open blockchain*, second edition ed. Sebastopol, CA: O'Reilly, 2017, oCLC: ocn953432201.

[45] C. Nguyen, H. Dinh Thai, D. Nguyen, D. Niyato, H. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. PP, pp. 1–1, 06 2019.

[46] M. Platt, J. Sedlmeir, D. Platt, J. Xu, P. Tasca, N. Vadgama, and J. I. Ibanez, "The Energy Footprint of Blockchain Consensus Mechanisms Beyond Proof-of-Work," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. Hainan, China: IEEE, Dec. 2021, pp. 1135–1144. [Online]. Available: https://ieeexplore.ieee.org/document/9741872/

[47] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[48] W. Li, M. He, and S. Haiquan, "An overview of blockchain technology: Applications, challenges and future trends," pp. 31–39, 2021.

[49] H. Pagnia and F. C. G. Darmstadt, "On the impossibility of fair exchange without a trusted third party," 1999.

[50] N. Asokan, M. Schunter, and M. Waidner, "Optimistic protocols for fair exchange," in *Proceedings of the 4th ACM conference on Computer and communications security - CCS '97*. Zurich, Switzerland: ACM Press, 1997, pp. 7–17. [Online]. Available: http://portal.acm.org/citation.cfm?doid=266420.266426

[51] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*. Oakland, CA, USA: IEEE Comput. Soc, 1998, pp. 86–99. [Online]. Available: http://ieeexplore.ieee.org/document/674826/

[52] G. Avoine, F. Gärtner, R. Guerraoui, and M. Vukolić, "Gracefully degrading fair exchange with security modules," in *Dependable Computing - EDCC 5*, M. Dal Cin,

M. Kaâniche, and A. Pataricza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 55–71.

[53] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest, "A fair protocol for signing contracts," *IEEE Transactions on Information Theory*, vol. 36, no. 1, pp. 40–46, Jan. 1990. [Online]. Available: http://ieeexplore.ieee.org/document/50372/

[54] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, vol. 28, no. 6, pp. 637–647, Jun. 1985. [Online]. Available: https://dl.acm.org/doi/10.1145/3812.3818

[55] M. O. Rabin, "Transaction protection by beacons," *Journal of Computer and System Sciences*, vol. 27, no. 2, pp. 256–267, Oct. 1983. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0022000083900429

[56] C. Badertscher, U. M. amd Daniel Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Advances in Cryptology – CRYPTO 2017*, ser. LNCS, vol. 10401 (Proceedings Part I). Springer, 8 2017, pp. 324–356.

[57] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 477–498.

[58] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 705–734.

[59] A. Küpçü and A. Lysyanskaya, "Usable optimistic fair exchange," *Computer Networks*, vol. 56, no. 1, pp. 50–63, Jan. 2012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S138912861100301X

[60] R. A. Fisher and F. Yates, *Statistical tables for biological, agricultural, and medical research*, 6th ed. New York: Hafner Pub. Co, 1963, oCLC: 550706.

[61] (2021) web3.py. [Online]. Available: https://github.com/ethereum/web3.py

[62] E. Kapengut and B. Mizrach, "An event study of the ethereum transition to proof-of-stake," 2022. [Online]. Available: https://arxiv.org/abs/2210.13655

[63] Chainlink. (2022) Verifiable random function (vrf). [Online]. Available: https://blog.chain.link/verifiable-random-function-vrf/

[64] L. Breidenbach, C. Chacin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, S. Nazarov, A. Topliceanu, F. Tramèr, and F. Zhang. (2021) Chainlink 2.0. the next steps in the evolution of decentralized oracle networks. [Online]. Available: https://research.chain.link/whitepaper-v2.pdf

[65] Ethereum gas and fees. [Online]. Available: https://ethereum.org/en/developers/docs/gas/