Blockchain-Based P2P Content Delivery with Monetary Incentivization and Fairness Guarantee

Songlin He, *Member, IEEE,* Yuan Lu, Qiang Tang, *Member, IEEE,* Guiling Wang, *Fellow, IEEE,* and Chase Qishi Wu, *Senior Member, IEEE*

Abstract—Peer-to-peer (P2P) content delivery is up-and-coming to provide benefits comprising cost-saving and scalable peak-demand handling compared with centralized content delivery networks (CDNs), and also complementary to the popular decentralized storage networks such as Filecoin. However, reliable P2P delivery demands proper enforcement of delivery fairness, i.e., the deliverers should be rewarded in line with their in-time delivery. Unfortunately, most existing studies on delivery fairness are on the basis of non-cooperative game-theoretic assumptions that are arguably unrealistic in the ad-hoc P2P setting.

We propose an expressive yet still minimalist security requirement for desired fair P2P content delivery, and give two efficient blockchain-enabled and monetary-incentivized solutions FairDownload and FairStream for P2P downloading and P2P streaming scenarios, respectively. Our designs not only ensure delivery fairness where deliverers are paid (nearly) proportional to their in-time delivery, but also guarantee exchange fairness where content consumers and content providers are also fairly treated. The fairness of each party can be assured even when other two parties collude to arbitrarily misbehave. Our protocols provide a general design of fetching content chunk from any specific position so the delivery can be resumed in the presence of unexpected interruption. Further, our systems are efficient in the sense of achieving asymptotically optimal on-chain costs and optimal delivery communication.

We implement the prototype and deploy on the Ethereum Ropsten network. Extensive experiments in both LAN and WAN settings are conducted to evaluate the on-chain costs as well as the efficiency of downloading and streaming. Experimental results show the practicality and efficiency of our protocols.

Index Terms—Content Delivery, Peer-to-Peer, Delivery Fairness, Blockchain Application, Monetary Incentivization

1 Introduction

Peer-to-peer (P2P) content delivery systems are permissionless decentralized services to seamlessly replicate contents to the end consumers [1]. These systems typically consist of a large ad-hoc network of deliverers to overcome the bandwidth bottleneck of the original content providers. In contrast to giant pre-planned content delivery networks such as Akamai [2], P2P content delivery can crowdsource unused bandwidth resources of tremendous Internet peers, thus having a wide array of benefits including robust service availability, bandwidth cost savings, and scalable peak-demand handling [3, 4]. Recently, renewed attentions to P2P content delivery are gathered [3, 5, 6] due to the fast popularization of decentralized storage networks (DSNs) [7–9]. Indeed, DSNs feature decentralized and robust *content storage*, but lack well-designed *content delivery* mechanisms catering for a prosperous content consumption market in the P2P setting, where the content shall not only be reliably stored but also must be always quickly retrievable despite potentially malicious participants [10].

The primary challenge of designing a proper P2P delivery mechanism lies in realizing strict guarantee of "fairness" against adversarial peers, i.e., a fair P2P content delivery system has to promise well-deserved items (e.g., retrieval of valid contents, well-paid rewards to spent bandwidth) to all participants [11]. Otherwise, free-riding parties can abuse the system [12–14] and cause rational ones to escape, eventually leading to possible system collapse [15].

Exchange fairness vs. delivery fairness. Conventional fairness [16–21], specifically for digital goods (such as electronic cash, signatures or videos), refers to ensuring that one party's input is kept *confidential* until it does learn the other party's input. Unfortunately, such a definition is insufficient in the P2P content delivery setting where the deliverer's input is bandwidth resource. Concretely, a deliverer may receive no reward after spending bandwidth to transfer a huge amount of *encrypted* data to a malicious content consumer, which clearly breaks the deliverer's expectation on being well-paid but does not violate conventional fairness at all. More seriously, (D)DoS attacks may be conducted by exhausting the deliverer's bandwidth.

Taking FairSwap [20] as a concrete example: the deliverer first sends the encrypted content and secure digest to the consumer, then waits for a message from the consumer (via blockchain) to confirm her receiving of ciphertext, so the deliverer can reveal his encryption key on-chain; but, in case the consumer aborts, all bandwidth used to send ciphertext is wasted, causing no reward for the deliverer. A seemingly enticing way to mitigate the above attack could be splitting the content into n smaller chunks and run FairSwap for each chunk, but the on-chain cost would then grow linear in n, leading to prohibitive on-chain cost for large contents.

- *Songlin He is with Southwest Jiaotong University, Chengdu, Sichuan, 610031, China. Email: sohe@swjtu.edu.cn.*
- *Yuan Lu is with Institute of Software Chinese Academy of Sciences, Beijing, 100190, China. Email: luyuan@iscas.ac.cn.*
- *Qiang Tang is with the School of Computer Science, The University of Sydney, NSW 2006, Australia. Email: qiang.tang@sydney.edu.au.*
- *Guiling Wang and Chase Qishi Wu are with the Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA. E-mail: {gwang, chase.wu}@njit.edu.*

- *A preliminary version of this manuscript was previously published in ESORICS'21 [1].*

- *Please send correspondence to Yuan Lu (luyuan@iscas.ac.cn).*

To capture the "specially" exchanged item, i.e., bandwidth, for deliverers, we give a more fine-grained definition of fairness in the P2P content delivery setting (ref. § 4) where *exchange fairness* denotes the conventional fairness while the *delivery fairness* states that deliverers can receive rewards (nearly) proportional to their bandwidth contribution.

Challenges of ensuring fairness. It is well-known that a fair exchange protocol cannot be designed to provide complete fairness without a trusted third party (TTP) [22]. Traditionally such a trusted middleman is played by some escrow services [19], which, however, turns to be hard to find in practice, as exemplified by many "bogus" escrow services. Recently, blockchain offers an attractive way to instantiate a non-private TTP with desired security properties. A few results [20, 21, 23, 24] leverage this innovative decentralized infrastructure to facilitate fair exchange. Unfortunately, these protocols fail to guarantee delivery fairness as they cannot capture fairness property for the specially exchanged item, i.e., bandwidth. In addition, the "transparency" property of blockchain and its limited on-chain computation power demand proper design, i.e., privacy preservation of sensitive information published on-chain and optimized on-chain costs, of proposed protocols.

Besides the natural delivery fairness for deliverers, it is equally vital to ensure exchange fairness for providers and consumers in the context of P2P content delivery, especially with the end goal to complement DSNs and enable some content providers to sell contents to consumers with delegating costly delivery/storage to a P2P network. Particularly, the content provider should be guaranteed to receive payments proportional to the amount of *correct* data learned by the consumer; meanwhile, the consumer only has to pay if indeed receiving *qualified* content.

Naïve attempts of tuning a fair exchange protocol [18–21, 25] into P2P content delivery can guarantee neither delivery fairness (as analyzed earlier) nor exchange fairness: simply running fair exchange protocols twice between the deliverers and the content providers and between the deliverers and the consumers, respectively, would leak valuable contents, raising the threat of massive content leakage. Even worse, this idea disincentivizes the deliverers as they have to pay for the whole content before making a living by delivering the content.

Insufficiency of the existing model. Though a range of existing literature [26–30] involve fairness guarantee for P2P delivery, yet their designs, to our knowledge, are presented in *non-cooperative game-theoretic* setting where rational and independent attackers free ride spontaneously without negotiating their strategies. Unfortunately, such an assumption may be particularly elusive to stand in an open environment accessible by any potentially malicious party. For example, a deliverer may collude with a consumer (or both are corrupted by an adversary) to reap the payment from the content provider without actual delivery, or a content provider may collude with consumers to prevent deliverers from receiving their deserved payments.

On the contrary, we seek to design the fair P2P content delivery protocol in the *cryptographic* sense where the security properties such as fairness can be strictly ensured and the security is against an adversary who can coordinate all malicious parties. Practically, the occurrences of tremendous real-world attacks in open systems [31] hint us how vulnerable the prior studies' heavy assumptions, i.e., no tolerance against collusion, can be and further weaken the confidence of using them in real-world P2P content delivery.

Contributions. Overall, it remains an open problem to achieve a strong fairness guarantee in P2P content delivery for *all* parties. We formalize such security intuitions into a well-defined cryptographic problem on fairness, and present a couple of efficient blockchain-based monetary-incentivized protocols to solve it. In sum,

1) We formulate the problem of P2P content delivery with desired security and efficiency goals, where fairness ensures that every party is fairly treated even if all others are corrupted or collude and arbitrarily misbehave.
2) We propose two blockchain-based P2P content delivery protocols accommodating downloading (FairDownload) and streaming (FairStream) scenarios, respectively. Both protocols attain only $\tilde{O}(\eta + \lambda)$ on-chain computational costs even in the worst case, which only relates to the small chunk size η and even smaller security parameter λ. Considering the fact that $\lambda \ll \eta$, both protocols realize asymptotically optimal delivery communication complexity, as a deliverer only sends $O(\eta + \lambda)$ bits amortized for each η-bit chunk. Both protocols also support fetching content from any of its chunk position.
3) We implement[1] both protocols with making various non-trivial optimizations to reduce their critical on-chain costs. Extensive experiments in both LAN and WAN settings show their real-world applicability.

2 PRELIMINARIES

Notations. Let $[n]$ denote $\{1, \ldots, n\}$, $[a, b]$ denote $\{a, \ldots, b\}$, $x||y$ denote concatenating x and y, $\leftarrow_{\$}$ denote uniformly random sampling, and $\lambda \in \mathbb{N}$ denote the security parameter given (sometimes implicitly) as a parameter to all cryptographic algorithms.

Global ledger**.** It provides the primitive of cryptocurrency that can deal with "coin" transfers transparently. Each entry of the dictionary $\mathsf{ledger}[\mathcal{P}_i]$ records the balance of the party $\mathcal{P}_i$. The global ledger is accessible by all system participants and can be a subroutine of *smart contract* to transact "coins" to a designated party when some conditions are met.

Merkle tree. A Merkle tree scheme consists of the algorithms $(\mathsf{BuildMT}, \mathsf{GenMTP}, \mathsf{VerifyMTP})$: $\mathsf{BuildMT}$ accepts as input a sequence of elements $m = (m_1, m_2, \cdots, m_n)$ and outputs the Merkle tree MT with root, denoted by $\mathsf{root}(\mathsf{MT})$, that commits m; GenMTP takes as input the Merkle tree MT (built for m) and the i-th element m_i, and outputs a proof π_i to attest the inclusion of m_i at the position i of m; $\mathsf{VerifyMTP}$ takes as input $\mathsf{root}(\mathsf{MT})$, the index i, the Merkle proof π_i, and m_i, and outputs either 1 (accept) or 0 (reject). The security of Merkle tree scheme ensures that: for any *probabilistic polynomial-time* (P.P.T.) adversary $\mathcal{A}$, any sequence m and any index i, conditioned on MT is a Merkle tree built for m, $\mathcal{A}$ cannot produce a fake Merkle tree proof fooling $\mathsf{VerifyMTP}$ to accept $m'_i \neq m_i \in m$ except with negligible probability given m, MT and security parameters.

1. Code: https://github.com/Blockchain-World/FairThunder.git

Verifiable decryption. We consider a specific verifiable public key encryption (VPKE) scheme consisting of the algorithms $(\mathsf{VPKE.KGen}, \mathsf{VEnc}, \mathsf{VDec}, \mathsf{ProvePKE}, \mathsf{VerifyPKE})$ and allowing the decryptor to produce the plaintext along with a proof attesting the correct decryption [32]. Specifically, KGen outputs a public-private key pair, i.e., $(h, k) \leftarrow \mathsf{VPKE.KGen}(1^\lambda)$. The public key encryption satisfies semantic security. Furthermore, the $\mathsf{ProvePKE}_k$ algorithm takes as input the private key k and the cipher c, and outputs a message m with a proof π; while the $\mathsf{VerifyPKE}_h$ algorithm takes as input the public key h and (m, c, π), and outputs $1/0$ to accept/reject the statement that $m = \mathsf{VDec}_k(c)$. Besides the semantic security, the verifiable decryption scheme need satisfy the following extra properties:

- *Completeness*. $Pr[\mathsf{VerifyPKE}_h(m, c, \pi) = 1 | (m, \pi) \leftarrow \mathsf{ProvePKE}_k(c)] = 1$, for $\forall\ c$ and $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$;
- *Soundness*. For any $(h, k) \leftarrow \mathsf{KGen}(1^\lambda)$ and c, no P.P.T. adversary $\mathcal{A}$ can produce a proof π fooling $\mathsf{VerifyPKE}_h$ to accept that c is decrypted to m' if $m' \neq \mathsf{VDec}_k(c)$ except with negligible probability;
- *Zero-Knowledge*. The proof π can be simulated by a P.P.T. simulator $\mathcal{S}_{\mathsf{VPKE}}$ taking as input only public knowledge m, h, c, hence nothing more than the truthness of the statement $(m, c) \in \{(m, c) | m = \mathsf{VDec}_k(c)\}$ is leaked.

Cryptographic primitives. We also consider the following standard cryptographic primitives: (i) a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ modeled as a random oracle; (ii) a *semantically secure* (fixed-length) symmetric encryption scheme consisting of $(\mathsf{SE.KGen}, \mathsf{SEnc}, \mathsf{SDec})$; (iii) an *existential unforgeability under chosen message attack (EU-CMA)* secure digital signature scheme consisting of $(\mathsf{SIG.KGen}, \mathsf{Sign}, \mathsf{Verify})$.

3 Building Blocks

Before diving into the protocol details, we first present two building blocks for the downloading protocol.

3.1 Verifiable Fair Delivery

To quantify a deliverer's bandwidth contribution, we put forth the notion of *verifiable fair delivery* (VFD), which enables an honest verifier $\mathcal{V}$ to verify that a sender $\mathcal{S}$ indeed transferred some amount of data to a receiver $\mathcal{R}$. It later acts as a key module in FairDownload. The high-level idea of VFD is: the receiver $\mathcal{R}$ needs to send back a signed "receipt" to acknowledge the sender $\mathcal{S}$'s bandwidth contribution and continuously receives the next chunk. Consider that the data chunks of the same size η (as a system parameter) are transferred *sequentially* starting from a chunk with index ζ, later the sender $\mathcal{S}$ can always use the latest receipt to count how many chunks are transferred and prove to $\mathcal{V}$ about the total contribution. Intuitively $\mathcal{S}$ *at most* wastes bandwidth of transferring one chunk of size η.

Syntax. The VFD protocol involving a sender $\mathcal{S}$, a receiver $\mathcal{R}$ and a verifier $\mathcal{V}$, follows the ensuing syntax:

- **Sender** $\mathcal{S}$ starts the delivery via $\mathcal{S}.\mathsf{send}()$ that inputs n sequential data chunks and their validation strings, denoted by $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, the start chunk index $\zeta \in [n]$, and there exists an efficient and global predicate $\Psi(i, c_i, \sigma_{c_i}) \rightarrow \{0, 1\}$ to check whether c_i is the i-th valid chunk due to σ_{c_i}; once the delivery starts, $\mathcal{S}$ interacts with $\mathcal{R}$ and opens an interface $\mathcal{S}.\mathsf{prove}()$ that can be invoked by $\mathcal{V}$ to generate a proof π indicating the number of sent chunks;
- **Receiver** $\mathcal{R}$ is activated by an interface $\mathcal{R}.\mathsf{recv}()$ that inputs the starting chunk index ζ and the global predicate $\Psi(\cdot)$ to interact with $\mathcal{S}$, and outputs a sequence of valid (due to $\Psi(\cdot)$) chunks starting from chunk ζ;
- **Verifier** $\mathcal{V}$ takes as input the proof π generated by $\mathcal{S}.\mathsf{prove}()$ and the start index $\zeta \in [n]$, and outputs an integer $\mathsf{ctr} \in \{0, \cdots, n\}$ as the number of sent chunks.

Properties. The VFD protocol should satisfy the following security properties:

- **Completeness**. If both $\mathcal{S}$ and $\mathcal{R}$ are honest, after $2n$ rounds, $\mathcal{S}$ is able to generate a proof π which $\mathcal{V}$ can take as input and output $\mathsf{ctr} \equiv n$, while $\mathcal{R}$ can output $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, which is $\mathcal{S}$'s input;
- **Termination**. If at least one of $\mathcal{S}$ and $\mathcal{R}$ is honest, the VFD protocol terminates within $2n$ rounds, where n is the number of chunks for a content;
- **Verifiable η delivery fairness**. When one of $\mathcal{S}$ and $\mathcal{R}$ maliciously aborts, VFD should satisfy the following delivery fairness requirements:
  - *Sender fairness*. An honest sender $\mathcal{S}$ is guaranteed to generate a proof π, which enables $\mathcal{V}$ to output at least ctr if $\mathcal{S}$ has sent $\mathsf{ctr} + 1$ valid sequential chunks. In other words, at most $\mathcal{S}$ wastes bandwidth for delivering one chunk of size η;
  - *Receiver fairness*. An honest receiver $\mathcal{R}$ is guaranteed to obtain ctr valid sequential chunks if $\mathcal{S}$ can generate a proof π, which enables $\mathcal{V}$ to output ctr.

Protocol Π_{VFD}. We consider $\mathcal{S}$ and $\mathcal{R}$ have generated public-private key pairs $(pk_\mathcal{S}, sk_\mathcal{S})$ and $(pk_\mathcal{R}, sk_\mathcal{R})$ for digital signature, respectively; and they have announced the public keys to bind to themselves. Then VFD with the global predicate $\Psi(\cdot)$ can be realized by the protocol Π_{VFD} hereunder among $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{V}$ against P.P.T. and static adversary in the *stand-alone* setting[2] assuming synchronous network:

- **Construction of sender.** The sender $\mathcal{S}$, after activated via $\mathcal{S}.\mathsf{send}()$ with the input $((c_1, \sigma_{c_1}), \ldots, (c_n, \sigma_{c_n}))$, $pk_\mathcal{S}$, $pk_\mathcal{R}$, and the start chunk index $\zeta \in [n]$, starts a timer $\mathcal{T}_\mathcal{S}$ lasting two synchronous rounds, initializes a variable $\pi_\mathcal{S} := \emptyset$, and executes as follows:
  - For each $i \in [\zeta, n]$: $\mathcal{S}$ sends $(\mathsf{deliver}, i, c_i, \sigma_{c_i})$ to $\mathcal{R}$, and waits for response message $(\mathsf{receipt}, i, \sigma^i_\mathcal{R})$ from $\mathcal{R}$. If $\mathcal{T}_\mathcal{S}$ expires before receiving the receipt, breaks the iteration; otherwise $\mathcal{S}$ verifies whether $\mathsf{Verify}(\mathsf{receipt} || i || pk_\mathcal{R} || pk_\mathcal{S}, \sigma^i_\mathcal{R}, pk_\mathcal{R}) \equiv 1$ or not, if *true*, resets $\mathcal{T}_\mathcal{S}$, outputs $\pi_\mathcal{S} = (i, \sigma^i_\mathcal{R})$ and continues to run the next iteration (i.e., increasing i by one); if *false*, breaks the iteration;
  - Upon that $\mathcal{S}.\mathsf{prove}()$ is invoked, it returns $\pi_\mathcal{S}$ as the VFD proof and halts.
- **Construction of receiver.** The receiver $\mathcal{R}$, after activated via $\mathcal{R}.\mathsf{recv}()$ with the input $pk_\mathcal{S}$, the start chunk index $\zeta \in [n]$, and $(pk_\mathcal{R}, sk_\mathcal{R})$, starts a timer $\mathcal{T}_\mathcal{R}$ lasting two synchronous rounds and operates as: for each $j \in [\zeta, n]$: $\mathcal{R}$ waits for $(\mathsf{deliver}, j, c_j, \sigma_{c_j})$ from $\mathcal{S}$ and halts if

2. To defend against *replay* attack in concurrent sessions, it is trivial to let the authenticated messages include a unique session id field.

$\mathcal{T}_{\mathcal{R}}$ expires before receiv
wise $\mathcal{R}$ verifies whether
resets $\mathcal{T}_{\mathcal{R}}$, outputs $(c_j, \sigma_$
$\mathcal{S}$ where $\sigma^i_{\mathcal{R}} \leftarrow$ Sign(rec
false. Note that the glob
essentially it just perforr

- **Construction of verifier**
$\mathcal{V}$ parses it into $(i, \sigma^i_{\mathcal{R}})$
$i \geq \zeta$, and Verify(receipt
not; if *true*, it outputs ct
0. Recall that Verify is to

Lemma 1. *In the synchronous alone setting, the protocol* Π_{VF} *ness and the verifiable* η *deliv P.P.T. adversary that corrupts o*

Proof. The *completeness* and to see. For the η *delivery fairness* of VFD, on one hand, for the malicious $\mathcal{R}^*$ corrupted by $\mathcal{A}$, if $\mathcal{V}$ takes the honest $\mathcal{S}$'s proof and can output ctr, then $\mathcal{S}$ *at most* has sent $(\mathsf{ctr}+1)$ chunk-validation pairs, i.e., (c_i, σ_{c_i}), to $\mathcal{A}$. Overall, $\mathcal{S}$ *at most* wastes bandwidth of delivering one chunk of size η. On the other hand, the malicious $\mathcal{S}^*$ corrupted by $\mathcal{A}$ may abort after receiving the receipt, say with index ζ' ($\zeta' \in [\zeta, n]$) . In that case, $\mathcal{R}$ is also guaranteed to receive a valid sequence of $((c_\zeta, \sigma_{c_\zeta}), \cdots, (c_{\zeta'}, \sigma_{c_{\zeta'}}))$ with overwhelming probability, unless $\mathcal{A}$ can forge $\mathcal{R}$'s signature. However, it requires $\mathcal{A}$ to break the underlying EU-CMA signature scheme, which is of negligible probability. Hence, the η delivery fairness of VFD is rigorously guaranteed.

3.2 Structured Key Derivation

To keep the content confidentiality during delivery, we encrypt the content chunks and delegate to deliverers. Later only the decryption keys for chunks need to be revealed (via blockchain) to a consumer. However, a naive approach by revealing the n decryption keys on-chain results in linear storage costs. We therefore propose an efficient *structured key derivation* (SKD) scheme to reduce the number of revealed elements, thus considerably decrease the on-chain storage costs. Specifically, to encrypt n data chunks[3], a sender $\mathcal{S}$ can utilize a randomly sampled master key mk to deterministically generate a key tree KT with n leaf nodes as (symmetric) encryption keys; later for a receiver $\mathcal{R}$ who received ctr sequential encrypted chunks starting from the chunk ζ, $\mathcal{S}$ only needs to reveal few elements for $\mathcal{R}$ to recover the ctr decryption keys. In the best case revealing only one element is sufficient and in the worst case $O(\log n)$ elements are needed, yielding $\tilde{O}(1)$ costs for all cases. Fig. 1 gives two illustrations of the key derivation scheme.

The scheme consists of three algorithms, and the details are presented in Algorithms 1, 2 and 3:

- $\mathsf{KT} \leftarrow \mathsf{SKD.GenSubKeys}(n, mk_{\mathsf{hash}})$: It takes as input an $n \in \mathbb{Z}^+$ and the hash of a randomly sampled master key, i.e., $mk_{\mathsf{hash}} := \mathcal{H}(mk)$, and outputs a key tree KT with n leaf nodes.
- $rk \leftarrow \mathsf{SKD.RevealKeys}(n, \zeta, \mathsf{ctr}, mk_{\mathsf{hash}})$: It takes as input $n \in \mathbb{Z}^+$, $\zeta \in [n]$, $\mathsf{ctr} \in [n-\zeta+1]$, and the hash of the master key mk_{hash}, and outputs an array rk containing the minimum number of elements in KT that can recover the ctr keys.
- $ks \leftarrow \mathsf{SKD.RecoverKeys}(n, \mathsf{ctr}, rk)$: It takes as input $n \in \mathbb{Z}^+$, $\mathsf{ctr} \in [n-\zeta+1]$, and the revealed key array rk, and outputs the recovered keys ks.

Property arguments. The scheme SKD satisfies the following properties: (i) *Correctness*. The ctr recovered keys ks by a receiver $\mathcal{R}$ is the same as the keys in key tree KT that generated by the sender $\mathcal{S}$. This follows from that the hash process is deterministic; (ii) *Succinctness*. In the worst case, only $\tilde{O}(1)$ elements are revealed. This follows from the tree architecture of the key tree KT; (iii) *Robustness*. The scheme is robust, e.g., an adversary cannot derive the sender's master key from the revealed elements, due to the random oracle model of hash function.

3. W.l.o.g., we assume $n = 2^k$ for $k \in \mathbb{Z}^+$ for presentation simplicity.

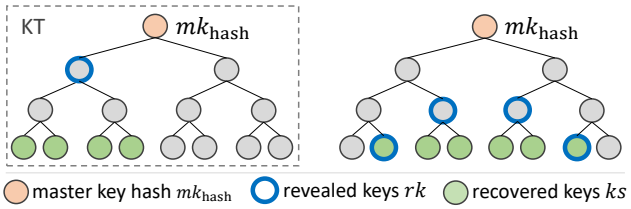


Fig. 1: Illustrations of the key derivation scheme. For the left-side one: $n = 8$, $\mathsf{ctr} = 4$ and $\zeta = 1$. For the right-side one: $n = 8$, $\mathsf{ctr} = 6$ and $\zeta = 2$.

Algorithm 1 GenSubKeys algorithm

Input: n, mk_{hash}
Output: a $(2n-1)$-array KT
let KT be an empty array where its size $|\mathsf{KT}| = 2n-1$
$\mathsf{KT}[0] = mk_{\mathsf{hash}}$
if $n \equiv 1$ **then**
  return KT
if $n > 1$ **then**
  for i in $[0, n-2]$ **do**
    $\mathsf{KT}[2i+1] = \mathcal{H}(\mathsf{KT}[i]||0)$
    $\mathsf{KT}[2i+2] = \mathcal{H}(\mathsf{KT}[i]||1)$
return KT

Algorithm 2 RevealKeys algorithm

Input: n, ζ, ctr, and mk_{hash}
Output: rk, an array of revealed keys
let rk and ind be empty arrays
let $st = n + \zeta - 2$
$\mathsf{KT} \leftarrow \mathsf{GenSubKeys}(n, mk_{\mathsf{hash}})$
if ctr $\equiv 1$ **then**
  rk appends $(st, \mathsf{KT}[st])$
  return rk
if ctr $\equiv 2$ **then**
  if st is odd **then**
    rk appends $((st-1)/2, \mathsf{KT}[(st-1)/2])$
  else
    rk appends $(st, \mathsf{KT}[st])$
    rk appends $(st+1, \mathsf{KT}[st+1])$
  return rk
for i in $[0, \mathsf{ctr}-1]$ **do**
  $ind[i] = st + i$
if st is even **then**
  rk appends $(st, \mathsf{KT}[st])$
  ind removes $ind[0]$
while *true* **do**
  let t be an empty array
  for j in $[0, \lfloor |ind|/2 \rfloor - 1]$ **do**
    $p_l = (ind[2j]-1)/2$
    $p_r = (ind[2j+1]-2)/2$
    ▷ merge elements with the same parent node in KT
    if $p_l \equiv p_r$ **then**
      t appends p_l
    else
      t appends $ind[2j]$
      t appends $ind[2j+1]$
  if $|ind|$ is odd **then**
    t appends $ind[|ind|-1]$
  if $|ind| \equiv |t|$ **then**
    break
  $ind = t$
for x in $[0, |ind|-1]$ **do**
  rk appends $(ind[x], \mathsf{KT}[ind[x]])$
return rk

Algorithm 3 RecoverKeys algorithm

Input: n, ctr, and rk
Output: a ctr-sized array ks
let ks be an empty array
for each $(i, \mathsf{KT}[i])$ in rk **do**
  $n_i = 2^{(\log n - \lfloor \log(i+1) \rfloor)}$
  $v_i = \mathsf{GenSubKeys}(n_i, \mathsf{KT}[i])$
  ks appends $v_i[n_i - 1 : 2n_i - 2]$
return ks

4 Formalizing P2P Content Delivery

Now we formulate the problem of fair P2P content delivery. Blockchain is leveraged to play the role of a non-private TTP.

Furthermore, blockchain-enabled cryptocurrencies and the guaranteed execution of smart contracts can provide monetary incentivization to facilitate the guarantee of fairness.

4.1 System Model

Participants. We consider the following parties:

- *Content Provider* $\mathcal{P}$ is an entity that owns the original content m composed of n chunks,[4] satisfying a publicly known predicate $\phi(\cdot)$,[5] and $\mathcal{P}$ is willing to sell to any user of interest. Meanwhile, $\mathcal{P}$ would like to delegate the delivery of m to a set of deliverers with promise to pay $\$_{\mathcal{P}}$ for each successfully delivered chunk.
- *Content Deliverer* $\mathcal{D}$ contributes its idle bandwidth resources to deliver the content on behalf of the provider $\mathcal{P}$ and would receive the payment proportional to the amount of delivered data.
- *Content Consumer* $\mathcal{C}$ is an entity that would pay $\$_{\mathcal{C}}$ for each chunk in content m by interacting with $\mathcal{P}$ and $\mathcal{D}$.

Adversarial model. We consider the adversary $\mathcal{A}$ with the following standard abilities [34]: $\mathcal{A}$ is static and can corrupt some parties only before the course of protocol executions and $\mathcal{A}$ is restricted to P.P.T. algorithms.

Communication model. We adopt the *synchronous* network model [35, 36] of authenticated point-to-point channels to describe the ability of $\mathcal{A}$ on controlling communications. W.l.o.g., we consider a global clock in the system, and $\mathcal{A}$ can delay the messages up to a clock round [23, 37].

Arbiter smart contract $\mathcal{G}$. The system is in a hybrid model with oracle access to an arbiter smart contract $\mathcal{G}$. The contract $\mathcal{G}$ is a stateful ideal functionality that leaks all its internal states to the adversary $\mathcal{A}$ and all parties, while allowing to pre-specify some immutable conditions to transact "coins" over the cryptocurrency ledger, thus "mimicking" the contracts in real life transparently. In practice, the contract can be instantiated via many real-world blockchains like Ethereum [38]. Throughout this paper, the details of $\mathcal{G}$ follow the conventional pseudo-code notations in [37].

4.2 Design Goals

Syntax. A P2P content delivery protocol $\Pi = (\mathcal{P}, \mathcal{D}, \mathcal{C})$ executes among a set of parties modeled as interactive Turing machines (ITMs), and contains two phases:

- *Preparation*. The provider $\mathcal{P}$ takes as input public parameters and the content m where $\phi(m) \equiv 1$ and outputs some auxiliary data, e.g., encryption keys; a deliverer $\mathcal{D}$ takes as input public parameters and outputs some auxiliary data, e.g., encrypted content; the consumer $\mathcal{C}$ is not involved in this phase. Note that $\mathcal{P}$ deposits a budget of $n \cdot \$_{\mathcal{P}}$ in ledger to incentivize deliverers to participate in.
- *Delivery*. The provider $\mathcal{P}$ and the deliverer take as input their auxiliary data obtained in the previous phase, respectively, and they would receive the deserved payment; the consumer $\mathcal{C}$ takes as input public parameters and outputs the content m with $\phi(m) \equiv 1$. Note that $\mathcal{C}$ has a budget of $n \cdot \$_{\mathcal{C}}$ in ledger to "buy" the content m with $\phi(m) \equiv 1$ and $\$_{\mathcal{C}} > \$_{\mathcal{P}}$.

A fair P2P content delivery protocol Π shall meet all following properties, including completeness, fairness, confidentiality, timeliness, and non-trivial efficiency.

Completeness. For any content predicate $\phi(\cdot)$ in the form of $\phi(m) = [\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m]$, conditioned on $\mathcal{P}, \mathcal{D}$ and $\mathcal{C}$ are all honest, the protocol Π realizes:

- The consumer $\mathcal{C}$ receives the qualified content m with $\phi(m) \equiv 1$, and its balance in the global ledger$[\mathcal{C}]$ would reduce by $n \cdot \$_{\mathcal{C}}$, where $\$_{\mathcal{C}}$ represents the amount paid by $\mathcal{C}$ for each content chunk.
- The deliverer $\mathcal{D}$ obtains the payment $n \cdot \$_{\mathcal{P}}$ from the global ledger, where $\$_{\mathcal{P}}$ represents the amount paid by $\mathcal{P}$ to $\mathcal{D}$ for delivering a content chunk to the consumer.
- The provider $\mathcal{P}$ receives its well-deserved payments over the ledger, namely, ledger$[\mathcal{P}]$ would increase by $n \cdot (\$_{\mathcal{C}} - \$_{\mathcal{P}})$ as it receives $n \cdot \$_{\mathcal{C}}$ from the consumer while it pays out $n \cdot \$_{\mathcal{P}}$ to the deliverer.

Fairness. The protocol Π shall meet the following fairness requirements:

- **Exchange fairness for consumer**. For $\forall$ corrupted P.P.T. $\mathcal{D}^*$ and $\mathcal{P}^*$ controlled by $\mathcal{A}$, it is guaranteed to the honest consumer $\mathcal{C}$ with overwhelming probability that: the ledger$[\mathcal{C}]$ decreases by $\ell \cdot \$_{\mathcal{C}}$ only if $\mathcal{C}$ indeed receives a sequence of chunks $(m_i, \ldots, m_j)$ belonging to m where $\ell = (j - i + 1)$ and $\phi(m) \equiv 1$. This property states that $\mathcal{C}$ pays proportional to valid chunks it *de facto* obtains.
- **Delivery fairness for deliverer.** For $\forall$ malicious P.P.T. $\mathcal{C}^*$ and $\mathcal{P}^*$ corrupted by $\mathcal{A}$, it is assured to the honest deliverer $\mathcal{D}$ that: if $\mathcal{D}$ sent overall $O(\ell \cdot \eta + 1)$ bits during the protocol, $\mathcal{D}$ should *at least* obtain the payment of $(\ell - 1) \cdot \$_{\mathcal{P}}$. Intuitively, if $\mathcal{D}$ spent bandwidth for delivering ℓ valid chunks, at least it will receive the deserved payment for delivering $\ell - 1$ chunks, and at most the bandwidth for delivering one chunk of size η can be wasted. The unpaid delivery is bounded by $O(\eta)$ bits where η is a system parameter.
- **Exchange fairness for provider**. For $\forall$ corrupted P.P.T. $\mathcal{C}^*$ and $\mathcal{D}^*$ controlled by $\mathcal{A}$, it is ensured to the honest provider $\mathcal{P}$ that: if $\mathcal{A}$ can output $\eta \cdot \ell$ bits consisted in the content m, the provider $\mathcal{P}$ shall obtain at least $(\ell - 1) \cdot (\$_{\mathcal{C}} - \$_{\mathcal{P}})$ net income, i.e., ledger$[\mathcal{P}]$ increases by $(\ell - 1) \cdot (\$_{\mathcal{C}} - \$_{\mathcal{P}})$, with all except negligible probability. $\mathcal{P}$ is ensured that *at most* $O(\eta)$-bit, i.e., one chunk of size η, content are revealed without being paid.

Confidentiality against deliverers. This is to protect copyrighted data against probably corrupted deliverers, otherwise a malicious consumer may pretend to be or collude with a deliverer to obtain the plaintext content without paying for the provider, which violates the exchange fairness for $\mathcal{P}$. Informally, we require that the corrupted $\mathcal{D}^*$ on receiving protocol scripts (e.g., the delegated content chunks from the provider) cannot produce the provider's input content with all but negligible probability in a delivery session.

Timeliness. When at least one of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ is honest (i.e., others are corrupted by $\mathcal{A}$), the honest ones are ensured to halt in $O(n)$ synchronous rounds where n is the number of

4. Remark that the content m is *dividable* in the sense that each chunk is independent to others, e.g., each chunk is a small 10-second video.

5. Throughout the paper, we consider that the predicate ϕ is in the form of $\phi(m) = [\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m]$, where root is the Merkle tree root of the content m. In practice, it can be acquired from a semi-trusted third party, such as BitTorrent forum sites [19] or VirusTotal [33].

content chunks. At completion or abortic
tioned fairness and confidentiality are alw
Non-trivial efficiency. We require the nec
efficiency to preclude potentially trivial ap

- The messages sent to $\mathcal{G}$ by honest par
bounded by $\tilde{O}(1)$ bits, which rules ou
using the smart contract to directly re
- In the delivery phase, the messages
are uniformly bounded by $n \cdot \lambda$ bits, v
cryptographic parameter, thus ensur
smaller than the content size $|m|$. T
utilizing peers for content delivery so
bandwidth upon the completion of p
and excludes the notion of delivery o

Remarks. We make the following discu
above definitions: (i) $\phi(\cdot)$ is a public par
all parties before the protocol execution
requirements have already implied the ca
versary corrupts one party of $\mathcal{P}$, $\mathcal{D}$ and
since whenever the adversary corrupts t
let one of these corrupted two follow the
(iii) the global ledger model captures the
coin transfers, from this perspective, it is unnecessary to distinguish the usage of either the account-based model or the unspent transaction output (UTXO)-based model; (iv) like all cryptographic protocols, it does not make sense to consider all parties are corrupted, so do we not; (v) the deliverer and the provider might lose well-deserved payment, but *at most* lose that for one chunk, i.e., the level of unfairness is rigorously bounded; (vi) upon finishing the one-time preparation phase, the delivery phase is repeatable; (vii) it is reasonable that there are a great number of deliverers and at least some of them can be honest. Hence, if a consumer fails to obtain the entire content due to unexpected situations occurred in the middle of transmission, it can always ask another deliverer to initiate a new session and fetch the remaining chunks. Our designs allow consumers to fetch the content chunks from any specific position.

5 FairDownload: Fair P2P Downloading

This section presents the fair P2P downloading protocol Π_{FD}, allowing the consumers to view the content after obtaining (partial or all) the chunks, namely *view-after-delivery*.

5.1 FairDownload Overview

The protocol Π_{FD} can be constructed with the modules of verifiable fair delivery (VFD) and structured key derivation (SKD), and proceeds in *Prepare*, *Deliver* and *Reveal* phases as depicted in Fig. 2. The core ideas are highlighted as follows:

- The provider $\mathcal{P}$ encrypts each chunk, signs the encrypted chunks, and delegates to the deliverer $\mathcal{D}$; the deliverer (as the sender $\mathcal{S}$) and the consumer $\mathcal{C}$ (as the receiver $\mathcal{R}$) can run a specific instance of VFD, where the global predicate $\Psi(\cdot)$ is instantiated to verify that each chunk must be correctly signed by $\mathcal{P}$; additionally, the non-interactive honest verifier $\mathcal{V}$ in VFD is instantiated via smart contract, hence upon the contract receives a VFD proof from $\mathcal{D}$ claiming the in-time delivery of ctr chunks, it can assert that $\mathcal{C}$ indeed received ctr encrypted chunks signed by the provider, who can then present to reveal the elements (via smart contract) for decrypting these ctr chunks.
- The structured key derivation SKD scheme can minimize the storage costs of revealed elements on-chain, i.e., a short $\tilde{O}(\lambda)$-bit message. To ensure confidentiality against malicious deliverers, the revealed elements are encrypted using the consumer $\mathcal{C}$'s public key. Once the decryption keys are recovered and the raw content chunks are derived, $\mathcal{C}$ can check the validity of each chunk (by comparing the digest of each decrypted chunk with the corresponding leaf node value in the merkle tree of the content) and raise complaint to smart contract if any invalid chunk is found. $\mathcal{C}$ can get refund for a valid proof of misbehavior (PoM). Otherwise the provider eventually gets the payment (after timeout).

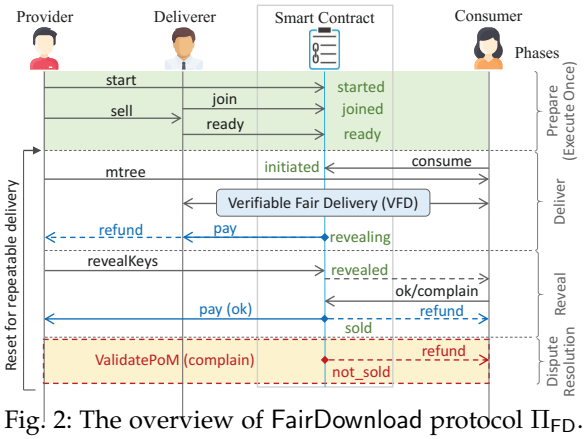


Fig. 2: The overview of FairDownload protocol Π_{FD}.

5.2 Arbiter Contract $\mathcal{G}_d^{\mathsf{ledger}}$ for Downloading

The arbiter contract $\mathcal{G}_d^{\mathsf{ledger}}$ (abbr. $\mathcal{G}_d$) shown in Fig. 3 is a stateful ideal functionality having accesses to ledger to assist the fair delivery via downloading. We remark that: (i) The description of $\mathcal{G}_d$ captures the essence of real-world smart contracts as it reflects that the Turing-complete smart contract can be seen as a stateful program to transparently handle pre-specified functionalities, and captures that a smart contract can access the cryptocurrency ledger to faithfully deal with conditional payments upon its own internal states; (ii) $\mathcal{G}_d$ can invoke the VFD verifier $\mathcal{V}$ as a subroutine. VFD's predicate $\Psi(\cdot)$ is instantiated to verify that each chunk is indeed signed by the provider $\mathcal{P}$; (iii) the ValidateRKeys and ValidatePoM subroutines allow a consumer to prove to the contract if the provider $\mathcal{P}$ behaves maliciously.

5.3 Π_{FD}: FairDownload Protocol for P2P Downloading

Now we present the details of Π_{FD} considering one deliverer, while multi-deliverer scheme is discussed later in § 6. The protocol aims to deliver a content m made of n chunks with a-priori known digest in the form of Merkle tree root, i.e., root_m. We omit the session id *sid* and the content digest root_m during the protocol description since they remain the same within a delivery session.

Phase I for Prepare. The provider $\mathcal{P}$ and the deliverer $\mathcal{D}$ interact with the contract functionality $\mathcal{G}_d$ in this phase as:

The Arbiter Contract Functionality $\mathcal{G}_d^{\mathsf{ledger}}$ for P2P Downloading

The arbiter contract $\mathcal{G}_d$ has access to the ledger, and it interacts with the provider $\mathcal{P}$, the deliverer $\mathcal{D}$, the consumer $\mathcal{C}$ and the adversary $\mathcal{A}$. It locally stores the times of repeatable delivery θ, the number of content chunks n, the content digest root_m, the price $\$_{\mathcal{P}}$, $\$_{\mathcal{C}}$ and $\$_{\mathsf{pf}}$, the number of delivered chunks ctr (initialized as 0), the start chunk index ζ of request content, the public addresses $pk_{\mathcal{P}}, pk_{\mathcal{D}}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}$, the revealed keys' hash erk_{hash}, the state Σ and three timers $\mathcal{T}_{\mathsf{round}}$ (implicitly), $\mathcal{T}_{\mathsf{deliver}}$, and $\mathcal{T}_{\mathsf{dispute}}$.

Phase 1: Prepare

- On receive $(\mathsf{start}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}})$ from $\mathcal{P}$:
  - assert $\mathsf{ledger}[\mathcal{P}] \geq (\theta \cdot (n \cdot \$_{\mathcal{P}} + \$_{\mathsf{pf}})) \wedge \Sigma \equiv \emptyset$
  - store $pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}}$
  - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] - \theta \cdot (n \cdot \$_{\mathcal{P}} + \$_{\mathsf{pf}})$ and $\Sigma := \mathsf{started}$
  - send $(\mathsf{started}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}})$ to all entities
- On receive $(\mathsf{join}, pk_{\mathcal{D}})$ from $\mathcal{D}$:
  - assert $\Sigma \equiv \mathsf{started}$
  - store $pk_{\mathcal{D}}$ and let $\Sigma := \mathsf{joined}$
  - send $(\mathsf{joined}, pk_{\mathcal{D}})$ to all entities
- On receive $(\mathsf{prepared})$ from $\mathcal{D}$:
  - assert $\Sigma \equiv \mathsf{joined}$, and let $\Sigma := \mathsf{ready}$
  - send (ready) to all entities

Phase 2: Deliver

- On receive $(\mathsf{consume}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}, \zeta)$ from $\mathcal{C}$:
  - assert $\theta > 0$, $\zeta \in [n]$ and then store ζ
  - assert $\mathsf{ledger}[\mathcal{C}] \geq (n - \zeta + 1) \cdot \$_{\mathcal{C}} \wedge \Sigma \equiv \mathsf{ready}$
  - store $pk_{\mathcal{C}}, vpk_{\mathcal{C}}$
  - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] - (n - \zeta + 1) \cdot \$_{\mathcal{C}}$
  - start a timer $\mathcal{T}_{\mathsf{deliver}}$ and let $\Sigma := \mathsf{initiated}$
  - send $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}, \zeta)$ to all entities
- On receive $(\mathsf{delivered})$ from $\mathcal{C}$ or $\mathcal{T}_{\mathsf{deliver}}$ times out:
  - assert $\Sigma \equiv \mathsf{initiated}$ and $\mathsf{ctr} \equiv 0$
  - send $(\mathsf{getVFDProof})$ to $\mathcal{D}$, and wait for two rounds to receive the proof $\pi_{\mathsf{VFD}} = (\mathsf{receipt}, i, \sigma_{\mathcal{C}}^i)$, assert $i \in [\zeta, n]$, then execute $\mathsf{verifyVFDProof}(\pi_{\mathsf{VFD}})$ to let $\mathsf{ctr} := (i - \zeta + 1)$
  - let $\mathsf{ledger}[\mathcal{D}] := \mathsf{ledger}[\mathcal{D}] + \mathsf{ctr} \cdot \$_{\mathcal{P}}$
  - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n - \mathsf{ctr}) \cdot \$_{\mathcal{P}}$
  - store ctr, let $\Sigma := \mathsf{revealing}$, and send $(\mathsf{revealing}, \mathsf{ctr}, \zeta)$ to all entities

Phase 3: Reveal

- On receive $(\mathsf{revealKeys}, erk)$ from $\mathcal{P}$:
  - assert $\Sigma \equiv \mathsf{revealing}$
  - store erk (essentially erk's hash) and start a timer $\mathcal{T}_{\mathsf{dispute}}$
  - let $\Sigma := \mathsf{revealed}$
  - send $(\mathsf{revealed}, erk)$ to all entities
- Upon $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{dispute}}$
  - $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + \mathsf{ctr} \cdot \$_{\mathcal{C}} + \$_{\mathsf{pf}}$
  - $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \zeta + 1 - \mathsf{ctr}) \cdot \$_{\mathcal{C}}$
  - let $\Sigma := \mathsf{sold}$ and send (sold) to all entities
  - reset( )

▷ Below is the dispute resolution

- On receive $(\mathsf{wrongRK})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} < \mathcal{T}_{\mathsf{dispute}}$
  - if $(\mathsf{ValidateRKeys}(n, \zeta, \mathsf{ctr}, erk) \equiv false)$:
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \zeta + 1) \cdot \$_{\mathcal{C}} + \$_{\mathsf{pf}}$
    - let $\Sigma := \mathsf{not\_sold}$ and send $(\mathsf{not\_sold})$ to all entities
    - reset( )
- On receive $(\mathsf{PoM}, i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi_{\mathsf{MT}}^i, rk, erk, \pi_{\mathsf{VD}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{dispute}}$ times out:
  - assert $\Sigma \equiv \mathsf{revealed}$ and current time $\mathcal{T} < \mathcal{T}_{\mathsf{dispute}}$
  - invoke the $\mathsf{ValidatePoM}(i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi_{\mathsf{MT}}^i, rk, erk, \pi_{\mathsf{VD}})$ subroutine, if $true$ is returned:
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n - \zeta + 1) \cdot \$_{\mathcal{C}} + \$_{\mathsf{pf}}$
    - let $\Sigma := \mathsf{not\_sold}$ and send $(\mathsf{not\_sold})$ to all entities
    - reset( )

▷ Reset to the ready state for repeatable delivery

- function reset( ):
  - assert $\Sigma \equiv \mathsf{sold}$ or $\Sigma \equiv \mathsf{not\_sold}$
  - set ctr, $\mathcal{T}_{\mathsf{deliver}}$, $\mathcal{T}_{\mathsf{dispute}}$, ζ as 0
  - nullify $pk_{\mathcal{C}}$ and $vpk_{\mathcal{C}}$
  - let $\theta := \theta - 1$, and $\Sigma := \mathsf{ready}$
  - send (ready) to all entities

Fig. 3: The arbiter contract functionality $\mathcal{G}_d^{\mathsf{ledger}}$. "Sending to all entities" captures the contract is transparent to the public.

- The provider $\mathcal{P}$ deploys contracts and starts [6] Π_{FD} by taking as input the security parameter λ, the incentive parameters $\$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}} \in \mathbb{N}$, where $\$_{\mathsf{pf}}$ is the *penalty fee*[7] in a delivery session to discourage the misbehavior from the provider $\mathcal{P}$, the number of times θ of repeatable delivery allowed for the contract, the n-chunk content $m = (m_1, \ldots, m_n) \in \{0,1\}^{\eta \times n}$ satisfying $\mathsf{root}(\mathsf{BuildMT}(m)) \equiv \mathsf{root}_m$ where root_m is the content digest in the form of Merkle tree root, and executes $(pk_{\mathcal{P}}, sk_{\mathcal{P}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$, and sends $(\mathsf{start}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}})$ to $\mathcal{G}_d$.
- Upon $\Sigma \equiv \mathsf{joined}$, the provider $\mathcal{P}$ would execute:
  - Randomly samples a master key $mk \leftarrow_{\$} \{0,1\}^\lambda$, and runs $\mathsf{KT} \leftarrow \mathsf{SKD.GenSubKeys}(n, \mathcal{H}(mk))$; stores mk and KT locally;
  - Uses the leaf nodes of KT, namely $\mathsf{KT}[n-1:2n-2]$ to encrypt $(m_1, \ldots, m_n)$ to get $c = (c_1, \ldots, c_n) \leftarrow (\mathsf{SEnc}_{\mathsf{KT}[n-1]}(m_1), \ldots, \mathsf{SEnc}_{\mathsf{KT}[2n-2]}(m_n))$;
  - Signs the encrypted chunks to obtain the sequence $((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n}))$ where the signature $\sigma_{c_i} \leftarrow \mathsf{Sign}(i||c_i, sk_{\mathcal{P}}), i \in [n]$; meanwhile, computes $\mathsf{MT} \leftarrow \mathsf{BuildMT}(m)$ and signs the Merkle tree MT to obtain $\sigma_{\mathcal{P}}^{\mathsf{MT}} \leftarrow \mathsf{Sign}(\mathsf{MT}, sk_{\mathcal{P}})$, then locally stores $(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ and sends $(\mathsf{sell}, ((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n})))$ to $\mathcal{D}$;
  - Waits for (ready) from $\mathcal{G}_d$ to enter the next phase.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving $(\mathsf{started}, pk_{\mathcal{P}}, \mathsf{root}_m, \theta, n, \$_{\mathcal{P}}, \$_{\mathcal{C}}, \$_{\mathsf{pf}})$ from $\mathcal{G}_d$, executes $(pk_{\mathcal{D}}, sk_{\mathcal{D}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$, and sends $(\mathsf{join}, pk_{\mathcal{D}})$ to $\mathcal{G}_d$;
  - Waits for $(\mathsf{sell}, ((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n})))$ from $\mathcal{P}$ and then: for every (c_i, σ_{c_i}) in the sell message, asserts that $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$; if hold, sends $(\mathsf{prepared})$ to $\mathcal{G}_d$, and stores $((c_1, \sigma_{c_1}), \cdots, (c_n, \sigma_{c_n}))$ locally;
  - Waits for (ready) from $\mathcal{G}_d$ to enter the next phase.

At the end of this phase, $\mathcal{P}$ owns a master key mk, the key tree KT, and the Merkle tree MT while $\mathcal{D}$ receives the encrypted content chunks and is ready to deliver.

Phase II for Deliver. The consumer $\mathcal{C}$, the provider $\mathcal{P}$, and the deliverer $\mathcal{D}$ interact with $\mathcal{G}_d$ in this phase as:

- The consumer $\mathcal{C}$ would execute as follows:
  - Asserts $\Sigma \equiv \mathsf{ready}$, runs $(pk_{\mathcal{C}}, sk_{\mathcal{C}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$ and $(vpk_{\mathcal{C}}, vsk_{\mathcal{C}}) \leftarrow \mathsf{VPKE.KGen}(1^\lambda)$, and sends

6. $\mathcal{P}$ can retrieve the deposits of $\$_{\mathcal{P}}$ and $\$_{\mathsf{pf}}$ back if there is no deliverer responds timely.

7. $\$_{\mathsf{pf}}$ can be set proportional to $(n \times \$_{\mathcal{C}})$ in case $\mathcal{P}$ deliberately lowers such an amount.

Algorithm 4 ValidateRKeys algorithm

Input: n, ζ, ctr and erk
Output: $true/false$; whether the correct number (i.e., ctr) of decryption keys can be recovered
if $n \equiv$ ctr and $|erk| \equiv 1$ and the position of $erk[0] \equiv 0$ **then**
  return $true$ ▷ root of KT
Initialize $chunks\_index$ as a set of numbers $\{(n+\zeta-2), \ldots, (n+\zeta+\mathsf{ctr}-3)\}$
for each $(i, \_)$ in erk **do**
  $d_i = \log(n) - \lfloor \log(i+1) \rfloor$
  $l_i = i, r_i = i$
  if $d_i \equiv 0$ **then**
    $chunks\_index$ removes i
  else
    while $(d_i\text{--}) > 0$ **do**
      $l_i = 2l_i + 1$
      $r_i = 2r_i + 2$
    $chunks\_index$ removes the elements from l_i to r_i
if $chunks\_index \equiv \emptyset$ **then**
  return $true$
return $false$

$(\mathsf{consume}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}, \zeta)$ to $\mathcal{G}_d$, where ζ indicates the start chunk index of the request content, e.g., $\zeta = 1$ indicates requesting from the first chunk;
- Upon receiving the message $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ from $\mathcal{P}$ where $\mathsf{Verify}(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}}, pk_{\mathcal{P}}) \equiv 1$ and $\mathsf{root}(\mathsf{MT}) \equiv \mathsf{root}_m$, stores the Merkle tree MT and then activates the receiver $\mathcal{R}$ in the VFD subroutine by invoking $\mathcal{R}.\mathsf{recv}()$ and instantiating the external validation function $\Psi(i, c_i, \sigma_{c_i})$ as $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}})$, and then waits for the execution of VFD to return the delivered chunks $((c_\zeta, \sigma_{c_\zeta}), (c_{\zeta+1}, \sigma_{c_{\zeta+1}}), \cdots)$ and stores them; upon receiving the last (i.e., n-th) chunk, sends $(\mathsf{delivered})$ to $\mathcal{G}_d$;
- Waits for $(\mathsf{revealing}, \mathsf{ctr}, \zeta)$ from $\mathcal{G}_d$ to enter the next phase.

- The provider $\mathcal{P}$ executes as follows during this phase: upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}, \zeta)$ from $\mathcal{G}_d$, asserts $\Sigma \equiv \mathsf{initiated}$, and sends $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ to $\mathcal{C}$, and then enters the next phase.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, vpk_{\mathcal{C}}, \zeta)$ from $\mathcal{G}_d$: asserts $\Sigma \equiv \mathsf{initiated}$, and then activates the sender $\mathcal{S}$ in the VFD module by invoking $\mathcal{S}.\mathsf{send}()$ and instantiating the external validation function $\Psi(i, c_i, \sigma_{c_i})$ as $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}})$, and feeds VFD module with input $((c_\zeta, \sigma_{c_\zeta}), \ldots, (c_n, \sigma_{c_n}))$;
  - Upon receiving $(\mathsf{getVFDProof})$ from $\mathcal{G}_d$, sends the latest proof, namely $(\mathsf{receipt}, i, \sigma_{\mathcal{C}}^i)$ to $\mathcal{G}_d$;
  - Waits for $(\mathsf{revealing}, \mathsf{ctr}, \zeta)$ from $\mathcal{G}_d$ to halt.

At the end of this phase, $\mathcal{C}$ receives the valid sequence of encrypted chunks $(c_\zeta, c_{\zeta+1}, \ldots)$, and $\mathcal{D}$ receives the payment for the bandwidth contribution of delivered chunks, and the contract records the start chunk index ζ and the number of delivered chunks ctr.

Phase III for Reveal. This phase is completed by $\mathcal{P}$, $\mathcal{C}$ and the arbiter contract $\mathcal{G}_d$, which proceeds as follows:

- The provider $\mathcal{P}$ operates as follows during this phase:
  - Asserts the state $\Sigma \equiv \mathsf{revealing}$, executes $rk \leftarrow \mathsf{SKD.RevealKeys}(n, \zeta, \mathsf{ctr}, \mathcal{H}(mk))$ to generate the revealed elements rk, and encrypts rk via $erk \leftarrow \mathsf{VEnc}_{vpk_{\mathcal{C}}}(rk)$, and then sends $(\mathsf{revealKeys}, erk)$ to $\mathcal{G}_d$; waits for (sold) from $\mathcal{G}_d$ to halt.
- The consumer $\mathcal{C}$ would first assert $\Sigma \equiv \mathsf{revealing}$ and wait for $(\mathsf{revealed}, erk)$ from $\mathcal{G}_d$ to execute:
  - Runs Alg. 4, namely $\mathsf{ValidateRKeys}(n, \zeta, \mathsf{ctr}, erk)$ to preliminarily check whether the revealed elements erk can recover the correct number (i.e, ctr) of keys. If $false$ is returned, sends $(\mathsf{wrongRK})$ to $\mathcal{G}_d$ and halts;
  - If $\mathsf{ValidateRKeys}(n, \zeta, \mathsf{ctr}, erk) \equiv true$, decrypts erk to obtain $rk \leftarrow \mathsf{VDec}_{vsk_{\mathcal{C}}}(erk)$, and then runs $ks = (k_\zeta, \cdots, k_{\zeta+\mathsf{ctr}-1}) \leftarrow \mathsf{SKD.RecoverKeys}(n, \mathsf{ctr}, rk)$ to recover the chunk keys. Then $\mathcal{C}$ uses these keys to decrypt $(c_\zeta, \cdots, c_{\zeta+\mathsf{ctr}-1})$ to obtain $m'_i = \mathsf{SDec}_{k_i}(c_i)$, $i \in [\zeta, (\zeta + \mathsf{ctr} - 1)]$, and checks whether for every $m'_i \in (m'_\zeta, \cdots, m'_{\zeta+\mathsf{ctr}-1})$, $\mathcal{H}(m'_i)$ equates the i-th leaf node, i.e., $\mathcal{H}(m_i)$, in Merkle tree MT received from $\mathcal{P}$ in the *Deliver* phase. If all are consistent, it means that $\mathcal{C}$ receives all the desired chunks and there is no dispute, $\mathcal{C}$ outputs $(m'_\zeta, \cdots, m'_{\zeta+\mathsf{ctr}-1})$, and then waits for (sold) from $\mathcal{G}_d$ to halt. Otherwise, $\mathcal{C}$ can raise complaint by: choosing one inconsistent position (e.g., the i-th chunk), and computes $(rk, \pi_{\mathsf{VD}}) \leftarrow \mathsf{ProvePKE}_{vsk_{\mathcal{C}}}(erk)$ and $\pi_{\mathsf{M}}^i \leftarrow \mathsf{GenMTP}(\mathsf{MT}, \mathcal{H}(m_i))$, and then sends $(\mathsf{PoM}, i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi_{\mathsf{MT}}^i, rk, erk, \pi_{\mathsf{VD}})$ to the contract $\mathcal{G}_d$, where i is the index of the incorrect chunk to be proved; j is the index of the element in erk that can induce the key k_i for the position i; c_i and σ_{c_i} are the i-th encrypted chunk and its signature received in the *Deliver* phase; $\mathcal{H}(m_i)$ is the value of the i-th leaf node in MT; π_{MT}^i is the Merkle proof for $\mathcal{H}(m_i)$; rk is decryption result from erk; erk is the encrypted revealed key; π_{VD} is the verifiable decryption proof attesting to the correctness of decrypting erk to rk.

Dispute resolution. For the sake of completeness, the details of ValidatePoM subroutine is presented in Alg. 5, which allows the consumer to prove that it decrypts a chunk inconsistent to the digest root_m. The time complexity is $O(\log n)$, which is critical to achieve the efficiency requirement. Additionally, we consider a natural (optimistic) case where an honest consumer $\mathcal{C}$ would not complain to the contract if receiving valid content.

Repeatable delivery. Π_{FD} can support repeatable delivery for at most θ times, where θ is a pre-specified parameter. Here θ is a finite number used to determine the minimum deposits of the provider placed in the contract. Once a delivery session completes (i.e., the contract state Σ either becomes sold or not_sold), the reset function will be invoked to start a new delivery session that can serve another consumer as long as not repeat for more than θ times. After θ-time deliveries, $\mathcal{P}$ can utilize the same contract to re-deposit sufficient collaterals for another θ repeatable deliveries.

Monetary incentivization. Monetary collateral is a widely adopted way to ensure fairness using blockchain [23]. Other alternative collaterals like reputation or credits require unique digital identities [39] in the permissionless setting, which is still under active exploration. Note that the potential overhead of utilizing deposits lies in their time value, e.g., the provider may face an opportunity cost in the form of forgone returns that they could have accrued in alternative investments. However, such overhead would not disincentivize the content providers as long as they can expect to receive more payments by selling their contents.

5.4 Analyzing FairDownload Protocol

Theorem 1 characterizes the properties of the protocol Π_{FD}.

Algorithm 5 ValidatePoM algorithm

Input: $(i, j, c_i, \sigma_{c_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}}, rk, erk, \pi_{\mathsf{VD}})$
$(\mathsf{root}_m, n, erk_{\mathsf{hash}}, pk_{\mathcal{P}}, vpk_{\mathcal{C}})$ are stored in the contract and so accessible
Output: $true$ or $false$
1: assert $j \in [0, |erk| - 1]$
2: assert $\mathcal{H}(erk) \equiv erk_{\mathsf{hash}}$
3: assert $\mathsf{VerifyPKE}_{vpk_{\mathcal{C}}}(erk, rk, \pi_{\mathsf{VD}}) \equiv 1$
4: assert $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$
5: assert $\mathsf{VerifyMTP}(\mathsf{root}_m, i, \pi^i_{\mathsf{MT}}, \mathcal{H}(m_i)) \equiv 1$
6: $k_i = \mathsf{RecoverChunkKey}(i, j, n, rk)$
7: assert $k_i \neq \perp$
8: $m'_i = \mathsf{SDec}(c_i, k_i)$
9: assert $\mathcal{H}(m'_i) \neq \mathcal{H}(m_i)$
10: **return** $false$ in case of any assertion error or $true$ otherwise

Algorithm 6 RecoverChunkKey algorithm

Input: (i, j, n, rk)
Output: k_i or $\perp$
1: $(x, y) \leftarrow rk[j]$
▷ parse the j-th element in rk to get the key x and the value y
2: let k_path be an empty stack
3: $ind = n + i - 2$ ▷ index in KT
4: **if** $ind < x$ **then**
5: **return** $\perp$
6: **if** $ind \equiv x$ **then**
7: **return** y ▷ $k_i = y$
8: **while** $ind > x$ **do**
9: k_path pushes 0 if ind is odd
10: k_path pushes 1 if ind is even
11: $ind = \lfloor (ind - 1)/2 \rfloor$
12: let $b = |k\_path|$
13: **while** $(b\text{--}) > 0$ **do**
14: pop k_path to get the value t
15: $k_i = \mathcal{H}(y||t)$
16: **return** k_i

Theorem 1. *Conditioned on that the underlying cryptographic primitives are secure, the protocol* FairDownload *satisfies the completeness, fairness, confidentiality against deliverer, timeliness and non-trivial efficiency properties in the synchronous authenticated network,* $\mathcal{G}_d^{\mathsf{ledger}}$*-hybrid and stand-alone model.*

Proof. The proof for theorem 1 is analyzed as follows:

Completeness. The completeness of Π_{FD} is immediate to see: when all three participating parties honestly follow the protocol, the provider $\mathcal{P}$ gets a net income of $n \cdot (\beth_{\mathcal{C}} - \beth_{\mathcal{P}})$; the deliverer $\mathcal{D}$ obtains the well-deserved payment of $n \cdot \beth_{\mathcal{P}}$; the consumer $\mathcal{C}$ receives the valid content m, i.e., $\phi(m) \equiv 1$.

Fairness. The fairness for each party in Π_{FD} can be reduced to the underlying cryptographic building blocks.

- *Consumer fairness*. Consumer fairness means that the honest $\mathcal{C}$ only needs to pay proportional to what it *de facto* obtains even though malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ may collude with each other. This case can be modeled as an adversary $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{D}$ to provide and deliver the content to the honest $\mathcal{C}$. In the *Deliver* phase, the VFD subroutine ensures that if $\mathcal{C}$ receives $\ell \in [n]$ encrypted chunks and $\mathcal{A}$ maliciously aborts, $\mathcal{A}$ can only claim payment from the contract of $\ell \cdot \beth_{\mathcal{P}}$, which is paid by the $\mathcal{A}$ itself due to the collusion. In the *Reveal* phase, if $\mathcal{A}$ reveals correct elements in KT to recover the ℓ decryption keys, then $\mathcal{C}$ can decrypt to obtain the valid ℓ chunks. Otherwise, $\mathcal{C}$ can raise complaint by sending the (wrongRK) and further (PoM) to the contract and gets refund. It is obvious to see that $\mathcal{C}$ either pays for the ℓ valid chunks or pays nothing. The fairness for the consumer is guaranteed unless $\mathcal{A}$ can: (i) break VFD to forge $\mathcal{C}$'s signature; (ii) find Merkle tree collision, namely find another chunk $m'_i \neq m_i$ in position i of m to bind to the same root_m so that $\mathcal{A}$ can fool the contract to reject $\mathcal{C}$'s complaint (by returning $false$ of ValidatePoM) while indeed sent wrong chunks; (iii) manipulate the execution of smart contract in blockchain. However, according to the security guarantee of the underlying signature scheme, the second-preimage resistance of hash function in Merkle tree, and that the smart contract is modeled as an ideal functionality, the probability to break $\mathcal{C}$'s fairness is negligible. Hence, the consumer fairness is strictly guaranteed.
- *Deliverer fairness*. Deliverer fairness states that the honest $\mathcal{D}$ receives the payment proportional to the expended bandwidth though the malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ may collude with each other. This amounts to the case that $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{C}$ and try to reap $\mathcal{D}$'s bandwidth contribution without paying. In the VFD subroutine, considering $\mathcal{D}$ delivers ℓ chunks, then it can correspondingly obtain either ℓ ($\ell \in [n]$) or $\ell - 1$ (i.e., $\mathcal{A}$ stops sending the ℓ-th receipt) receipts acknowledging the bandwidth contribution. Later $\mathcal{D}$ can use the latest receipt containing $\mathcal{C}$'s signature to claim payment $\ell \cdot \beth_{\mathcal{P}}$ or $(\ell - 1) \cdot \beth_{\mathcal{P}}$ from the contract. At most $\mathcal{D}$ may waste bandwidth for delivering one chunk-validation pair of $O(\eta)$ bits. To break the security, $\mathcal{A}$ has to violate the contract functionality (i.e., control the execution of smart contract), which is of negligible probability. Therefore, the deliverer fairness is strictly ensured.
- *Provider fairness*. Provider fairness indicates that the honest $\mathcal{P}$ receives the payment proportional to the number of valid content chunks that $\mathcal{C}$ learns. The malicious $\mathcal{D}^*$ can collude with the malicious $\mathcal{C}^*$ or simply create multiple fake $\mathcal{C}^*$ (i.e., Sybil attack), and then cheat $\mathcal{P}$ without real delivery. These cases can be modeled as an adversary $\mathcal{A}$ corrupts both $\mathcal{D}$ and $\mathcal{C}$. $\mathcal{A}$ can break the fairness for the honest $\mathcal{P}$ from two aspects by: (i) letting $\mathcal{P}$ pay for the delivery without truly delivering any content; (ii) obtaining the content without paying for $\mathcal{P}$. For case (i), $\mathcal{A}$ can claim that ℓ ($\ell \in [n]$) chunks have been delivered and would receive the payment $\ell \cdot \beth_{\mathcal{P}}$ from the contract. Yet this procedure would also update $\mathsf{ctr} := \ell$ in the contract, which later allows $\mathcal{P}$ to retrieve the payment $\ell \cdot \beth_{\mathcal{C}}$ after $\mathcal{T}_{\mathsf{dispute}}$ expires unless $\mathcal{A}$ can manipulate the execution of smart contract, which is of negligible probability. Hence, $\mathcal{P}$ can still obtain the well-deserved payment $\ell \cdot (\beth_{\mathcal{C}} - \beth_{\mathcal{P}})$. For case (ii), $\mathcal{A}$ can either try to decrypt the delivered chunks by itself without utilizing the revealing keys from $\mathcal{P}$, or try to fool the contract to accept the PoM and therefore repudiate the payment for $\mathcal{P}$ though $\mathcal{P}$ honestly reveals chunk keys. The former situation can be reduced to the violation of semantic security of the underlying encryption scheme and the pre-image resistance of cryptographic hash functions, and the latter requires $\mathcal{A}$ to forge $\mathcal{P}$'s signature, or break the soundness of the verifiable decryption scheme, or control the execution of the smart contract. Obviously, the occurrence of aforementioned situations are of negligible probability. Overall, the provider fairness is strictly assured.

In sum, Π_{FD} strictly guarantees the fairness for $\mathcal{P}$ and $\mathcal{C}$, and the unpaid delivery for $\mathcal{D}$ is bounded to $O(\eta)$ bits. The fairness requirement of Π_{FD} is satisfied.

Confidentiality. This property states that on input all protocol scripts and the corrupted $\mathcal{D}^*$'s private input and all internal states, it is still computationally infeasible for the adversary $\mathcal{A}$ to output the provider's raw content in a

protocol instance. In Π_{FD}, each chunk delegated to $\mathcal{D}$ is encrypted via symmetric encryption scheme using the keys that derived from SKD.GenSubKeys(). Furthermore, the revealed on-chain elements erk for recovering decryption keys are encrypted utilizing the consumer $\mathcal{C}$'s pubic key. Additionally, $\mathcal{C}$ receives the Merkle tree MT of the content m before the verifiable fair delivery (VFD) procedure starts. To break the confidentiality, $\mathcal{A}$ has to violate any of the following conditions: (i) the pre-image resistance of cryptographic hash function in the Merkle tree; and (ii) the security of the public key encryption scheme. The probability of violating the aforementioned security properties is negligible and therefore Π_{FD} satisfies the confidentiality property.

Timeliness. Timeliness states that the honest parties in the protocol Π_{FD} terminates in $O(n)$ synchronous rounds, where n is the number of content chunks, and when the protocol completes or aborts, the fairness and confidentiality are always preserved. The guarantee of confidentiality can be straightforwardly derived from prior analysis even if malicious parties abort, we only focus on the assurance of fairness. Now we elaborate the following termination cases for the protocol Π_{FD} with the arbiter contract $\mathcal{G}_d$ and at least one honest party:

No abort. If all of $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ are honest, Π_{FD} terminates in the *Reveal* phase, after $\mathcal{T}_{\mathsf{dispute}}$ expires. The *Prepare* phase and the *Reveal* phase need $O(1)$ synchronous rounds, and the *Deliver* phase requires $O(n)$ rounds, yielding total $O(n)$ rounds for Π_{FD} to terminate and the fairness is guaranteed at completion since each party obtains the deserved items.

Aborts in the Prepare phase. This phase involves the interaction between $\mathcal{P}$, $\mathcal{D}$ and the contract $\mathcal{G}_d$. It is obvious this phase can terminate in $O(1)$ rounds if any party maliciously aborts or the honest party does not receive response after $\mathcal{T}_{\mathsf{round}}$ expires. Besides, after each step in this phase, the fairness for both $\mathcal{P}$ and $\mathcal{D}$ is preserved no matter which one of them aborts, i.e., $\mathcal{P}$ does not lose any coins in the ledger or leak any content chunks, while $\mathcal{D}$ does not waste any bandwidth resource.

Aborts in the Deliver phase. This phase involves $\mathcal{P}$, $\mathcal{D}$, $\mathcal{C}$ and the contract $\mathcal{G}_d$. It can terminate in $O(n)$ rounds. If other parties abort after $\mathcal{C}$ sends (consume) message to $\mathcal{G}_d$, $\mathcal{C}$ would get its deposit back upon $\mathcal{T}_{\mathsf{round}}$ expires. The VFD procedure in this phase only involves $\mathcal{D}$ and $\mathcal{C}$, and the fairness is guaranteed whenever one of the two parties aborts, as early analyzed. The timer $\mathcal{T}_{\mathsf{deliver}}$ in $\mathcal{G}_d$ indicates that the delivery should be completed within such a time period, or else $\mathcal{G}_d$ would continue with the protocol by informing $\mathcal{D}$ to claim payment and update ctr after $\mathcal{T}_{\mathsf{deliver}}$ times out. $\mathcal{D}$ is naturally self-motivated not to maliciously abort until receiving the payment from $\mathcal{G}_d$. At the end of this phase, $\mathcal{D}$ completes its task in the delivery session, while for $\mathcal{P}$ and $\mathcal{C}$, they are motivated to enter the next phase and the fairness for them at this point is guaranteed, i.e., $\mathcal{P}$ decreases coins by $\mathsf{ctr} \cdot \$_{\mathcal{P}}$ in ledger, but the contract has also updated ctr, which allows $\mathcal{P}$ to receive $\mathsf{ctr} \cdot \$_{\mathcal{C}}$ from the ledger if keys are revealed honestly, and $\mathcal{C}$ obtains the encrypted chunks while does not lose any coins in ledger.

Aborts in the Reveal phase. This phase involves $\mathcal{P}$, $\mathcal{C}$ and the contract $\mathcal{G}_d$. It can terminate in $O(1)$ rounds after $\mathcal{G}_d$ sets the state as sold or not_sold. If $\mathcal{C}$ aborts after $\mathcal{P}$ reveals the chunk keys on-chain, $\mathcal{P}$ can wait until $\mathcal{T}_{\mathsf{dispute}}$ times out and attain the deserved payment $\mathsf{ctr} \cdot \$_{\mathcal{C}}$. If $\mathcal{P}$ reveals incorrect keys and then aborts, $\mathcal{C}$ can raise complaint within $\mathcal{T}_{\mathsf{dispute}}$ by sending message (wrongRK) and further (PoM) to get refund. Hence, the fairness for either $\mathcal{P}$ and $\mathcal{C}$ is guaranteed no matter when and who aborts maliciously in this phase.

Non-trivial efficiency. The analysis of ensuring this property can be conducted in the following three aspects:

- *Communication complexity*. In the *Prepare* phase, $\mathcal{P}$ delegates the signed encrypted chunks to $\mathcal{D}$, where the communication complexity is $O(n)$. Typically this phase only needs to be executed once for the same content. In the *Deliver* phase, $\mathcal{P}$ sends the content Merkle tree MT to $\mathcal{C}$, and $\mathcal{D}$ activates the VFD subroutine to deliver the chunks to $\mathcal{C}$. The communication complexity in this phase is also $O(n)$. In the *Reveal* phase, the revealed elements for recovering ctr keys is *at most* $O(\log n)$. Finally, if dispute happens, the communication complexity of sending PoM (mostly due to the merkle proof π^i_{MT}) to the contract is $O(\log n)$. Therefore, the communication complexity of the protocol Π_{FD} is $O(n)$.
- *On-chain costs*. In the *optimistic* case where no dispute occurs, the on-chain costs of Π_{FD} include: (i) the functions (i.e., start, join and prepared) in the *Prepare* phase are all $O(1)$; (ii) in the *Deliver* phase, the consume and delivered functions are $O(1)$. Note in the delivered function, the cost of signature verification is $O(1)$ as $\mathcal{D}$ only needs to submit the latest receipt containing one signature of $\mathcal{C}$; (iii) the storage cost for revealed elements (i.e., erk) is *at most* $O(\log n)$, where n is the number of chunks. Overall, the on-chain cost is *at most* $O(\log n)$, namely $\widetilde{O}(1)$. In the *pessimistic* case where dispute happens, the on-chain cost is only related to the chunk size η no matter how large the content is.
- *Message volume for $\mathcal{P}$*. Considering that the contract is deployed and the deliverer is ready to deliver. Every time when a new consumer joins in, a new delivery session starts. The provider $\mathcal{P}$ shows up twice for: (i) sending the Merkle tree MT, which can be represented by $n \cdot \lambda$ bits where λ is a small cryptographic parameter, to $\mathcal{C}$ in the *Deliver* phase, and (ii) revealing erk, which is *at most* $\log n \cdot \lambda$ bits, to $\mathcal{C}$ in the *Reveal* phase. The total message volume can be represented as $n \cdot \lambda$ bits, which is much smaller than the content size $|m| = n \cdot \eta$ where η is the chunk size and considering the fact that $\lambda \ll \eta$.

6 FairStream: Fair P2P Streaming

In this section, we present the fair P2P streaming protocol Π_{FS}, allowing consumers to *view-while-delivery*.

6.1 FairStream Overview

As depicted in Fig. 4, Π_{FS} works as three phases, i.e., *Prepare*, *Stream*, and *Payout*. The key ideas for Π_{FS} are:

- Same as the *Prepare* phase in Π_{FD}, the provider $\mathcal{P}$ deploys the smart contract, encrypts the chunks, signs and delegates the encrypted chunks to the deliver $\mathcal{D}$.
- The streaming process consists of $O(n)$ communication rounds, where n is the number of chunks. In each round, the consumer $\mathcal{C}$ would receive an encrypted

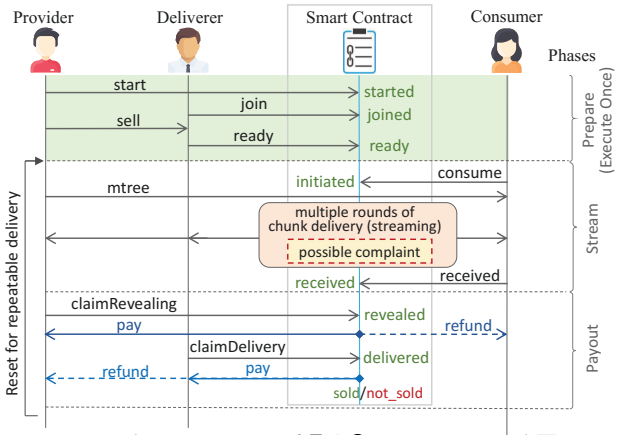


Fig. 4: The overview of FairStream protocol Π_{FS}.

chunk from $\mathcal{D}$ and a decryption key from $\mathcal{P}$; a party may abort in a certain round due to, e.g., a timely response or invalid message; especially, in case erroneous chunk is detected during streaming, $\mathcal{C}$ can complain and get compensated with a valid and short, i.e., $O(\eta + \lambda)$-bit proof;

- Eventually all parties enter the *Payout* phase, where $\mathcal{D}$ and $\mathcal{P}$ can claim the deserved payment by submitting the latest receipt signed by $\mathcal{C}$ before a timer maintained in contract expires; the contract determines the final internal state ctr, namely the number of delivered chunks or revealed keys, as the *larger* one of the indexes in $\mathcal{P}$ and $\mathcal{D}$'s receipts. If no receipt is received from $\mathcal{P}$ or $\mathcal{D}$ before the timer expires, the contract would treat the submitted index for that party as 0. Such a design is critical to ensure fairness as analyzed in § 6.4.

Fig. 5 illustrates the concrete message flow of one round chunk delivery in the *Stream* phase. We highlight that a black-box call of the VFD module is not applicable to the streaming setting as VFD only allows the consumer $\mathcal{C}$ to obtain the encrypted chunks, so that the provider $\mathcal{P}$ merely needs to show up once to reveal a minimum number of elements and get all chunk keys recovered. However, the streaming procedure demands much less latency of retrieving each content chunk, leading to the intuitive design to let $\mathcal{C}$ receive both an encrypted chunk and a corresponding chunk decryption key in one same round. $\mathcal{P}$ is therefore expected to keep online and reveal each chunk key to $\mathcal{C}$. Overall, Π_{FS} requires relatively more involvement of $\mathcal{P}$ compared with Π_{FD}, but the advantage is that instead of downloading all chunks in $O(n)$ rounds before viewing, $\mathcal{C}$ now can retrieve each chunk with $O(1)$ latency. All other properties including each party's fairness, the on-chain computational cost, and the deliverer's communication complexity remain the same as those in the downloading setting.

6.2 Arbiter Contract $\mathcal{G}_s^{\mathsf{ledger}}$ for Streaming

The arbiter contract $\mathcal{G}_s^{\mathsf{ledger}}$ (abbr. $\mathcal{G}_s$) illustrated in Fig. 6 is a stateful ideal functionality that can access to ledger functionality to facilitate the fair content delivery via streaming. The timer $\mathcal{T}_{\mathsf{receive}}$ ensures that when any party maliciously aborts or the consumer $\mathcal{C}$ receives invalid chunk during the streaming process, Π_{FS} can smoothly continue and enter the next phase. The dispute resolution in contract is relatively

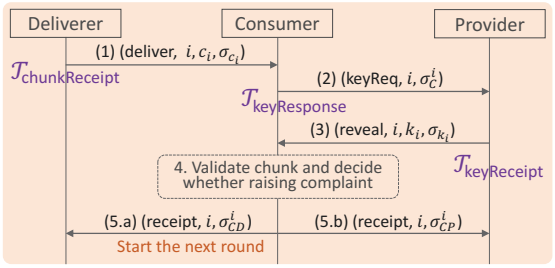


Fig. 5: The message flow of one round chunk delivery in the *Stream* phase of Π_{FS}. All these messages are sent off-chain.

simpler than the downloading setting since no verifiable decryption is needed. The timer $\mathcal{T}_{\mathsf{finish}}$ indicates that both $\mathcal{D}$ and $\mathcal{P}$ are supposed to send the request of claiming their payment before $\mathcal{T}_{\mathsf{finish}}$ times out, and therefore it is natural to set $\mathcal{T}_{\mathsf{finish}} > \mathcal{T}_{\mathsf{receive}}$. Once $\mathcal{T}_{\mathsf{finish}}$ expires, the contract determines the final ctr by choosing the maximum index in $\mathcal{P}$ and $\mathcal{D}$'s receipts, namely $\mathsf{ctr}_{\mathcal{P}}$ and $\mathsf{ctr}_{\mathcal{D}}$, respectively, and then distributes the deserved payment for each party. Once the delivery session completes, $\mathcal{P}$ can invoke the contract by sending (reset) to $\mathcal{G}_s$ to reset to the ready state and continue to receive new requests from consumers.

6.3 Π_{FS}: FairStream Protocol for P2P Streaming

We now present the concrete message flow in Π_{FS}.

Phase I for Prepare. This phase executes the same as the *Prepare* phase in the Π_{FD} protocol.

Phase II for Stream. The consumer $\mathcal{C}$, the deliverer $\mathcal{D}$ and the provider $\mathcal{P}$ interact with the contract $\mathcal{G}_s$ in this phase as:

- The consumer $\mathcal{C}$ who is interested in the content with digest root_m and wants to start the streaming from the ζ-th ($\zeta \in [n]$) chunk, would operate as follows:
  - Asserts $\Sigma \equiv \mathsf{ready}$, runs $(pk_{\mathcal{C}}, sk_{\mathcal{C}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$, and sends $(\mathsf{consume}, pk_{\mathcal{C}}, \zeta)$ to $\mathcal{G}_s$;
  - Upon receiving the message $(\mathsf{mtree}, \mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}})$ from $\mathcal{P}$, asserts $\mathsf{Verify}(\mathsf{MT}, \sigma_{\mathcal{P}}^{\mathsf{MT}}, pk_{\mathcal{P}}) \equiv 1 \wedge \mathsf{root}(\mathsf{MT}) \equiv \mathsf{root}_m$, and stores the Merkle tree MT, or else halts;
  - Upon receiving the message $(\mathsf{deliver}, i, c_i, \sigma_{c_i})$ from $\mathcal{D}$, checks whether $i \equiv \zeta \wedge \mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$, if hold, starts (for the first delivery) a timer $\mathcal{T}_{\mathsf{keyResponse}}$ or resets (for not the first deliveries) it, sends $(\mathsf{keyReq}, i, \sigma_{\mathcal{C}}^i)$ where $\sigma_{\mathcal{C}}^i \leftarrow \mathsf{Sign}(i||pk_{\mathcal{C}}, sk_{\mathcal{C}})$ to $\mathcal{P}$ (i.e., the step (2) in Fig. 5). If failing to check or $\mathcal{T}_{\mathsf{keyResponse}}$ times out, halts;
  - Upon receiving the message $(\mathsf{reveal}, i, k_i, \sigma_{k_i})$ from $\mathcal{P}$ before $\mathcal{T}_{\mathsf{keyResponse}}$ times out, checks whether $i \equiv \zeta \wedge \mathsf{Verify}(i||k_i, \sigma_{k_i}, pk_{\mathcal{P}}) \equiv 1$, if failed, halts. Otherwise, starts to validate the content chunk based on received c_i and k_i: decrypts c_i to obtain m_i', where $m_i' = \mathsf{SDec}_{k_i}(c_i)$, and then checks whether $\mathcal{H}(m_i')$ is consistent with the i-th leaf node in the Merkle tree MT, if inconsistent, sends $(\mathsf{PoM}, i, c_i, \sigma_{c_i}, k_i, \sigma_{k_i}, \mathcal{H}(m_i), \pi_{\mathsf{MT}}^i)$ to $\mathcal{G}_s$. If it is consistent, sends the receipts $(\mathsf{receipt}, i, \sigma_{\mathcal{CD}}^i)$ to $\mathcal{D}$ and $(\mathsf{receipt}, i, \sigma_{\mathcal{CP}}^i)$ to $\mathcal{P}$, where $\sigma_{\mathcal{CD}}^i \leftarrow \mathsf{Sign}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, sk_{\mathcal{C}})$ and $\sigma_{\mathcal{CP}}^i \leftarrow \mathsf{Sign}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, sk_{\mathcal{C}})$, and sets $\zeta := \zeta + 1$, and then waits for the next (deliver) message from $\mathcal{D}$. Upon ζ is set to be $n + 1$, sends (received) to $\mathcal{G}_s$;

The Arbiter Contract Functionality $\mathcal{G}_s^{\mathsf{ledger}}$ for P2P Streaming

The contract $\mathcal{G}_s$ can access to ledger and interacts with $\mathcal{P}$, $\mathcal{D}$, $\mathcal{C}$ and the adversary $\mathcal{A}$. It locally maintains the following variables θ, n, ζ, root_m, $\mathbb{B}_{\mathcal{P}}$, $\mathbb{B}_{\mathcal{C}}$, $\mathbb{B}_{\mathsf{pf}}$, $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, ctr (all $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, ctr are initially 0), $pk_{\mathcal{P}}, pk_{\mathcal{D}}, pk_{\mathcal{C}}$, the penalty flag plt (initially $false$), the state Σ, and sets up three timers $\mathcal{T}_{\mathsf{round}}$ (implicit), $\mathcal{T}_{\mathsf{receive}}$, $\mathcal{T}_{\mathsf{finish}}$.

Phase 1: Prepare

- This phase is the same as the *Prepare* phase in $\mathcal{G}_d$.

Phase 2: Stream

- On receive $(\mathsf{consume}, pk_{\mathcal{C}}, \zeta)$ from $\mathcal{C}$:
  - assert $\theta > 0$, $\zeta \in [n]$, and store ζ
  - assert $\mathsf{ledger}[\mathcal{C}] \geq (n-\zeta+1)\cdot \mathbb{B}_{\mathcal{C}} \wedge \Sigma \equiv \mathsf{ready}$
  - store $pk_{\mathcal{C}}$ and let $\mathsf{ledger}[\mathcal{C}] = \mathsf{ledger}[\mathcal{C}] - (n-\zeta+1)\cdot \mathbb{B}_{\mathcal{C}}$
  - start two timers $\mathcal{T}_{\mathsf{receive}}$, and $\mathcal{T}_{\mathsf{finish}}$
  - let $\Sigma :=$ initiated and send $(\mathsf{initiated}, pk_{\mathcal{C}}, \zeta)$ to all entities
- On receive (received) from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{receive}}$ times out:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv$ initiated
  - let $\Sigma :=$ received and send (received) to all entities
- Upon $\mathcal{T}_{\mathsf{receive}}$ times out:
  - assert current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv$ initiated
  - let $\Sigma :=$ received and send (received) to all entities

▷ Below is to resolve dispute during streaming in Π_{FS}

- On receive $(\mathsf{PoM}, i, c_i, \sigma_{c_i}, k_i, \sigma_{k_i}, \mathcal{H}(m_i), \pi^i_{\mathsf{MT}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{receive}}$ expires:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{receive}}$ and $\Sigma \equiv$ initiated
  - assert $i \in [\zeta, n]$
  - assert $\mathsf{Verify}(i||c_i, \sigma_{c_i}, pk_{\mathcal{P}}) \equiv 1$
  - assert $\mathsf{Verify}(i||k_i, \sigma_{k_i}, pk_{\mathcal{P}}) \equiv 1$
  - assert $\mathsf{VerifyMTP}(\mathsf{root}_m, i, \pi^i_{\mathsf{MT}}, \mathcal{H}(m_i)) \equiv 1$
  - $m'_i = \mathsf{SDec}(c_i, k_i)$
  - assert $\mathcal{H}(m'_i) \neq \mathcal{H}(m_i)$
  - let plt $:= true$
  - let $\Sigma :=$ received and send (received) to all entities

Phase 3: Payout

- On receive $(\mathsf{claimDelivery}, i, \sigma^i_{\mathcal{CD}})$ from $\mathcal{D}$:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{finish}}$
  - assert $\Sigma \equiv$ received or $\Sigma \equiv$ payingRevealing
  - assert ctr $\equiv 0$ and $i \in [\zeta, n]$
  - assert $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, \sigma^i_{\mathcal{CD}}, pk_{\mathcal{C}}) \equiv 1$
  - let $\mathsf{ctr}_{\mathcal{D}} := i-\zeta+1$, $\Sigma :=$ payingDelivery, and then send (payingDelivery) to all entities
- On receive $(\mathsf{claimRevealing}, i, \sigma^i_{\mathcal{CP}})$ from $\mathcal{P}$:
  - assert current time $\mathcal{T} < \mathcal{T}_{\mathsf{finish}}$
  - assert $\Sigma \equiv$ received or $\Sigma \equiv$ payingDelivery
  - assert ctr $\equiv 0$ and $i \in [\zeta, n]$
  - assert $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, \sigma^i_{\mathcal{CP}}, pk_{\mathcal{C}}) \equiv 1$
  - let $\mathsf{ctr}_{\mathcal{P}} := i-\zeta+1$, $\Sigma :=$ payingRevealing, and then send (payingRevealing) to all entities
- Upon $\mathcal{T}_{\mathsf{finish}}$ times out:
  - assert current time $\mathcal{T} \geq \mathcal{T}_{\mathsf{finish}}$
  - let ctr $:= \max\{\mathsf{ctr}_{\mathcal{D}}, \mathsf{ctr}_{\mathcal{P}}\}$
  - let $\mathsf{ledger}[\mathcal{D}] := \mathsf{ledger}[\mathcal{D}] + \mathsf{ctr}\cdot \mathbb{B}_{\mathcal{P}}$
  - if plt:
    - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n-\mathsf{ctr})\cdot \mathbb{B}_{\mathcal{P}} + \mathsf{ctr}\cdot \mathbb{B}_{\mathcal{C}}$
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n-\zeta+1-\mathsf{ctr})\cdot \mathbb{B}_{\mathcal{C}} + \mathbb{B}_{\mathsf{pf}}$
  - else:
    - let $\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] + (n-\mathsf{ctr})\cdot \mathbb{B}_{\mathcal{P}} + \mathsf{ctr}\cdot \mathbb{B}_{\mathcal{C}} + \mathbb{B}_{\mathsf{pf}}$
    - let $\mathsf{ledger}[\mathcal{C}] := \mathsf{ledger}[\mathcal{C}] + (n-\zeta+1-\mathsf{ctr})\cdot \mathbb{B}_{\mathcal{C}}$
  - if ctr > 0: let $\Sigma :=$ sold and send (sold) to all entities
  - else let $\Sigma :=$ not_sold and send (not_sold) to all entities
  - reset( )

▷ Reset to the ready state for repeatable delivery

- function reset( ):
  - assert $\Sigma \equiv$ sold or $\Sigma \equiv$ not_sold
  - set ctr, $\mathsf{ctr}_{\mathcal{D}}$, $\mathsf{ctr}_{\mathcal{P}}$, $\mathcal{T}_{\mathsf{receive}}$, $\mathcal{T}_{\mathsf{finish}}$, ζ as 0
  - nullify $pk_{\mathcal{C}}$
  - let $\theta := \theta - 1$ and $\Sigma :=$ ready
  - send (ready) to all entities

Fig. 6: The arbiter contract functionality $\mathcal{G}_s^{\mathsf{ledger}}$. "Sending to all entities" captures smart contract is transparent to the public.

  - Waits for the messages (received) from $\mathcal{G}_s$ to halt.
- The deliverer $\mathcal{D}$ initializes a variable $x := 1$ and executes as follows in this phase:
  - Upon receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, \zeta)$ from $\mathcal{G}_s$, sets $x = \zeta$, sends the message $(\mathsf{deliver}, i, c_i, \sigma_{c_i}), i = x$ to $\mathcal{C}$ and starts a timer $\mathcal{T}_{\mathsf{chunkReceipt}}$;
  - Upon receiving the message $(\mathsf{receipt}, i, \sigma^i_{\mathcal{CD}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{chunkReceipt}}$ times out, checks whether $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{D}}, \sigma^i_{\mathcal{CD}}, pk_{\mathcal{C}}) \equiv 1 \wedge i \equiv x$ or not, if succeed, continues with the next iteration: sets $x := x+1$, sends $(\mathsf{deliver}, i, c_i, \sigma_{c_i}), i = x$ to $\mathcal{C}$, and resets $\mathcal{T}_{\mathsf{chunkReceipt}}$ (i.e., the step (1) in Fig. 5); otherwise $\mathcal{T}_{\mathsf{chunkReceipt}}$ times out, enters the next phase.
- The provider $\mathcal{P}$ initializes a variable $y := 1$ and executes as follows in this phase:
  - On receiving $(\mathsf{initiated}, pk_{\mathcal{C}}, \zeta)$ from $\mathcal{G}_s$: asserts $\Sigma \equiv$ initiated, lets $y = \zeta$ and sends $(\mathsf{mtree}, \mathsf{MT}, \sigma^{\mathsf{MT}}_{\mathcal{P}})$ to $\mathcal{C}$;
  - Upon receiving $(\mathsf{keyReq}, i, \sigma^i_{\mathcal{C}})$ from $\mathcal{C}$, checks whether $i \equiv y \wedge \mathsf{Verify}(i||pk_{\mathcal{C}}, \sigma^i_{\mathcal{C}}, pk_{\mathcal{C}}) \equiv 1$, if succeed, sends $(\mathsf{reveal}, i, k_i, \sigma_{k_i})$ where $\sigma_{k_i} \leftarrow \mathsf{Sign}(i||k_i, sk_{\mathcal{P}})$, to $\mathcal{C}$ and starts (for the first delivery) a timer $\mathcal{T}_{\mathsf{keyReceipt}}$ or resets (for not the first deliveries) it (i.e., the step (3) in Fig. 5), otherwise enters the next phase;
  - On input $(\mathsf{receipt}, i, \sigma^i_{\mathcal{CP}})$ from $\mathcal{C}$ before $\mathcal{T}_{\mathsf{keyReceipt}}$ expires, checks $\mathsf{Verify}(\mathsf{receipt}||i||pk_{\mathcal{C}}||pk_{\mathcal{P}}, \sigma^i_{\mathcal{CP}}, pk_{\mathcal{C}}) \equiv 1 \wedge i \equiv y$ or not, if succeed, sets $y := y+1$. Otherwise if $\mathcal{T}_{\mathsf{keyReceipt}}$ times out, enters the next phase.

Phase III for Payout. The provider $\mathcal{P}$ and the deliverer $\mathcal{D}$ interact with the contract $\mathcal{G}_s$ in this phase as:

- The provider $\mathcal{P}$ executes as follows in this phase:
  - Upon receiving (received) or (delivered) from $\mathcal{G}_s$, or receiving the n-th receipt from $\mathcal{C}$ (i.e., y is set to be $n+1$), sends $(\mathsf{claimRevealing}, i, \sigma^i_{\mathcal{CP}})$ to $\mathcal{G}_s$;
  - Waits for (revealed) from $\mathcal{G}_s$ to halt.
- The deliverer $\mathcal{D}$ executes as follows during this phase:
  - Upon receiving (received) or (revealed) from $\mathcal{G}_s$, or receiving the n-th receipt from $\mathcal{C}$ (i.e., x is set to be $n+1$), sends $(\mathsf{claimDelivery}, i, \sigma^i_{\mathcal{CD}})$ to $\mathcal{G}_s$;
  - Waits for (delivered) from $\mathcal{G}_s$ to halt.

6.4 Analyzing FairStream Protocol

Theorem 2 characterizes the properties of the protocol Π_{FS}.

Theorem 2. *Conditioned on that the underlying cryptographic primitives are secure, the protocol* FairStream *satisfies the completeness, fairness, confidentiality against deliverer, timeliness, and non-trivial efficiency properties in the synchronous authenticated network, $\mathcal{G}_s^{\mathsf{ledger}}$-hybrid and stand-alone model.*

Proof. For theorem 2, the guarantee for *completeness*, *confidentiality*, *timeliness* and *non-trivial efficiency* properties are easy to see. Next we mainly focus on the analysis of *fairness*.

Fairness. The fairness guarantee for each party can be reduced to the underlying cryptographic building blocks.

- *Consumer fairness*. This property means that the honest $\mathcal{C}$ needs to pay proportional to what it *de facto* receives even though malicious $\mathcal{P}^*$ and $\mathcal{D}^*$ may collude with each other. This case can be modeled as an adversary $\mathcal{A}$ corrupts $\mathcal{P}$ and $\mathcal{D}$ to provide and deliver the content to $\mathcal{C}$. During the *Stream* phase, $\mathcal{C}$ can stop sending back the receipts any time when an invalid chunk is detected and then raise complaint to the contract to get compensation. If $\mathcal{C}$ receives $\ell \in [n]$ valid chunks, it is ensured that $\mathcal{A}$ who corrupts both $\mathcal{P}$ and $\mathcal{D}$ can *at most* get ℓ receipts and claim payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ and $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$, where the former is paid by $\mathcal{A}$ itself due to collusion. Overall, $\mathcal{C}$ either pays $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$ and obtains ℓ valid chunks or pays nothing. To violate the fairness for $\mathcal{C}$, $\mathcal{A}$ has to break the security of the EU-CMA signature by forging $\mathcal{C}$'s signature. Therefore, the consumer fairness being against the collusion of $\mathcal{P}^*$ and $\mathcal{D}^*$ is ensured. Note that breaking the security of the Merkle tree (i.e., finding another chunk $m_i' \neq m_i$ in position i of m to bind to the same root_m so as to fool the contract to reject $\mathcal{C}$'s PoM) or controlling the execution of smart contract in blockchain, which are of negligible probability due to the second-preimage resistance of hash function in Merkle tree and the fact that contract is modeled as an ideal functionality, can only repudiate the penalty fee $\ddot{\mathbb{B}}_{\mathsf{pf}}$ and would not impact $\mathcal{C}$'s fairness in streaming.
- *Deliverer fairness*. This property states that the honest $\mathcal{D}$ receives the payment proportional to the contributed bandwidth though the malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ may collude with each other. This case can be modeled as $\mathcal{A}$ corrupts both $\mathcal{P}$ and $\mathcal{C}$ to reap $\mathcal{D}$'s bandwidth resource without paying. In the *Stream* phase, if the honest $\mathcal{D}$ delivers ℓ chunks, then it is guaranteed to obtain ℓ or $\ell - 1$ (i.e., $\mathcal{A}$ does not respond with the ℓ-th receipt) receipts. In the *Payout* phase, $\mathcal{A}$ cannot lower the payment for the honest $\mathcal{D}$ since $\mathcal{D}$ can send the ℓ-th or $(\ell - 1)$-th receipt to the contract, which would update the internal state $\mathsf{ctr}_{\mathcal{D}}$ as ℓ or $\ell - 1$. Once $\mathcal{T}_{\mathsf{finish}}$ times out, $\mathcal{D}$ can receive the well-deserved payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ or $(\ell - 1) \cdot \ddot{\mathbb{B}}_{\mathcal{P}}$ from the contract, and *at most* waste bandwidth for delivering one chunk of size η. To violate the fairness for $\mathcal{D}$, $\mathcal{A}$ has to control the execution of smart contract to refuse $\mathcal{D}$'s valid request of claiming payment, which is of negligible probability, and therefore the deliverer fairness being secure against the collusion of malicious $\mathcal{P}^*$ and $\mathcal{C}^*$ is assured.
- *Provider fairness*. This property indicates that the honest $\mathcal{P}$ receives the payment proportional to the number of valid chunks that $\mathcal{C}$ receives. The malicious $\mathcal{D}^*$ and $\mathcal{C}^*$ may collude with each other or $\mathcal{D}^*$ can costlessly create multiple fake $\mathcal{C}^*$ (i.e., Sybil attack), and then cheat $\mathcal{P}$ without true delivery. These cases can be modeled as $\mathcal{A}$ corrupts both $\mathcal{D}$ and $\mathcal{C}$. There are two situations $\mathcal{P}$'s fairness would be violated: (i) $\mathcal{A}$ claims payment (paid by $\mathcal{P}$) without real delivery; (ii) $\mathcal{A}$ obtains content chunks without paying for $\mathcal{P}$. For case (i), $\mathcal{A}$ would try to maximize the payment paid by $\mathcal{P}$ by increasing the $\mathsf{ctr}_{\mathcal{D}}$ via the $(\mathsf{claimDelivery})$ message sent to $\mathcal{G}_s$. However, $\mathcal{G}_s$ would update the counter ctr as $\max\{\mathsf{ctr}_{\mathcal{D}}, \mathsf{ctr}_{\mathcal{P}}\}$ after $\mathcal{T}_{\mathsf{finish}}$ times out, and the intention that $\mathcal{A}$ tries to maximize $\mathsf{ctr}_{\mathcal{D}}$ would correspondingly maximize ctr. Consider that $\mathcal{A}$ wants to claim the payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{P}}, \ell \in [n]$ by letting the $(\mathsf{claimDelivery})$ message contain the index of ℓ while no content is actually delivered, essentially the honest $\mathcal{P}$ can correspondingly receive the payment of $\ell \cdot \ddot{\mathbb{B}}_{\mathcal{C}}$, and therefore a well-deserved net income of $\ell \cdot (\ddot{\mathbb{B}}_{\mathcal{C}} - \ddot{\mathbb{B}}_{\mathcal{P}})$, unless $\mathcal{A}$ can manipulate the execution of smart contract. For case (ii), on one hand, each content chunk is encrypted before receiving the decryption keys from $\mathcal{P}$. Hence, $\mathcal{A}$ has to violate the semantic security of the underlying symmetric encryption scheme to break the provider fairness, which is of negligible probability. On the other hand, during the streaming procedure, $\mathcal{P}$ can always stop revealing the chunk key to $\mathcal{A}$ if no valid receipt for the previous chunk key is responded in time. *At most* $\mathcal{P}$ would lose one content chunk of size η. To violate the fairness, $\mathcal{A}$ again has to control the execution of smart contract, which is of negligible probability, to deny the payment for $\mathcal{P}$ though the submitted receipt is valid. Therefore, the provider fairness against the collusion of malicious $\mathcal{D}^*$ and $\mathcal{C}^*$ is guaranteed.

In sum, the fairness for $\mathcal{C}$ is strictly ensured in Π_{FS}, while for $\mathcal{P}$ and $\mathcal{D}$, the unpaid revealed content for $\mathcal{P}$ and the unpaid bandwidth resource of delivery are bounded to $O(\eta)$ bits. Overall, Π_{FS} satisfies the defined fairness property.

6.5 Discussion and Extension

Fetching from any specific chunk position. It is worth pointing out that both FairDownload and FairStream can support requesting content from any specific chunk with index ζ. This is a vital functionality in the setting of P2P content delivery, because a consumer can resume the content retrieval from any specific chunk if encountering unexpected termination of delivery.

Extension to multiple deliverers. Our previous description of FairDownload and FairStream protocols is explained by delegating the content to merely one deliverer. Nevertheless, to accelerate the delivery, our protocols can be easily extended to support multiple deliverers. In case of κ deliverers, a content m can be split into κ fragments, and each fragment is just the input content for a FairDownload/FairStream instance, cf. Fig. 7, where θ_i indicates the times of repeatable delivery that associated with each content segment. We report more details of the performance in the next section.

Other extensions. Both the downloading and the streaming protocols meet the basic while necessary security requirements for P2P content delivery. Nevertheless, there could be more interesting extensions, e.g., (i) an adaptive mechanism to choose the proper deliverers for each delivery task [40]; (ii) preventing the linkage between the involved parties and the content; (iii) a digital rights management (DRM) scheme to preserve the digital rights against pirating consumers.

7 Implementation and Evaluations

We implement, deploy and evaluate FairDownload and FairStream in the *Ethereum*[8] *Ropsten* network. The arbiter

8. Over 72% DApps are all deployed atop Ethereum according to [41].

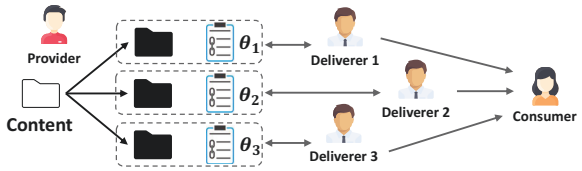


Fig. 7: The multi-deliverer paradigm.

contract is implemented in Solidity and split into *Optimistic* and *Pessimistic* modules, the former of which is executed if no dispute and the later is additionally invoked when dispute occurs. The contracts are only deployed once and can be used for multiple times to facilitate many deliveries, such that the one-time deployment cost can be amortized.

Cryptographic instantiations. The hash function uses *keccak256* and the digital signature is via ECDSA over secp256k1 curve. The encryption of each chunk m_i with key k_i is instantiated as: parse m_i into t 32-byte blocks $(m_{i,1}, \ldots, m_{i,t})$ and output $c_i = (m_{i,1} \oplus \mathcal{H}(k_i||1), \ldots, m_{i,t} \oplus \mathcal{H}(k_i||t))$. The decryption is same to the encryption. We construct public key encryption scheme based on ElGamal: Let $\mathcal{G} = \langle g \rangle$ be G_1 group over *alt-bn128* curve [42] of prime order q, where g is group generator; The private key $k \leftarrow_{\$} \mathbb{Z}_q$, the public key $h = g^k$, the encryption $\mathsf{VEnc}_h(m) = (c_1, c_2) = (g^r, m \cdot g^{kr})$ where $r \leftarrow_{\$} \mathbb{Z}_q$ and m is encoded into $\mathcal{G}$ with Koblitz's method [43], and the decryption $\mathsf{VDec}_k((c_1, c_2)) = c_2/c_1^k$. To augment ElGamal for verifiable decryption, we adopt Schnorr protocol [44] for Diffie-Hellman tuples with using Fiat-Shamir transform [45] in the random oracle model. The concrete construction is illustrated in Fig. 8.

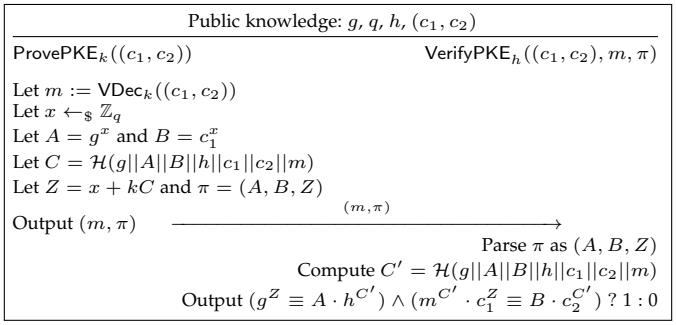


Fig. 8: The construction of ProvePKE and VerifyPKE.

7.1 FairDownload On-Chain Evaluation

Table 1 presents the on-chain gas costs for all functions in Π_{FD} protocol. We stress that instantiating and deploying our protocols utilizing other cryptocurrencies such as Ethereum classic [9], Solana [10] or Cardano [11] may further decrease the costs for execution.

Optimistic costs. Without complaint the protocol Π_{FD} only executes the functions in *Deliver* and *Reveal* phases when a new consumer joins in, yielding the total cost of 478,996 *gas* for all involved parties except the one-time cost for deployment and the *Prepare* phase. Note that such an on-chain cost is *constant* no matter how large the content size

9. https://ethereumclassic.org/
10. https://solana.com/
11. https://cardano.org/

TABLE 1: The on-chain costs of all functions in FairDownload

Phase	*Function*	*Caller*	*Gas Costs*
Deploy	(Optimistic)	$\mathcal{P}$	3 083 841
Deploy	(Pessimistic)	$\mathcal{P}$	2 924 903
Prepare	start	$\mathcal{P}$	165 965
	join	$\mathcal{D}$	70 865
	prepared	$\mathcal{D}$	33 845
Deliver	consume	$\mathcal{C}$	150 801
	delivered	$\mathcal{C}$	61 492
	verifyVFDProof	$\mathcal{D}$	66 881
Reveal	revealKeys	$\mathcal{P}$	138 763
	payout	$\mathcal{G}_d$	61 059
Dispute Resolution	wrongRK	$\mathcal{C}$	26 079
	PoM	$\mathcal{C}$	392 017
Reset	reset	$\mathcal{P}$	74 061

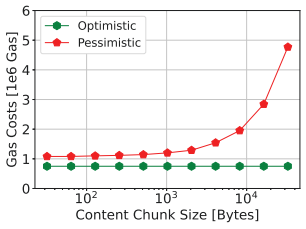


Fig. 9: Gas costs with different chunk sizes in Π_{FD}.

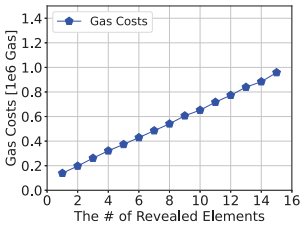


Fig. 10: Gas for various # of revealed elements in Π_{FD}.

or the chunk size are, as illustrated in Fig. 9 optimistic costs. In a worse case, up to $\log n$ elements in the key tree KT need to be revealed. In that case, Fig. 10 depicts the relationship between the number of revealed elements and the corresponding gas costs.

Pessimistic costs. When complaint arises, the arbiter contract involves to resolve dispute. The cost of executing wrongRK function relates to the values of n, ctr and $|erk|$, and in Table 1, the cost is evaluated on $n \equiv \mathsf{ctr} \equiv 512$, and $|erk| \equiv 1$. The cost of PoM function validating misbehavior varies by the content chunk size η, as depicted in Fig. 9 pessimistic costs. The results demonstrate the on-chain costs increase linearly in the chunk size (mostly due to chunk decryption in contract). Fig. 11 illustrates the total gas costs for each party within a protocol instance for Π_{FD}.

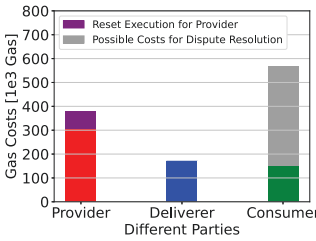


Fig. 11: Gas for each party in a protocol instance for Π_{FD}.

Encryption/decryption efficiency. The efficiency of off-chain symmetric encryption (by providers) and decryption (by consumers) for content chunks is about 22.46Mb/s, which could be further improved with more engineering optimizations, e.g., via multi-thread programming.

7.2 FairStream On-Chain Evaluation

Table 2 illustrates the on-chain costs of all functions in FairStream. As the deployment of contract and the *Prepare* phase can be executed only *once*, we discuss the costs in both optimistic and pessimistic modes after a new consumer participates in, i.e., starting from the *Stream* phase.

Optimistic costs. When no dispute occurs, the Π_{FS} protocol executes the functions in *Stream* and *Payout* phases except the PoM function for verifying proof of misbehavior, yielding a total cost of 416,007 *gas* for all involved parties. Note

TABLE 2: The on-chain costs of all functions in FairStream

Phase	*Function*	*Caller*	*Gas Costs*
Deploy	(Optimistic)	$\mathcal{P}$	2 000 615
Deploy	(Pessimistic)	$\mathcal{P}$	1 024 714
Prepare	start	$\mathcal{P}$	165 971
	join	$\mathcal{D}$	53 865
	prepared	$\mathcal{D}$	33 913
Stream	consume	$\mathcal{C}$	127 190
	received	$\mathcal{C}$	33 887
	receiveTimeout	$\mathcal{G}_s$	33 879
	PoM	$\mathcal{C}$	112 176
Payout	claimDelivery	$\mathcal{D}$	77 091
	claimRevealing	$\mathcal{P}$	77 101
	finishTimeout	$\mathcal{G}_s$	100 738
Reset	reset	$\mathcal{P}$	74 056

only one of the (received) and (receiveTimeout) functions would be invoked. The (claimDelivery) and (claimRevealing) functions may be called in different orders. Besides, the costs in the optimistic mode is *constant* regardless of the content size and chunk size, as shown in Fig. 12 optimistic costs.

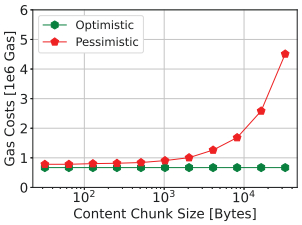


Fig. 12: Gas costs with different chunk sizes in Π_{FS}.

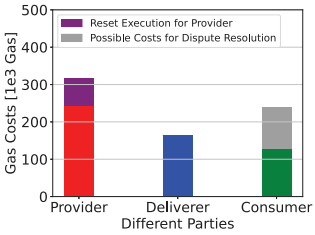


Fig. 13: Gas for each party in a protocol instance for Π_{FS}.

Pessimistic costs. When complaint arises, the contract is involved to resolve dispute. The cost of PoM function: (i) increases slightly in the number of chunks n since it computes $O(\log n)$ hashes to verify the Merkle proof; (ii) increase linearly in the the content chunk size η due to chunk decryption in the contract, as depicted in the pessimistic case in Fig. 12, which exhibits lower overall costs than the downloading setting as no verification of verifiable decryption proof is needed.

7.3 Evaluating Downloading and Streaming Efficiency

Experiment environment. To demonstrate the efficiency of Π_{FD} and Π_{FS}, we conduct experiments in both LAN and WAN settings, whose bandwidth are shown in Fig. 14. In LAN, three VM instances on three servers reside on the same rack and connect with different switches, and the servers are all Dell PowerEdge R740. The VMs have the same configuration of 8 vCPUs, 24 GB memory and 800 GB hard drive. In WAN, three *Google cloud* VM instances are initialized in *us-east4-c*, *us-east1-b* and *europe-north1-a*, respectively. Each VM is configured with 2 vCPUs, 4 GB memory and 10 GB hard drive. Considering that $\mathcal{P}$ possesses information to choose the proper deliverer $\mathcal{D}$ to ensure better delivery quality (e.g., less delay from $\mathcal{D}$ to $\mathcal{C}$), the link between $\mathcal{D}$ and $\mathcal{C}$ is therefore evaluated in a higher bandwidth environment.

Downloading efficiency. Fig. 15a and 15b illustrate the latency of downloading a content of 512 MB in LAN and WAN settings with various number of deliverers and chunk size. We observe that: (i) obviously, multiple deliverers can accelerate the delivery; (ii) as expected, the latency in WAN is higher and the delivery is less stable than that in LAN; (iii) the latency becomes stable with the increased chunk size. This is reasonable since with smaller chunk size, the communication rounds would be large, while increased chunk size would cause larger single-round latency, leading to higher total latency.

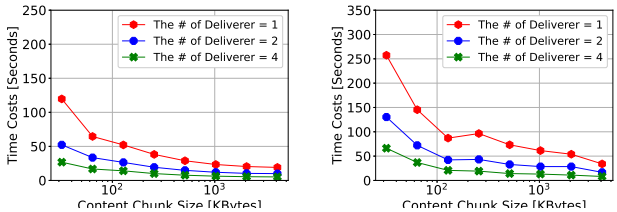


Fig. with

Streaming efficiency. Fig. 16a and 16b illustrate the time costs of consecutively streaming 512 various-size content chunks in both LAN and WAN with one deliverer, which indicate that: (i) obviously the time costs increase due to the growth of chunk size; (ii) the delivery process remains stable with only slight fluctuation, as reflected by the slope for each chunk size. Furthermore, Fig. 17a and 17b depict the latency for streaming a content of 512 MB in LAN and WAN settings with various chunk size and number of deliverers. Besides the three similar observations as in the downloading setting, the streaming process has an overall higher latency. This is because each steaming round also involves the provider for key revealing, as shown in Fig. 5.

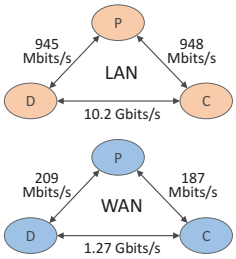
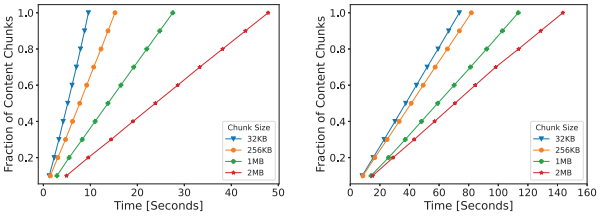
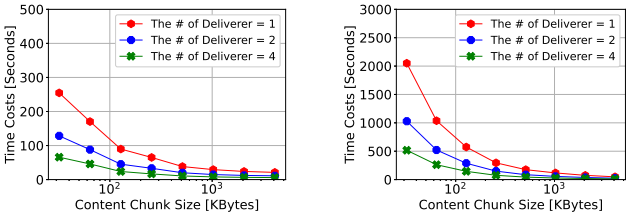


Fig. 14: Bandwidths among entities for Π_{FD} and Π_{FS}.



(a) LAN. (b) WAN.

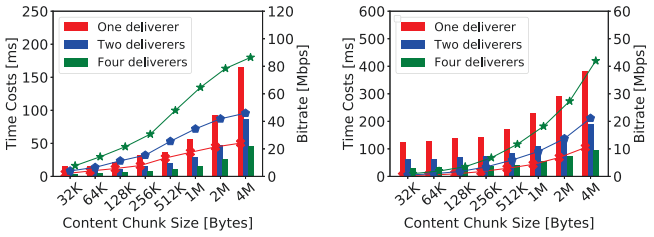
Fig. 16: Time costs of streaming 512 content chunks with various chunk size and one deliverer.



(a) LAN. (b) WAN.

Fig. 17: The latency of streaming a content 512 MB with various chunk size and different number of deliverers.

Fig. 18a and 18b further illustrate the average time costs for delivering each chunk of various sizes and the corre-



(a) LAN. (b) WAN.

Fig. 18: The average chunk delivery latency and the corresponding bitrate of streaming a content 512 MB with various chunk size and different number of deliverers.

sponding bitrate (i.e., the number of bits that are delivered per unit of time). The results show that the bitrate can reach 10 Mpbs even in the public network (i.e., WAN) with just one deliverer, which in principle is sufficient to support high-quality content streaming, e.g., the video bitrate for HD 720 and HD 1080 are *at most* 4 Mbps and 8 Mbps, respectively [46].

8 Related Work

We review the related technologies and discuss their insufficiencies in the context of P2P content delivery. Table 3 summarizes the comparisons with other related works.

P2P information exchange schemes. Many works [14, 26–30, 48] focused on the basic challenge to incentivize users in the P2P network to voluntarily exchange information. However, these schemes have not been notably successful in combating free-riding problem and strictly ensuring the fairness. Specifically, the schemes in BitTorrent [48], BitTyrant [14], FairTorrent [26], PropShare [30] support direct reciprocity (i.e., the willingness for participants to continue exchange basically depends on their past direct interactions, e.g., the *Tit-for-Tat* mechanism in BitTorrent) for participants, which cannot accommodate the *asymmetric* interests (i.e., participants have distinct types of resources such as bandwidth vs. cryptocurrencies) in the P2P content delivery setting. For indirect reciprocity (e.g., reputation, currency, credit-based) mechanisms including Eigentrust [27], Dandelion [28] are obsessed by Sybil attacks, e.g., a malicious peer could trivially generate a sybil peer and "deliver to himself" and then rip off the credits. For T-chain [29], it still considers rational attackers and cannot strictly ensure the delivery fairness as an adversary can waste a lot of bandwidth of deliverers though the received content is encrypted.

Fair exchange and fair MPC. There are also intensive works on fair exchange protocols in cryptography. Some traditional ways hinge on a TTP [18, 19, 25, 49] to solve this problem, which has been reckon hard to find such a TTP in practice. To avoid the available TTP requirement [22], some other studies [16, 17, 50, 51] rely on the "gradual release" approach, in which the parties act in turns to release their private values bit by bit, even if one malicious party aborts, the honest one can recover the desired output by investing computational resources (in form of CPU time) comparable to that of the adversary uses. Recently, the blockchain offers an attractive way to instantiate a non-private TTP, and a few results [20, 21, 23, 24] leverage this innovative decentralized infrastructure to facilitate fair exchange and fair MPC despite the absence of honest majority. Unfortunately, all above fair exchange and fair MPC protocols fail to guarantee *delivery fairness* in the specific P2P content delivery setting, as they cannot capture the fairness property for the special exchanged item, i.e., bandwidth.

State channels. A state channel establishes a private P2P medium, managed by pre-set rules, allowing the involved parties to update state unanimously by exchanging authenticated state transitions off-chain [52]. Though our protocols can be reckon as the application of payment channel networks (PCNs) (or more general state channels [53]) yet there are two key differences: i) fairness in state channels indicates that an honest party (with valid state transition proof) can always withdraw the agreed balance from the channel [52], while our protocols, dwelling on the delivery fairness, ensure the bandwidth contribution can be quantified and verified to generate such a state transition proof; ii) state channels essentially allow any two parties to interact, while our protocols target the interactions among any three parties with a totally different payment paradigm [3].

Decentralized content delivery. There exist some systems that have utilized the idea of exchanging bandwidth for rewards to incentivize users' availability or honesty such as Dandelion [28], Floodgate [54]. However, different drawbacks impede their practical adoption, as discussed in [3]. Here we elaborate the comparison with three protocols, i.e., CacheCash [3], Gringotts [6], and Ding *et al.* [47], that target the similar P2P content delivery scenario.

Application Scenario. Typically, the P2P content delivery setting involves asymmetric exchange interests of participants, i.e., the consumers expect to receive a specific content while the providers and the deliverers would share their content (valid due to the digest) and bandwidth in exchange of well-deserved payments/credits, respectively. Unfortunately, in [3], [6] and [47], the raw contents are delegated to deliverers, resulting in the lack of content confidentiality and exchange fairness, i.e., a malicious consumer can pretend to be or collude with a deliverer to obtain the plaintext content without paying for the provider.

Delivery Fairness. In [3], a set of deliverers are selected to distribute the chunks in parallel, which may cause the loss of bandwidth for all chunks in the worst case. [6] typically requires the deliverer to receive a receipt (for acknowledging the resource contribution) only after multiple chunks are delivered, posing the risk of bandwidth loss for delivering multiple chunks. In [47], the deliverer commits a transaction containing chunk-related validity proof to blockchain before each chunk delivery. We consider that the consumer would stop receiving the chunks upon detecting invalid proof though not explicitly stated, yielding the bandwidth loss for sending a chunk and a transaction (of several Kbytes). On the contrary, our protocols ensures that the unfairness of delivery is bounded to one chunk of size η.

On-chain Costs. In [3], the deliverers can obtain *lottery tickets* (i.e., similar to "receipts") from the consumer after each "valid" chunk delivery. The on-chain costs is highly pertinent to the winning probability p of tickets. E.g., $p = \frac{1}{n}$ means that on average the deliverer owns a winning ticket after n chunks are delivered, or $p = 1$ indicates that the deliverer receives a winning ticket after each chunk delivery, leading to at most $O(n)$ on-chain costs of handling redeem

TABLE 3: Security comparison with related representative approaches (Can: ✓, Not fully: $*$, Cannot: $\times$). TX is the transaction sent to blockchain for each chunk in [47], whose size is about 3 Kb. It also applies to Table 4.

Schemes \ Features		What to Exchange? (Incentive Type)	Confidentiality c.f., Sec.4	Exchange Fairness c.f., Sec.4	Delivery Fairness (the worst case), c.f., Sec.4
P2P Information Exchange	BitTorrent [48]	Files ↔ Files (Tit-for-Tat)	×	$*$	×
	Dandelion [28]	Files ↔ Credits (Reputation)	×	$*$	×
	T-Chain [29]	Files ↔ Files (Tit-for-Tat)	✓	$*$	×
Decentralized Content Delivery	CacheCash [3]	Bandwidth ↔ Coins (Monetary)	×	×	*all* chunks' bandwidth
	Gringotts [6]	Bandwidth ↔ Coins (Monetary)	×	×	*multiple* chunks' bandwidth
	Ding *et al.* [47]	Files ↔ Usual Payment (No Incentive)	×	×	*one* chunk + *one* TX's bandwidth
	Our Protocols	Bandwidth/Files ↔ Coins (Monetary)	✓	✓	*one* chunk's bandwidth

TABLE 4: Performance comparison with related works (n is the number of content chunks, e.g., $n = 1024$; $|\sigma|$ is the signature size of 32 bytes; $|h|$ is the hash size of 32 bytes; $|\mathsf{MAC}|$ is the message authentication code size of 16 bytes; tkt is the lottery ticket with size of 110 bytes; D and S indicate that the protocol supports downloading and streaming respectively).

Protocols	Comm. Rounds	Comm. Overhead (Besides content m)	On-Chain Costs
CacheCash [3]	$6n$ (D)	$n(\|\sigma\|+3\|\mathsf{tkt}\|+2\|h\|)$, (∼426Kb)	$[o(1), O(n)]$
Gringotts [6]	$5n$ (D, S)	$n(3\|\sigma\|+\|\mathsf{TX}\|+\|h\|)$, (∼3Mb)	$O(n)$
Ding *et al.* [47]	$2n$ (D, S)	$n(\|\mathsf{TX}\|+\|\mathsf{MAC}\|)$, (∼3Mb)	$O(n)$
Our Protocols	$2n$+3 (D), $4n$+3 (S)	$2n\|\sigma\|+(2n-1)\times\|h\|$, (∼128Kb for D) $4n\|\sigma\|+(2n-1)\times\|h\|$, (∼192Kb for S)	$\tilde{O}(1)$

transactions. For [6], it stores a record on the blockchain after each chunk delivery, and therefore the on-chain costs is in $O(n)$. For [47], a transaction is submitted to blockchain with chunk validity proof before every chunk delivery, leading to $O(n)$ on-chain costs. For our protocols, the on-chain costs is bounded to $\tilde{O}(1)$.

9 Conclusion

We present the first two fair P2P content delivery protocols atop blockchain to support fair P2P downloading and streaming, respectively. They enjoy strong fairness guarantees to protect any of the content provider, the content consumer, and the content deliverer from being ripped off by other colluding parties. Detailed complexity analysis and extensive experiments of prototype implementations are performed and demonstrate the practicality and efficiency.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments. Yuan is supported in part by National Key R&D Project of China (No. 2022YFB2701600), NSFC (No. 62102404) and the Youth Innovation Promotion Association CAS. Songlin He is supported in part by Sichuan Science and Technology Program (No. 2021YFG0040). Qiang Tang is supported in part by gifts from Ethereum Foundation, Stellar Foundations and Protocol Labs. Guiling Wang is supported in part by the FHWA EAR Project (No. 693JJ320C000021).

References

[1] S. He, Y. Lu, Q. Tang, G. Wang, and C. Q. Wu, "Fair peer-to-peer content delivery via blockchain," in *ESORICS*. Springer, 2021, pp. 348–369.

[2] Akamai, "Akamai," 2021. [Online]. Available: https://www.akamai.com/

[3] G. Almashaqbeh, "Cachecash: A cryptocurrency-based decentralized content delivery network," Ph.D. dissertation, Columbia University, 2019.

[4] N. Anjum, D. Karamshuk, M. Shikh-Bahaei, and N. Sastry, "Survey on peer-assisted content delivery networks," *Computer Networks*, pp. 79–95, 2017.

[5] W. Wang, D. Niyato, P. Wang, and A. Leshem, "Decentralized caching for content delivery based on blockchain: A game theoretic perspective," in *Proc. IEEE ICC*, 2018.

[6] P. Goyal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "Secure incentivization for decentralized content delivery," in *2nd USENIX Workshop on Hot Topics in Edge Computing*, 2019.

[7] Swarm, 2020. [Online]. Available: https://swarm.ethereum.org/

[8] J. Benet, "IPFS-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[9] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE S&P*, 2014, pp. 475–490.

[10] Filecoin, "Filecoin spec." 2020. [Online]. Available: https://spec.filecoin.io/

[11] B. Fan, J. C. Lui, and D.-M. Chiu, "The design trade-offs of bittorrent-like file sharing protocols," *IEEE/ACM Transactions on Networking*, pp. 365–376, 2008.

[12] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proc. ACM EC*, 2004.

[13] T. Locher, P. Moore, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *HotNets*, 2006.

[14] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent," in *Proc. NSDI*, 2007.

[15] G. Hardin, "The tragedy of the commons," *Journal of Natural Resources Policy Research*, pp. 243–253, 2009.

[16] M. Blum, "How to exchange (secret) keys," in *Proc. ACM STOC*, 1983, pp. 440–447.

[17] I. B. Damgård, "Practical and provably secure release of a secret and exchange of signatures," *Journal of Cryptology*, pp. 201–222, 1995.

[18] N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE Journal on Selected Areas in communications*, pp. 593–610, 2000.

[19] A. Küpçü and A. Lysyanskaya, "Usable optimistic fair exchange," in *Cryptographers RSA Conference*, 2010, pp. 252–267.

[20] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proc. ACM CCS*, 2018, pp. 967–984.
[21] G. Maxwell, "The first successful zero-knowledge contingent payment," Feb. 2016. [Online]. Available: https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/.
[22] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Tech. Rep., 1999.
[23] 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, pp. 705–734.
[24] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *Proc. ACM CCS*, 2017, pp. 719–728.
[25] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin, "Making p2p accountable without losing privacy," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, 2007, pp. 31–40.
[26] A. Sherman, J. Nieh, and C. Stein, "Fairtorrent: a deficit-based distributed algorithm to ensure fairness in peer-to-peer systems," *IEEE/ACM TON*, 2012.
[27] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proc. WWW*, 2003, pp. 640–651.
[28] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki, "Dandelion: Cooperative content distribution with robust incentives." in *Proc. USENIX ATC*, 2007.
[29] K. Shin, C. Joe-Wong, S. Ha, Y. Yi, I. Rhee, and D. S. Reeves, "T-chain: A general incentive scheme for cooperative computing," *IEEE/ACM TON*, 2017.
[30] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," *ACM SIGCOMM Computer Communication Review*, 2008.
[31] ENISA, "ENISA threat landscape 2020 -botnet," 2020. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-botnet.
[32] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Advances in Cryptology CRYPTO*, 2020, pp. 126–144.
[33] S. Janin, K. Qin, A. Mamageishvili, and A. Gervais, "Filebounty: Fair data exchange," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 357–366.
[34] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.
[35] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT*. Springer, 2015, pp. 281–310.
[36] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO*. Springer, 2017, pp. 357–388.
[37] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE S&P*, 2016, pp. 839–858.
[38] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, pp. 1–32, 2014.
[39] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *IEEE S&P*. IEEE, 2021, pp. 1348–1366.
[40] G. Yan and J. Li, "Towards latency awareness for content delivery network caching," in *USENIX Annual Technical Conference (USENIX ATC)*, 2022, pp. 789–804.
[41] State of the Dapps, "Dapp Statistics," 2022. [Online]. Available: https://www.stateofthedapps.com/stats
[42] C. Reitwiessner, "EIP-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128," Ethereum Improvement Proposals, No. 196, 2017. [Online]. Available: https://eips.ethereum.org/EIPS/eip-196
[43] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, pp. 203–209, 1987.
[44] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology – CRYPTO*, 1989, pp. 239–252.
[45] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology – CRYPTO*, 1986, pp. 186–194.
[46] IBM, "Internet connection and recommended encoding settings." [Online]. Available: https://support.video.ibm.com/hc/en-us/articles/207852117-Internet-connection-and-recommended-encoding-settings
[47] Y. Ding, Z. Wu, and L. Xie, "Enabling manageable and secure hybrid p2p-cdn video-on-demand streaming services through coordinating blockchain and zero-knowledge," *IEEE MultiMedia*, 2022.
[48] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, 2003, pp. 68–72.
[49] S. Micali, "Simple and fast optimistic protocols for fair electronic exchange," in *Proc. ACM PODC*, 2003, pp. 12–19.
[50] B. Pinkas, "Fair secure two-party computation," in *Advances in Cryptology – EUROCRYPT*, 2003, pp. 87–105.
[51] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang, "Resource fairness and composability of cryptographic protocols," in *Theory of Cryptography Conference*, 2006, pp. 404–428.
[52] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *FC*. Springer, 2020, pp. 201–226.
[53] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *FC*. Springer, 2019, pp. 508–526.
[54] S. K. Nair, E. Zentveld, B. Crispo, and A. S. Tanenbaum, "Floodgate: A micropayment incentivized p2p content delivery network," in *Proc. ICCCN*, 2008, pp. 1–7.