Atomic and Fair Data Exchange via Blockchain

Ertem Nusret Tas*
Stanford University

István András Seres†
Eötvös Loránd University

Yinuo Zhang‡
University of California, Berkeley

Márk Melczer§
Eötvös Loránd University

Mahimna Kelkar¶
Cornell University
Cornell Tech

Joseph Bonneau‖
A16Z Crypto Research
New York University

Valeria Nikolaenko**
A16Z Crypto Research

ABSTRACT

We introduce a blockchain *Fair Data Exchange* (FDE) protocol, enabling a storage server to transfer a data file to a client atomically: the client receives the file if and only if the server receives an agreed-upon payment. We put forth a new definition for a cryptographic scheme that we name verifiable encryption under committed key (VECK), and we propose two instantiations for this scheme. Our protocol relies on a blockchain to enforce the atomicity of the exchange and uses VECK to ensure that the client receives the correct data (matching an agreed-upon commitment) before releasing the payment for the decrypting key. Our protocol is trust-minimized and requires only constant-sized on-chain communication, concretely 3 signatures, 1 verification key, and 1 secret key, with most of the data stored and communicated off-chain. It also supports exchanging only a subset of the data, can amortize the server's work across multiple clients, and offers a general framework to design alternative FDE protocols using different commitment schemes. A prominent application of our protocol is the Danksharding data availability scheme on Ethereum, which commits to data via KZG polynomial commitments. We also provide an open-source implementation for our protocol with both instantiations for VECK, demonstrating our protocol's efficiency and practicality on Ethereum.

1 INTRODUCTION

Cloud data storage is a rapidly growing global market ($\approx$ 18.5% annual growth rate) with an estimated value of $78.6 billion in 2022 [50]. The volume of cloud data being stored today is counted in tens to hundreds of zettabytes (1 zettabyte = 2^{70} bytes) [61]. Data economy globally is much larger and is estimated at the trillions of US dollars (based on Canadian [19, 65] and European [45] assessment). Recent regulations [56] enforce companies to make the generated data more widely accessible potentially helping expand the global market for cloud data. Fair and secure protocols to purchase access to data are essential to unlock the massive potential of global data markets. However, most approaches for accessing data today are either subscription-based, where the client pays the server in advance and must trust the reputation of the server to deliver the data, or altruistic, where either the server provides the data free of charge or the data is exchanged between the users themselves as in BitTorrent [24]. The former approach does not safeguard the client from a malicious server that receives payment without fulfilling the data request, while the latter lacks incentives for clients to offer data for download, leading to free-riding and limited capacity. Moreover, the vast majority of storage systems do not provide data integrity guarantees to users [5].

While blockchains were originally conceived as payment systems, it was quickly observed that they could also be used for data storage [23]. Bitcoin [52] adopted a dedicated mechanism (OP_RETURN) for storing data on-chain in 2014, which has been used for a variety of purposes [68]. Ethereum [70] has always supported arbitrary data storage (calldata) as required by its Turing-complete smart contract platform (EVM). As append-only, immutable, and distributed ledgers, blockchains can provide data storage that is robust against faults and abuse of power.

However, by themselves, blockchains are highly limited in both storage and computational capacity, making on-chain data storage expensive. For example, storing 1 megabyte of data on Ethereum as calldata would cost approximately $2,100 at the time of writing. While many blockchain projects are working to improve capacity and reduce costs, it is conjectured that these systems will always be limited as increased capacity is at odds with maintaining security and decentralization (an observation dubbed the *blockchain trilemma* [14]).

Limited on-chain capacity has led to the development of so-called *layer-2* (L2) solutions (*e.g.*, rollups, validiums) for both computation and data storage. These solutions perform computation and store data off-chain while enabling the main blockchain (now called a layer-1) to verify off-chain computation via verifiable computation (as [63, 67, 71]) and off-chain storage via proofs-of-storage and replication (as in [34]). In the case of off-chain storage, the blockchain typically receives a succinct commitment to the data uploaded to the servers. It can then use the commitment to verify proofs-of-storage (also called proofs-of-retrievability [13]) or proofs-of-replication [35, 36] attesting to the persistence of the data.

A current gap in both theory and practice is that, while these proofs provide a natural mechanism to pay servers to *store* data, they don't provide a means to pay for actually serving the data when requested by the clients. Today's protocols only incentivize storage

*nusret@gmail.com. Big part of the work was done at a16z crypto research.
†seresistvanandras@gmail.com. Big part of the work was done at a16z crypto research.
‡yinuo.yz@gmail.com. Big part of the work was done at a16z crypto research.
§melczer7@gmail.com
¶mahimna@cs.cornell.edu. Big part of the work was done at a16z crypto research.
‖jbonneau@gmail.com
**valeria.nikolaenko@gmail.com

and assume servers will provide download access essentially "for free." This is problematic for two reasons: First, transferring the data comes with its own costs, which servers should be compensated for, proportional to the number of times the data is downloaded. Second, without any incentives, malicious servers might store the data (and receive payment for doing so) but never respond to legitimate download requests.[1] Storage is useless unless the data is made available for access.

The FDE problem. To fill this gap, in this work, we introduce the concept of a blockchain Fair Data Exchange (FDE) protocol, where clients and servers have cryptographic fairness and data integrity guarantees, and formalize its syntax and security properties (*cf.* Section 4.1). We notice that in most scenarios, it's natural to assume that the client holds a short cryptographic commitment to the stored data that the server possesses in its entirety. Therefore, we informally require that in FDE protocols, the server will receive the payment from the client if and only if the client receives the data beneath the commitment.

A prominent application of our scheme is ProtoDanksharding (EIP-4844) [15], a new data availability scheme being designed for Ethereum.In ProtoDanksharding, validators store special kind of blockchain data designated for storage and not for execution, called blob-data. The blob-data expires, but the KZG-commitment to the data is persisted. After the blob-data expires, data-availability servers or full-nodes might voluntarily continue storing it, but they are not required to. Our FDE protocol helps incentivize nodes to continue storing the data by enabling the users to purchase expired data from nodes in a trust-minimized, fair, and efficient way. Our scheme also applies to Danksharding [29], an extension that would disseminate blob-data to nodes avoiding data-replication.

A strawman blockchain-based solution. Pagnia and Gärtner's well-known impossibility result states that *fair exchange* is impossible without a *trusted third party* (TTP) [54]. A straightforward but inefficient solution to the FDE problem using an L1 blockchain (such as Ethereum) as the TTP would be as follows: Suppose the client has a short commitment C to the data that it wishes to obtain from the server. The client locks some funds for payment in an FDE smart contract on the blockchain along with the commitment C. The contract enables the server to receive these funds only if it publishes the requested data, that correctly verifies against C in a smart contract. While this is a secure and fair solution, it is highly inefficient as it requires all data to be written to the L1 chain. If the data is too large to fit in a single transaction, the parties would also need to interact multiple times with the blockchain, adding latency and additional costs. Our goal is to design a constant-round protocol with a small (ideally constant) storage footprint on the blockchain.

Our approach. We introduce a new FDE protocol (*cf.* Figure 1), where a client and a server atomically exchange funds for data committed using the KZG polynomial commitment scheme [47]. Our choice of KZG is due to two reasons: First, it has the ability to provide constant-size commitments and batchable opening proofs, which are particularly useful when a client only wants to retrieve a subset of the committed data. Second, KZG commitments are also used in Danksharding, making our protocol a natural fit.

We describe the blue-print of our FDE protocol in Figure 1. We consider data stored as a length $n+1$ vector of field elements. Let $\phi(\cdot)$ with commitment C_ϕ be the degree n polynomial whose evaluation over the points $\{0,\dots,n\}$ correspond to the data entries. To start the exchange of data for funds, the server posts a public verification key vk to a contract along with the specific details of the data exchange, *e.g.*, the agreed price and the client's blockchain address (step 1). Subsequently, the server sends the client (off-chain) the encryptions $\{\mathrm{ct}_i\}_{i=0}^n$ of the data points $\{\phi(i)\}_{i=0}^n$ along with a proof which shows that indeed for all $i \in [n]$, ct_i is the correct encryption of the evaluation $\phi(i)$ at index i of the polynomial $\phi(\cdot)$ committed by C_ϕ, under some decryption key sk committed by vk (step 2). After receiving the encrypted data, the client locks up funds in the on-chain contract if the details of the exchange and the proofs are correct with respect to the ciphertexts and the KZG commitment (step 3). The server can redeem the payment only if it provides the (secret) decryption key sk that matches its previously submitted verification key vk (step 4). The client can then read sk from the contract and decrypt the ciphertexts to obtain the data (steps 5 and 6). If the server does not reveal a decryption key, the client can withdraw its locked coins after a timeout. We show that our FDE protocol satisfies *correctness*, *client-fairness* (the server cannot receive any payment if the client does not obtain the data) and *server-fairness* (the client cannot learn anything about the data without paying the server).

Our protocol also extends to the multi-client setting in which multiple clients download the same data. We introduce a multi-client FDE protocol where, via preprocessing, we amortize the server's computation cost to serve multiple clients. In certain applications, this protocol can also reduce the clients' computation by having the blockchain verify the server's preprocessing output.

For the proof-system/encryption scheme combination required in FDE we make the following important observation: the decryption key for the ciphertext is produced together with the ciphertext itself, and the decryption key is only used once. Henceforth more efficient encryption schemes could be used, including symmetric-key or one-time schemes, opening a broader design space for the underlying cyprographic primitive that we introduce in this work: Verifiable Encryption under Committed Key (VECK). The blockchain FDE protocol is then built using VECK in a black-box way. We build two instantiations for VECK: using a symmetric-key version of exponential ElGamal encryption and using public-key version of Paillier encryption. Our instantiations of VECK allow encrypting evaluations of a polynomial under a KZG commitment, however we set forth more general definitions to capture other potential applications or commitment schemes.

Implementation and benchmarks. We provide a practical, open-source implementation[2] of our FDE protocols accompanied by asymptotic and concrete performance evaluations (*cf.* Table 1 for a comparison of our performance metrics with prominent related work). Our FDE protocols are practical and have low round complexity (3 rounds). For the server, the cost of proving the consistency of ciphertexts for 4,096 KZG-committed BLS12-381 field elements

[1] Game-theoretic approaches to enforce responses require posting the data on the blockchain in the worst-case [69].

[2] https://github.com/PopcornPaws/fde.

	# Rounds	data com.	$\lvert\pi_{\mathrm{disp}}\rvert$	$\mathcal{S} \to \mathcal{C}$ comm.	$\mathcal{C} \to \mathcal{E}$ comm.	$\mathcal{S} \to \mathcal{E}$ comm.	Online
FairSwap [30]	5	Merkle tree	$3\log(k)\lvert\mathsf{H}\rvert$	$(k+1)\lvert\mathsf{H}\rvert$	$2\lvert\mathsf{H}\rvert + \lvert\sigma\rvert$	$2\lvert\sigma\rvert + \lvert\mathsf{G}\rvert$	✓
FileBounty [46]	k	Merkle-Damgård [28]	3G	$k(\lambda + \lvert\mathsf{H}\rvert)$	$k\lvert\sigma\rvert$	$2\lvert\sigma\rvert$	✗
FairDownload [44]	k	Merkle tree	$\log(k)\lvert\mathsf{H}\rvert$	$k\lvert\mathrm{ct}\rvert$	$k\lvert\sigma\rvert$	$2\lvert\sigma\rvert + O(\log k)\lvert\mathsf{H}\rvert$	✗
FDE-ElGamal (cf. Figure 2)	3	KZG [47]	N/A	$k8\lvert\mathrm{ct}\rvert + 6\lvert\mathsf{G}\rvert$	$\lvert\sigma\rvert$	$2\lvert\sigma\rvert + 2\lvert\mathsf{G}\rvert$	✓
FDE-Paillier (cf. Figure 4)	3	KZG [47]	N/A	$k(2\lvert\mathrm{ct}\rvert + \lvert\mathsf{F}_p\rvert)$	$\lvert\sigma\rvert$	$2\lvert\sigma\rvert + 2\lvert\mathsf{G}\rvert$	✓

Table 1: Comparing our *FDE* protocols with our closest related works. k denotes the number of exchanged chunks of data. Each data chunk has λ bits. $\mathcal{S}, \mathcal{C}, \mathcal{E}$ denotes the server, the client, and the payment environment (typically a smart contract), respectively. The size of the proof submitted during the dispute protocol is denoted as $\lvert\pi_{\mathrm{disp}}\rvert$. We say a fair exchange protocol is online if the protocol assumes that $\mathcal{C}$ and $\mathcal{S}$ must be online during the entire execution of the protocol. Here, $\lvert\mathsf{H}\rvert$, $\lvert\mathsf{G}\rvert$, and $\lvert\mathsf{F}_p\rvert$ refer to the size of a single hash function output, (an elliptic curve or Z_{N^2}) group element, and field element, respectively, whereas $\lvert\sigma\rvert$ refers to the signature size. N/A means not applicable.

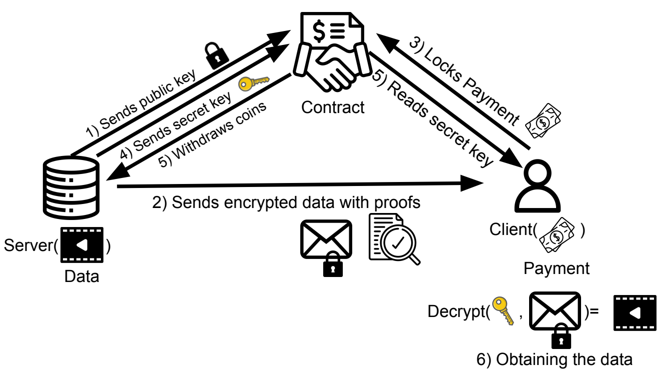

Figure 1: The blueprint of FDE protocols on Ethereum that relies on a smart contract for achieving fairness.

(enough to transfer 128 KiB of data) takes ≈ 89 seconds for exponential ElGamal (vs. ≈ 5 seconds for Paillier) on a consumer laptop. We foresee several venues for optimizing our proof of concept implementation. The bulk of the cost (≈ 40%) for ElGamal is due to generating range-proofs which can be amortized when serving multiple clients. Our exponential ElGamal-based protocol has a constant-size proof (9 G and 2 F_p elements), but we split the plaintexts into k smaller chunks to enable efficient decryption, resulting in a k-factor blowup in the ciphertext size. On the other hand, the Paillier-based protocol has linear-sized proofs in the number of exchanged BLS12-381 field elements. Overall, we observe a ≈ 10× bandwidth overhead in our ElGamal protocol (vs. ≈ 50× for Paillier), that is, the size of the ciphertexts and proofs compared solely to the size of the exchanged data. Despite of this, interestingly, both the Paillier prover and verifier are concretely more efficient than the ElGamal prover and verifier, *cf.* Figure 6. For more benchmarks, *cf.* Section 6.

Outline. The rest of this paper is organized as follows. We review the pertinent literature in Section 2. We introduce the notion of verifiable encryption under committed key (VECK) in Section 3. In Section 4, we introduce the syntax, security, and privacy requirements of blockchain Fair Data Exchange (FDE) protocols, and construct secure FDE protocol instances on Ethereum and Bitcoin using VECK protocol as a black-box. In Section 5, we provide two constructions for secure FDE protocols: based on exponential ElGamal and based on Paillier encryptions. We provide an extensive performance evaluation of our implementation in Section 6. In Section 7, we introduce a multi-client FDE protocol where, via preprocessing, we amortize the server's computation cost to serve multiple clients We conclude our paper with some discussions, possible extensions, and open problems in Section 8.

2 RELATED WORK

It has been long known that fair exchange without a trusted third party (TTP) is impossible [54]. Recently, with the development of blockchains as reliable trusted third parties, fair exchange protocols have received renewed attention [6, 30, 44, 46]. These protocols typically resolve a witness-selling problem: the buyer is willing to offer D coins for the witness value x (*e.g.*, factorization of a modulus) such that $f(x)$ = true. The protocol then goes as follows: the buyer locks D coins, and only if the seller provides x to the buyer does the seller get these coins; if no x is provided, the buyer gets its money back after a time-out. This protocol effectively boils down to exchanging a signature for a witness, and a more general version of it is tasked with fairly exchanging two witnesses (*e.g.*, two signatures or two keys, or a signature for a key, etc.). Such protocols roughly fall into two categories: optimistic (*e.g.*, [3]) and atomic. The former uses a TTP for dispute resolution, where one party encrypts its signature under the TTP's public key and sends it to the other party. TTP then helps the parties complete the exchange in the event one of the parties aborts or cheats. However, in these works, the trusted party is assumed to handle secrets which is not translatable to the general blockchain setting. In an atomic exchange, the TTP holds the coins and sends them to the seller if the given witness value x satisfies a statement, *i.e.*, $f(x)$ = true. In these schemes, the TTP need not store any secret information.

In **FairSwap** [30], Dziembowski et al. develop a general protocol for exchanging the witness, x, of an NP statement ϕ for a signature (or a payment). The server encrypts the input x of the NP statement and the wires of the circuit for evaluating the statement $\phi(x)$, commits to all of these encryptions in a Merkle tree, and submits the Merkle root to a smart contract. It also submits a Merkle root committing to the circuit ϕ. The client then locks the payment to the server in the smart contract, after which the server submits the decryption key. A complaint period allows the client to succinctly prove to the contract the inconsistency of the data, in which case the client gets its money back. Our solution avoids the complaint period and allows the exchange to happen at network speed. A follow-up work, **OptiSwap** [31], improves the performance of the protocol for the optimistic case when both parties behave honestly by introducing interaction to the dispute resolution process.

More efficient protocols have been built for a special case of the problem above, aiming at exchanging the data underneath the client's commitment for a signature (i.e. a payment). In **File-Bounty** [46], the commitment to the data $(M_1, M_2, \ldots, M_N)$ is assumed to be an application of an iterated hash function (*e.g.*, SHA256 based on Merkle-Damgard paradigm):

$$C = h(h(\ldots h(h(H_N, M_N), M_{N-1}) \ldots), M_1).$$

The data is transferred chunk-by-chunk from M_1 to M_N, and each chunk M_i is accompanied by a value H_i with $H_1 := C$ and it is checked that $h(H_{i+1}, M_i) = H_i$. In case any one of the parties cheats or disappears, the dispute resolution is done on a blockchain, where zkSNARKs are used in order to hide the data. If the server or the client stops executing the protocol in the middle, then at worst, the client receives a small chunk that it did not pay for, or the server is paid a small amount for a small chunk it did not provide. In both cases, the loss can be tolerated since the chunk and the corresponding payment are both small. This model, however, only works if the client's utility in receiving a portion of the data is proportional to its size, and it is not suitable for scenarios where the client is only interested in receiving the whole data. In these cases, the blockchain can be used to help participants complete the exchange, albeit at the expense of privacy concerns and higher costs. Our work mitigates these issues by employing efficient verifiable encryption protocols.

He et al. [44] provide an FDE protocol called **FairDownload**, where a Merkle root hash of the data is published on-chain. The data is the leaves of the Merkle tree. The client and the server exchange the data chunk by chunk without any consistency proofs with respect to the Merkle root hash. However, the exchanged chunks are signed by the server. Therefore, if there is any dispute between the client and the server, the client can prove the misbehavior of the server to the on-chain contract with a $O(\log k)$-sized Merkle-inclusion proof. Linus [49] also designs a similar scheme called **BitStream** with optimistic dispute resolution on Bitcoin with the help of Merkle proofs. The $O(\log k)$ cost of these schemes is in contrast with our constant-sized on-chain communication cost. Interestingly, He et al. define novel notions of fairness (*e.g.*, delivery fairness) and show that their protocol satisfies them.

Finally, **Bitcoin zk-Contingent-Payments** introduced as a concept by Gregory Maxwell in 2011 [51] aims to solve a fair exchange problem for the Bitcoin blockchain limited in scripting capabilities. However, concrete instantiations were shown to be insecure [18, 38].

In contrast to the previous work, our protocol has a minimum number of rounds and entirely avoids a dispute resolution phase. See Table 1 for a more detailed comparison. Our protocol also supports selective download, where a subset of the data is requested, and it is friendly to data-dispersal protocols which utilize erasure coding (*e.g.*, Danksharding [29]).

Our protocol builds a variant of **verifiable encryption** (VE) for committed values. VE allows an encryptor to prove an NP-relation about a plaintext encrypted under a *public key* encryption scheme. It was first introduced by Stadler [66] for discrete logarithms and later generalized by [4] for the fair exchange of signatures. A related problem of practical encryption for *discrete-logarithm* values was solved by Camenish and Shoup [16] using a CCA-secure encryption scheme based on Paillier's Decision Composite Residuosity (DCR) assumption [55], with application, among others, to the fair exchange of Schnorr or DSS signatures. This was the first scheme to avoid the expensive "cut-and-choose" paradigm adopted by the earlier works [4, 66].

Whereas VE can be instantiated generically using zero-knowledge (zk)SNARKs, this requires the inclusion of the encryption as part of the SNARK relation, with potential effects on efficiency. In this context, the LegoSNARK framework allows proving relations satisfied by a witness with respect to an existing commitment, thus combines the commitment and SNARK akin to lego pieces [17]. SAVER extends this idea to encryptions of the witness by allowing SNARK proofs without including encryption in the SNARK circuit [48].

Unlike VE, VECK allows the use of symmetric key encryption. In fact, VECK is generically realizable using symmetric encryption and NIZKs, as we discuss in Section 8. Morever, as our scheme uses a fresh key to encrypt the data in each request, unlike [16], we do not need CCA-security, and simpler, more efficient CPA-secure schemes such as ElGamal or Paillier would suffice. We explore both variants and demonstrate their efficiency in this work.

A related construction to VE is commitment consistent encryption (CCE), which is a public key encryption scheme with the ability to generate a commitment to the encrypted message (with the public key) and to subsequently open the commitment (with the secret key) [26]. CCEs were developed to provide universally verifiable voting schemes with perfectly private audit trail; so that the election results can be verified via the audit data while preserving voter privacy even after key leakages [26, 57]. Unlike CCEs, VECKs enable verifiable encryption of subsets of the messages under a vector or polynomial commitment.

...[1]...

3 VERIFIABLE ENCRYPTION UNDER COMMITTED KEY (VECK)

A *verifiable encryption under committed key* (VECK) is a scheme with the following functionality: given a commitment to the data, it allows to encrypt the data (or a pre-specified function of the data). The encryption outputs a verification key, a ciphertext and a zero-knowledge proof of correct encryption. It satisfies correctness, soundness and zero-knowledge. Correctness guarantees that the decryption key corresponding to the verification key decrypts the original data (or a pre-specified function of it). Thus, the verification key can be viewed as a commitment to the decryption key. Soundness guarantees that no polynomial-time adversary can generate a convincing proof about an incorrect encryption without breaking the underlying assumptions, or the security of the commitment schemes. Zero-knowledge guarantees that the ciphertext, verification key and the proof leak no information that enables the recovery of the underlying data, therefore, the data would remain private until the decryption key is revealed.

Although such functionality can be generically achieved using public-key encryption and generic SNARKs, we show that building it holistically using tailored one-time encryption and proofs results in a simpler and more elegant stand-alone construction.

VECK allows us to reduce the problem of fair data exchange to a problem of fair exchange of decryption key for a payment, as we show in Section 4. The latter exchange can be done fairly through a blockchain, since the validity of the decryption key can be verified against the verification key using a blockchain smart-contract.

Preliminaries. We let $\lambda \in \mathsf{N}$ denote the security parameter. A non-negative function $\sigma(\lambda)$ is called *negligible* if for every polynomial $p(\lambda)$ it holds that $\sigma(\lambda) \leq 1/p(\lambda)$ for all sufficiently large $\lambda \in \mathsf{N}$. For a random variable x we denote by $x \leftarrow_R X$ the process of sampling a value x from the set X uniformly at random.

Definition 3.1 (verifiable encryption under committed key (VECK)). *Let* $(\textsc{Setup}, \textsc{Commit})$ *be a non-interactive binding commitment scheme, where* $\textsc{Setup}(1^\lambda) \to \mathsf{crs}$ *generates a public common-reference string, and* $\textsc{Commit}(\mathsf{crs}, w \in \mathcal{W}) \to C_w \in C$ *generates a commitment. A non-interactive VECK scheme for a class functions* $\mathcal{F} = \{F : \mathcal{W} \to \mathcal{V}\}$ *is a tuple of algorithms* $\Pi_{\mathcal{F}} = (\textsc{Gen}, \textsc{Enc}, \textsc{Ver}_{\mathsf{ct}}, \textsc{Ver}_{\mathsf{key}}, \textsc{Dec})$*:*

- $\textsc{Gen}(\mathsf{crs}) \to \mathsf{pp}$*: Probabilistic polynomial-time algorithm that takes as input the* crs *generated by the setup of the commitment scheme and outputs parameters for the system, as well as the description of appropriate spaces. The parameters* pp *are implicitly taken by all the following algorithms, we omit them where it is clear from the context.*
- $\textsc{Enc}(\mathsf{pp}, F, C_w, w) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$*: Probabilistic polynomial-time algorithm, run by the server, It takes in the function* F*, the commitment to* w *and the* w *itself, and outputs a verification key* vk*, a decryption key* sk*, an encryption* ct *of* $F(w)$ *and a proof* π*.*
- $\textsc{Ver}_{\mathsf{ct}}(\mathsf{pp}, F, C_w, \mathsf{vk}, \mathsf{ct}, \pi) \to 1/0$*: A deterministic polynomial-time algorithm run by the client that outputs accept or reject.*
- $\textsc{Ver}_{\mathsf{key}}(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}) \to 1/0$[3]*: A deterministic polynomial-time algorithm run by the blockchain or a trusted third party that checks the validity of the secret key.*
- $\textsc{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to v/\perp$*: A deterministic polynomial-time algorithm run by the client, it outputs a value (such as an evaluation of* F *on* w*) or* $\perp$*.*

A VECK scheme satisfies the following properties:

Correctness: Verifications for honestly generated encryption succeed: $\forall w \in \mathcal{W}, \forall F \in \mathcal{F}$*, the following event holds with probability* 1*:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ \textsc{Ver}_{\mathsf{ct}}(F, C_w, \mathsf{vk}, \mathsf{ct}, \pi) = 1 \wedge & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ \textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1 & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ & (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi) \leftarrow \textsc{Enc}(F, C_w, w) \end{array}\right]$$

Soundness: No probabilistic polynomial time adversary can generate sk, vk, ct *and* π *such that verification succeeds, yet decryption does not output a valid value. Namely,* $\forall\, w \in \mathcal{W}, \forall F \in \mathcal{F}$*, for any PPT algorithm* $\mathcal{A}$*, there exists a negligible function* $\nu(\cdot)$ *such that the following is less than* $\nu(\lambda)$*:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ \textsc{Ver}_{\mathsf{ct}}(F, C_w, \mathsf{vk}, \mathsf{ct}, \pi) = 1 \wedge & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ \textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1 \wedge & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ y \neq F(w) & (\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi) \leftarrow \mathcal{A}(\mathsf{pp}, F, C_w) \\ & y \leftarrow \textsc{Dec}(F, \mathsf{sk}, \mathsf{ct}) \end{array}\right]$$

We note that for F *that computes identity, knowledge extraction (knowledge-soundness) is implicit in the definition of security and is given by the decryption,* i.e., *a valid* w *can be extracted from the adversary that convinces the verifiers.*

Computational Zero-Knowledge: The ciphertext and the proof leak no additional information about the witness. For any PPT algorithm $\mathcal{A}$*, there exists a PPT simulator* Sim *such that there is a negligible function* $\mu(\cdot)$*, s.t. for all* $w \in \mathcal{W}, \forall F \in \mathcal{F}$ *the following probability is less than* $1/2 + \mu(\lambda)$*:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ \mathcal{A}(\mathsf{pp}, F, C_w, \mathsf{vk}_b, \mathsf{ct}_b, \pi_b) = b & (\mathsf{vk}_0, \mathsf{sk}_0, \mathsf{ct}_0, \pi_0) \\ & \qquad \textsc{Enc}(\mathsf{pp}, F, C_w, w) \\ & (\mathsf{vk}_1, \mathsf{ct}_1, \pi_1) \\ & \qquad \mathsf{Sim}(\mathsf{pp}, F, C_w) \\ & b \leftarrow_R \{0, 1\} \end{array}\right]$$

Zero-knowledge property can also be statistical instead of computational, where instead of having the algorithm $\mathcal{A}$ distinguishing the real $(\mathsf{pp}, C_w, \mathsf{vk}_0, \mathsf{ct}_0, \pi_0)$ and the simulated $(\mathsf{pp}, C_w, \mathsf{vk}_1, \mathsf{ct}_1, \pi_1)$ distribution as above, we would say that they are *statistically* indistinguishable. Some of our constructions satisfy this stronger notion of zero-knowledge.

Note that VECK allows the use of symmetric key encryption and can be built generically using symmetric encryption and NIZKs as we discuss in Section 8. Likewise, it can be built using public key encryption, but the possibility of using symmetric key encryption opens up a prospect for using more efficient schemes.

For the FDE application, we will explore a polynomial commitment scheme, where the VECK function F is the evaluations of a given degree-ℓ polynomial: $F(\phi) = \{\phi(i)\}_{i \in [\ell]}$, where $\phi(X)$ is a polynomial of degree ℓ, and more generically F computes the subsets of evaluations: $F_S(\phi) = \{\phi(i)\}_{i \in S}$. In practice, instead of $i \in \{0, 1, \ldots, \ell\}$, an FFT-friendly set (and its subsets) can be used for efficiency: $\{1, \omega, \omega^2, \ldots, \omega^\ell\}$, where $\omega \in \mathsf{F}_p$ is a primitive $(\ell + 1)$-th root of unity.

Given our choice of F, we briefly recall polynomial commitments and their properties.

Polynomial Commitment schemes allow committing to univariate polynomials of degree at most ℓ over F_p and are comprised of the following algorithms, where Setup is randomized and the rest are deterministic (although Commit might also be randomized, but this case will not be our focus here):

- $\textsc{Setup}(1^\lambda, n) \to \mathsf{crs}$: generates public parameters for committing to polynomials of degree at most n.

[3]In our constructions, the key verification will not need full public parameters.

- Commit crs, $\phi(X)$ $\to C$: deterministically computes the commitment C to the polynomial $\phi(X) \in \mathsf{F}_p^{\leq n}[X]$ of degree not greater than n.
- VerifyPoly crs, $\phi(X), C$ $\to 0/1$: outputs 1 if it holds that Commit crs, $\phi(X)$ $= C$, and outputs 0 otherwise.
- Open(crs, $i, \phi(X)) \to \pi$: outputs a proof π for the fact that $\phi(X)$ evaluates to $\phi(i)$ at index i.
- VerifyEval crs, $C, i, \phi(i), \pi$ $\to 0/1$: verifies the proof.
- BatchOpen(crs, $S = (i_1, \ldots, i_k), \phi(X)) \to \pi$: outputs a proof for multiple evaluations of $\phi(X)$ at indices S.
- BatchVerify(crs, $C, (m_{i_1}, \ldots, m_{i_k}), (i_1, \ldots, i_k), \pi) \to 0/1$ verifies the batch proof.

A secure polynomial commitment scheme satisfies the following properties (for the full statements, please refer to *e.g.*, [47]):

Correctness: Honestly generated commitment and proofs verify correctly.

Polynomial Binding: No PPT adversary can generate a commitment C and two *different* polynomials $\phi(X), \phi'(X) \in \mathsf{F}_p^{\leq n}[X] \land \phi(X) < \phi'(X)$, such that the commitment verifies against both of them correctly, *i.e.*, they generate the same commitment: $C =$ Commit crs, $\phi(X)$ $=$ Commit crs, $\phi'(X)$ [4].

Evaluation Binding: No PPT adversary can generate a commitment C and two *different* evaluations $\phi(i) < \phi'(i)$ on the same point i with proofs π, π' that would verify correctly.

Polynomial commitments can be viewed as a special case of vector commitments, where a data vector $\mathbf{m} = (m_0, \ldots, m_\ell) \in \mathsf{F}_p^{\ell+1}$ is mapped to a polynomial $\phi(X) \in \mathsf{F}_p^{\ell}(X)$ of degree ℓ, s.t. $\forall i \in [\ell] : \phi(i) = m_i$ [20], BatchOpen then allows to generate subvector-opening proofs.

4 APPLICATION OF VECK: FAIR BLOCKCHAIN DATA EXCHANGE PROTOCOLS

We introduce the syntax of fair blockchain data exchange (FDE) protocols and instantiate them by combining a VECK scheme with smart contracts on Ethereum and adaptor signatures on Bitcoin.

We define a transparent payment environment $\mathcal{E}$ as a *trusted third party* that holds money under addresses belonging to the other parties. It can transfer money from one party's address to another but requires a message authorizing the transaction with the sender's signature. It is transparent in the sense that any message sent to $\mathcal{E}$ eventually becomes visible to all other parties.

4.1 The FDE Protocol Syntax and Properties

Definition 4.1 (Blockchain Data Exchange Protocols). *A FDE protocol consists of two PPT algorithms and two protocols between a client C and a server $\mathcal{S}$ involving a transparent payment environment $\mathcal{E}$:*

- *FDE.Setup*$(1^\lambda) \to$ pp. *Probabilistic polynomial-time algorithm that outputs the public parameters for the system (*e.g.*, the description of appropriate spaces). The public parameters are implicitly taken by all the following algorithms and protocols; we omit them for brevity.*
- *FDE.Com*$\langle C(\text{data}), \mathcal{S}() \rangle \to \langle C(\text{com}), \mathcal{S}(\text{data}) \rangle$. *The parties C and $\mathcal{S}$ engage in a non-interactive protocol, where C commits to* data $\in \{0,1\}^{\ell \cdot \lambda}$ *consisting of ℓ blocks of data and stores the commitment* com $\in \{0,1\}^\lambda$. *C then sends* data *to $\mathcal{S}$.*
- *FDE.Vrfy*(data, com) $\to \{0,1\}$. *Given* data *and* com, *the server $\mathcal{S}$ verifies the correctness of the commitment* com *with respect to the data* data.
- *FDE.Exc*$\langle C(\text{com}, \text{tk}), \mathcal{S}(\text{com}, \text{data}) \rangle \to \langle C(\text{data}), \mathcal{S}(\text{tk}) \rangle$. *The parties C and $\mathcal{S}$ engage in an interactive protocol to exchange the* data *held by $\mathcal{S}$ and the tokens* tk *held by C over $\mathcal{E}$.*

FDE protocols satisfy the following properties.

Definition 4.2 (FDE Correctness). *If the client C and server $\mathcal{S}$ are honest, given* pp $\leftarrow$ *FDE.Setup*(1^λ) *and* $\langle C(\text{com}), \mathcal{S}(\text{data}) \rangle \leftarrow$ *FDE.Com*$\langle C(\text{data}), \mathcal{S}() \rangle$*; with probability* 1,

$$\langle C(\text{data}), \mathcal{S}(p) \rangle \leftarrow \textit{FDE.Exc}\langle C(\text{com}, p), \mathcal{S}(\text{com}, \text{data}) \rangle,$$

i.e., *C receives the data, and $\mathcal{S}$ receives* tk *tokens.*

The client-fairness property guarantees that the server cannot receive any payment if the client does not obtain the data, and server-fairness guarantees that the client cannot learn anything about the data without paying the server.

Definition 4.3 (Client-Fairness). *Given an honest client C, for any* data *from an appropriate space, for all PPT $\mathcal{S}^*$, the following probability that C does not receive the whole data while $\mathcal{S}^*$ receives a positive payment*[5] *is negligible in λ:*

$$\Pr\left[\begin{array}{r|r} & \text{pp} \leftarrow \textit{FDE.Setup}(1^\lambda) \\ \textit{FDE.Vrfy}(\text{data}',\text{com})=0 & \langle C(\text{com}),\mathcal{S}(\text{data})\rangle \leftarrow \textit{FDE.Com}\langle C(\text{data}),\mathcal{S}()\rangle \\ \wedge\ \text{tk}' \text{¡}\ 0 & \textit{Inputs: } C(\text{com,tk}),\mathcal{S}^*(\text{com,data}) \\ & \langle C(\text{data}'),\mathcal{S}^*(\text{tk}')\rangle \leftarrow \textit{FDE.Exc}\langle C,\mathcal{S}^*\rangle \end{array}\right]$$

Definition 4.4 (Server-Fairness). *Given an honest server $\mathcal{S}$, for all PPT C^*, there exists a PPT simulator* Sim^{C^*} *with oracle access to C^* s.t. for all possible values* data, *the following probability is less than $\frac{1}{2} + \mu(\lambda)$ for a negligible function $\mu(\cdot)$:*

$$\Pr\left[\begin{array}{r|r} & \text{pp} \leftarrow \textit{FDE.Setup}(1^\lambda) \\ & \langle C(\text{com}),\mathcal{S}(\text{data})\rangle \leftarrow \textit{FDE.Com}\langle C(\text{data}),\mathcal{S}()\rangle \\ & \textit{Inputs: } C^*(\text{com,tk}),\mathcal{S}(\text{com,data}) \\ C^*(\text{com}, \alpha_b) = b & \alpha_0 \leftarrow \text{tr}(\langle C^*,\mathcal{S}\rangle \leftarrow \textit{FDE.Exc}\langle C^*,\mathcal{S}\rangle) \\ & \textit{Outputs: } C^*(\text{data}'),\mathcal{S}(\text{tk}')\ \textit{s.t.}\ \text{tk}' \text{Ÿ} \text{tk} \\ & \alpha_1 \leftarrow \text{Sim}^{C^*}(\text{pp,com,tk}) \\ & b \xleftarrow{R} \{0,1\} \end{array}\right]$$

Here, tr *denotes the interactive protocol's transcript. It includes all public inputs (the public parameters* pp *and commitment* com*), the exchanged bits, the payment* tk′ *server gets, and the data* data′ *client C^* obtains as outputs. In other words, C^* does not learn anything about the data other than FDE.Com*(data) = com *unless $\mathcal{S}$ receives a payment of* tk *tokens.*

Efficiency requirement. The asymptotic communication complexity between the server and client in the FDE.Exc protocol is linear in ℓ, the number of data blocks, as the client has to decrypt every data block. Therefore, we will minimize the communication complexity and size of the overhead of the FDE.Exc protocol *on top of the exchanged data*. We also minimize the amount of amortized computation made by the parties as part of the FDE.Exc protocol.

[4] We only work with polynomials of degree ℓ by assuming that the size of the committed data is known.

[5] We assume that the server $\mathcal{S}^*$ cannot ever receive payment greater than tk.

4.2 The FDE Protocol on Ethereum

Consider a client C interested in the output of a function $F(.)$ applied on a sequence of data (denoted by data $= (m_1, \dots, m_\ell)$) attested by the polynomial (or vector) commitment com. In return, C offers some payment tk to the server $\mathcal{S}$ that stores the data. To facilitate this exchange, the FDE protocol uses a VECK scheme and a smart contract on Ethereum.

The relation used by the VECK scheme involves a polynomial (or vector) commitment, and FDE.$\textsc{Setup}(1^\lambda)$, FDE.$\textsc{Com}\langle C(\mathsf{data}), \mathcal{S}()\rangle$ and FDE.$\textsc{Vrfy}(\mathsf{pp}, \mathsf{data}, \mathsf{com})$ are instantiated with the $\textsc{Setup}$ and $\textsc{Commit}$ algorithms for this polynomial commitment scheme (*cf.* Section 3). Let $\phi(.)$ be the degree ℓ polynomial that takes the values of the data points m_i at inputs $i \in [\ell]$: $\phi(i) = m_i$. Then; FDE.$\textsc{Setup}(1^\lambda) = \textsc{Setup}(1^\lambda, \ell) \to \mathsf{pp}$.

FDE.$\textsc{Com}\langle C(\mathsf{data}), \mathcal{S}()\rangle$: C runs $\textsc{Commit}(\mathsf{pp}, \phi(X)) \to \mathsf{com}$ and sends com to $\mathcal{S}$.

FDE.$\textsc{Vrfy}(\mathsf{data}, \mathsf{com}) \to [\textsc{Commit}(\mathsf{pp}, \phi(X)) = \mathsf{com}]$.

The protocol FDE.$\textsc{Exc}\langle C(\mathsf{com}, p), \mathcal{S}(\mathsf{com}, \mathsf{data})\rangle$ proceeds as:

1) The client C creates a smart contract called the *bonding contract* on Ethereum that allows the spending of tk tokens to only the address of the server $\mathcal{S}$ before a timelock expires. After it is deployed, the contract takes as input a verification key vk, tokens of amount tk and sends the tokens to $\mathcal{S}$ only if it receives the correct decryption key sk such that $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$. After the timelock expires, the tokens are returned to C.

2) The server $\mathcal{S}$ encrypts the data using the VECK scheme for the function F and the polynomial commitment scheme used by the FDE protocol (Section 3): $\textsc{Enc}(\mathsf{com}, \mathsf{data}) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$. It then posts vk to the contract and sends ct off-chain (along with the associated proof π) to the client C.

3) The client verifies the ciphertext ct: $\textsc{Ver}_{\mathsf{ct}}(\mathsf{com}, \mathsf{vk}, \mathsf{ct}, \pi) \to 0/1$. It then locks tk tokens in the contract.

4) The server $\mathcal{S}$ checks if C has locked the correct amount (tk) of tokens. If the timelock has not expired yet, and there are tk tokens locked in the contract, $\mathcal{S}$ posts the decryption key sk to the contract. The contract sends the tk tokens to $\mathcal{S}$ if $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$.

5) The client reads sk from the contract. Using sk, it decrypts ct and obtains the data: $\textsc{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{data}$.

4.3 FDE Protocol on Bitcoin

4.3.1 Preliminaries. In the absence of smart contracts, we design a bonding contract on Bitcoin using adaptor signatures. Consider a signature scheme $\Sigma = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify})$ and a hard relation $\mathcal{R}$. Let $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \textsc{KeyGen}(1^\lambda)$ and $(Y, y) \in \mathcal{R}$. An adaptor signature scheme Sig with respect to Σ and Y consists of the following four algorithms: $\hat{\sigma} \leftarrow \textsc{pSign}(\mathsf{sk}_\sigma, m, Y)$, $b \leftarrow \textsc{pVerify}(\mathsf{pk}_\sigma, m, \hat{\sigma}, Y)$, $\sigma \leftarrow \textsc{Adapt}(\mathsf{pk}_\sigma, \hat{\sigma}, y)$ and $y \leftarrow \textsc{Extract}(\sigma, \hat{\sigma}, Y)$. Here, $\hat{\sigma}$ is a presignature, b denotes the output of the verification for $\hat{\sigma}$, and σ denotes the adapted signature (that verifies against the public key $\mathsf{pk}_\sigma \cdot Y$) from which y can be extracted. For the FDE protocol on Bitcoin, we use adaptor signatures based on Schnorr signatures. For details on adaptor signatures, *cf.* Appendix A.5 and [27].

4.3.2 Protocol. Algorithms FDE.$\textsc{Setup}(1^\lambda)$, FDE.$\textsc{Vrfy}(\mathsf{data}, \mathsf{com})$ and the protocol FDE.$\textsc{Com}\langle C(\mathsf{data}), \mathcal{S}()\rangle$ are instantiated as in Section 4.2. The adaptor signature scheme used by the protocol is based on the relation satisfied by $y = \mathsf{sk}$ and $Y = \mathsf{vk}$ with language $L_{\mathsf{key}} = \{\mathsf{vk} | \textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1\}$. The FDE protocol proceeds as follows:

1) This is the same step as step (2) in Section 4.2, except that $\mathcal{S}$ sends both the verification key vk and the ciphertext ct to C.

2) The client verifies the ciphertext ct: $\textsc{Ver}_{\mathsf{ct}}(\mathsf{com}, \mathsf{vk}, \mathsf{ct}, \pi) \to 1/0$. Then, it creates a bonding contract on Bitcoin that does the following: Before a timelock expires, it allows the spending of tk tokens by a transaction with two signatures: one must verify with respect to $\mathcal{S}$'s public key $\mathsf{pk}_{\mathcal{S}}$, and the other must verify with respect to the public key of the adaptor signature, *i.e.*, $\mathsf{pk}_\sigma \cdot \mathsf{vk}$. After the expiry, the tokens can be spent to any address and returned to C with a signature that is created with C's signing key sk_C and that verifies under C's public key pk_C.

3) If the verification above succeeds, C also sends a pre-signature on a transaction tx: $\hat{\sigma} \leftarrow \textsc{pSign}(\mathsf{sk}_\sigma, \mathsf{tx}, \mathsf{vk})$, where tx transfers the tokens in the bonding contract to $\mathcal{S}$'s address.

4) If the timelock has not expired yet, there are tk tokens locked in a correctly structured bonding contract and $\hat{\sigma}$ verifies, namely, $\textsc{pVerify}(\mathsf{pk}_\sigma, \mathsf{tx}, \hat{\sigma}, \mathsf{vk}) = 1$, the server *adapts* the pre-signature using the decryption key sk to get a full signature σ on the transaction tx: $\sigma \leftarrow \textsc{Adapt}(\mathsf{pk}_\sigma, \hat{\sigma}, \mathsf{sk})$. It also signs tx with its signing key $\mathsf{sk}_{\mathcal{S}}$ corresponding to $\mathsf{pk}_{\mathcal{S}}$ and posts tx to Bitcoin with both signatures.

5) The client extracts sk from σ: $\mathsf{sk} \leftarrow \textsc{Extract}(\sigma, \hat{\sigma}, \mathsf{vk})$. Using sk, it decrypts ct and retrieves the data: $\textsc{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{data}$.

Note that the existence of both the adaptor signature and $\mathcal{S}$'s signature on tx prevents C from frontrunning the server. If tx were signed by only the adaptor signature, upon receiving the adaptor signature from the mempool, C could have extracted the decryption key sk, and created another transaction, signed by the adapted signature and spending the tokens to C's address, before $\mathcal{S}$'s transaction is confirmed on Bitcoin.

4.4 Security Proof

Theorem 4.1. *Suppose the VECK scheme satisfies correctness, security and computational zero-knowledge, and Ethereum (Bitcoin) satisfies security with some finite latency. Then, for a sufficiently long timelock period, the FDE protocol on Ethereum (Bitcoin) satisfies correctness, client-fairness, and server-fairness.*

The proof is given in Appendix C.1 and follows directly from the security of the VECK scheme and the security of Ethereum / Bitcoin.

5 VECK CONSTRUCTIONS

In this section, we design efficient instantiations of VECK schemes for selective openings of KZG polynomial commitments (Section 1). We start with preliminaries and recall KZG polynomial commitments. Section 5.1 describes a VECK protocol based on the Decisional Diffie-Hellman (DDH) assumption and uses a symmetric variant of exponential ElGamal encryption, whereas Section 5.2 describes a scheme based on the Decisional Composite Residuosity (DCR) assumption and uses Paillier encryption.

Preliminaries.

For a prime p, we use F_p to denote the finite field of size p. A bilinear operation $e : \mathsf{G}_1 \times \mathsf{G}_2 \rightarrow \mathsf{G}_T$ defined over three elliptic curve groups $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_T$ of prime order p and generators g_1, g_2, g_T, satisfies the following properties: for any $a, b \in \mathsf{F}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ and generators are respected: $e(g_1, g_2) = g_T$. Denote by $\mathsf{F}_p[X]$ the ring of polynomials over F_p, and by $\mathsf{F}_p^{\ell}[X] \subset \mathsf{F}_p[X]$ the group of polynomials of degree ℓ, by $\mathsf{F}_p^{\leq \ell}[X] \subset \mathsf{F}_p^{\ell}[X]$ – polynomials of degree at most ℓ. The Lagrange basis polynomial for a given set $S \subseteq \mathsf{F}_p$ and $x \in S$, denoted by $L_{x,S}(X) \in \mathsf{F}_p[X]$, is defined as follows:

$$L_{x,S}(X) = \prod_{\substack{i \in S \\ i \neq x}} \frac{X - i}{x - i}.$$

$L_{x,S}(X)$ has degree $|S| - 1$ and can alternatively be uniquely defined by its $|S|$ evaluations: $L_{x,S}(x) = 1$ and $L_{x,S}(i) = 0$ for $\forall i \in S, i \neq x$. When the set $S = \{0, 1, \ldots, \ell\}$ consists of $\ell + 1$ consequitive integers, we denote the Lagrange polynomial by $L_{x,\ell}(X) := L_{x,S}(X)$ For a polynomial $\phi(X)$, we let $\phi_S(X)$ denote the unique polynomial of degree at most $|S| - 1$ that agrees with $\phi(X)$ on the set S: $\forall i \in S, \phi(i) = \phi_S(i)$. It can be constructed as follows: $\phi_S(X) := \sum_{j \in S} \phi(j) L_{j,S}(X)$. For an integer B, we denote by $[B] = \{0, 1, \ldots, B\}$ the set of consecutive integers from 0 to B.

KZG Polynomial Commitments. The KZG [47] scheme commits to univariate polynomials $\phi(X) \in \mathsf{F}_p^{\leq n}[X]$ and works as follows:

Setup$(1^\lambda, n) \rightarrow \mathsf{crs}$: trusted setup that generates the group structure $\mathcal{G}$ comprised of elliptic curve groups: $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_T$ of order $p \geq 2^{2\lambda}$ with generators g_1, g_2, g_T respectively and bilinear pairing operation $e : \mathsf{G}_1 \times \mathsf{G}_2 \rightarrow \mathsf{G}_T$. It samples a uniformly random secret $\tau \leftarrow_R \mathsf{F}_p$ and computes the public parameters

$$\mathsf{crs} = (\mathcal{G}, \{g_1^{\tau^i}\}_{i=1}^n, \{g_2^{\tau^i}\}_{i=1}^n).$$

Such setup can also be run through an MPC ceremony [7, 12, 53].

Commit $(\mathsf{crs}, \phi(X)) \rightarrow C$: computes the commitment $C := g_1^{\phi(\tau)}$ using the public parameters crs and the coefficients of $\phi(X)$.

VerifyPoly $(\mathsf{crs}, \phi(X), C) \rightarrow 0/1$: outputs 1 if Commit $(\mathsf{crs}, \phi(X)) = C$, and outputs 0 - otherwise.

Open$(\mathsf{crs}, i, \phi(X)) \rightarrow \pi$: outputs the opening proof $\pi := g_1^{q(\tau)}$, where $q(X) := (\phi(X) - \phi(i))/(X - i)$ is a quotient polynomial, computed as the commitment using the public parameters, crs.

VerifyEval $(\mathsf{crs}, C, i, \phi(i), \pi) \rightarrow 0/1$: if $e(C/g_1^{\phi(i)}, g_2) = e(\pi, g_2^{\tau}/g_2^{i})$ outputs 1, otherwise, outputs 0.

BatchOpen$(\mathsf{crs}, S = (i_1, \ldots, i_k), \phi(X)) \rightarrow \pi$: outputs the proof $\pi := g_1^{q(\tau)}$, where $q(X)$ is a quotient polynomial defined as

$$q(X) = \frac{\phi(X) - \phi_I(X)}{\prod_{i \in S}(X - i)},$$

where $\phi_S(X) = \sum_{i \in S} \phi(i) L_{i,S}(X)$ is a polynomial of degree at most $k - 1$ that agrees with $\phi(X)$ on S [6].

[6]Feist and Khovratovich [32] provide optimizations for BatchOpen when the set S is a set of consecutive powers of the root of unity, and it is commonly used in practice.

BatchVerify$(\mathsf{crs}, C, (m_{i_1}, \ldots, m_{i_k}), (i_1, \ldots, i_k), \pi) \rightarrow 0/1$ accepts if the following holds:

$$e\left(C/g_1^{\phi_S(\tau)}, g_2\right) = e\left(\pi, g_2^{\prod_{j=1}^{k}(\tau - i_j)}\right)$$

Here, $\phi_S(X)$ is as defined above for BatchOpen.

The scheme described above is polynomial- and evaluation-binding provided that the t-BSDH assumption holds in $(\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_T)$.

5.1 ElGamal-based VECK for KZG Commitments

In this section, we present a VECK protocol based on the DDH assumption and prove its security in the Algebraic Group Model (AGM) [39] (*cf.* Appendix A.1.3). First, we build VECK protocols for the function defined from polynomials of degree $\ell \leq n$, *i.e.*, $\phi(X) \in \mathsf{F}_p^{\leq n}[X]$ (where n is the length of the crs and also an upper-bound on the polynomials that can be committed with this crs), to $\mathsf{F}_p^{\ell+1}$, that outputs their evaluations at the $\ell + 1$ points specified by the set $[\ell]$:

$$F_{[\ell]}^{\text{full-eval}}(\phi) = (\phi(0), \phi(1), \ldots, \phi(\ell)) \in \mathsf{F}_p^{\ell+1} \tag{5.1}$$

For instance, when $\phi(X) \in \mathsf{F}_p^{\ell}[X]$, *i.e.*, is of degree ℓ, $F_{[\ell]}^{\text{full-eval}}$ outputs $\ell + 1$ evaluations of $\phi(X)$ at the points in $[\ell]$. The function is index by full-eval since its output uniquely determines the input polynomial. We first show the protocol for polynomials where each evaluation is within a small range $\forall i \in [\ell] : \quad 0 \leq \phi(i) \leq \mathcal{B}$, and then we show how to generalize the protocol to arbitrary polynomials $\phi(X) \in \mathsf{F}_p^{\ell}[X]$.

The high-level intuition of our protocol is as follows. We use exponential ElGamal to encrypt the $\phi(i)$ values: $\forall i \in [\ell] : \mathsf{ct}_i := h_i^s \cdot g_1^{\phi(i)} \in \mathsf{G}_1$ with independent generators $g_1, \{h_i\}_{i=1}^{\ell}, h \in \mathsf{G}_1$, where the decryption key is $\mathsf{sk} = s \in \mathsf{F}_p$ and the verification key is $\mathsf{vk} = h^s$. Recall that in a VECK scheme, we want to prove that $\forall i \in [\ell], \mathsf{ct}_i$ indeed encrypts $\phi(i)$ for a secret polynomial $\phi(X) \in \mathsf{F}_p^{\ell}[X]$ KZG-committed by $C_\phi = g_1^{\phi(\tau)} \in \mathsf{G}_1$ for a trapdoor τ. For this purpose, a pseudo-random challenge $\alpha \in \mathsf{F}_p$ is sampled, and the polynomial commitment C_ϕ is opened in the exponent at α with a blinding factor $s(\tau - \alpha)$ to yield blinded $g_1^{\phi(\alpha)}$. The verifier in turn interpolates through the encryptions, combining them with Lagrange coefficients in order to get the ElGamal encryption of $\phi(\alpha)$. It then verifies that the value in the combined encryption matches the blinded opening. The full protocol is described in detail in Figure 2.

Theorem 5.1. *The protocol described in Figure 2 is a secure VECK in the random oracle and algebraic group models for function F defined in Equation* (5.1).

We give the proof of this Theorem in Appendix C.2.

Alternative approach. We note that there is an alternative, less efficient protocol that does not involve publishing C_α as part of the proof. The ciphertexts can be directly verified against the commitment by checking that the discrete logarithm of Q^* with respect to Q matches that of vk with respect to h, for Q defined as follows and $Q^* = Q^s$:

$$\prod_{i=0} e(\mathsf{ct}_i, g_2^{L_{i,\ell}(\tau)}) = Q^s \cdot e(C_\phi, g_2)$$

$$Q = \prod_{i=0} e(h_i, g_2^{L_{i,\ell}(\tau)})$$

However, this alternative approach requires ℓ computationally costly pairing operations and would be far less efficient for the client who does the ciphertext verification in our FDE protocol (for comparison, one pairing operation is 10x more expensive than one exponentiation in G_1 for the bn256 curve (see Table 15.1 of [11]), and multi-exponentiations can be done even faster).

Exponential ElGamal encryption only works for a small or low-entropy message space (*e.g.*, $\mathcal{M} = \{0,1\}^{32}$), as the decryption procedure outputs $g_1^m \in \mathsf{G}_1$, and obtaining the message $m \in \mathcal{M}$ requires the decryptor to brute-force the discrete logarithm of g_1^m to find m. Hence, we had to bound the evaluations of the polynomial. A common practice to adapt it to large messages is to split the message (*e.g.*, $\mathcal{M} = \{0,1\}^{256}$) into some k chunks of size $D = \log_2(|\mathcal{M}|)/k$ and encrypt these chunks separately, accompanying each encryption with a zero-knowledge range proof showing that the encrypted value is within the range $[0, \ldots, 2^D - 1]$, where D is such that it is efficient to brute force a discrete logarithm computation as in the decryption algorithm of Figure 2. We explain this approach in detail in Figure 3, and we show the parts of the protocol that need to be modified.

5.2 Paillier Encryption with KZG Commitments

The VECK based on exponential ElGamal has an inherent downside: while transmitting a message $m \in \mathcal{M}$, the ciphertext size and accompanying proofs are blown up by a factor of $\log_2(|M|)/D$. This section explores an alternative approach using Paillier encryption [55] to avoid the aforementioned ciphertext blow-up. Paillier encryption allows encrypting arbitrary messages $m \in \mathsf{Z}_N$ with a ciphertext-to-message length ratio of two: $|c|/|m| = 2$. However, we will be encrypting dlog values, so our ciphertext-to-message length ratio would be: $|c|/|m| = 2N/p$. We recall how Paillier encryption works in Appendix A.4, and in Figure 4, we show the VECK protocol that allows a server to prove in zero-knowledge that the Paillier ciphertexts $\{\mathsf{ct}_i\}_{i\in[\ell]}$ encrypt the evaluations $\{\phi(i)\}_{i=0}^{\ell}$ of a KZG committed polynomial ϕ.

We take inspiration from the Fouque and Stern construction [37] of a one-round distributed key generation protocol, where they show how to prove the recoverability of discrete logarithm values from the Paillier ciphertext.

The high-level intuition for our protocol (it is a Σ-protocol) is as follows. The verification key is $\mathsf{vk} = N$ and the secret key is the factorization of N. The prover encrypts the evaluations of ϕ with $\ell + 1$ Paillier ciphertexts: $\mathsf{ct}_i = (N+1)^{\phi(i)} U_i^N \mod N^2$ for $i \in [\ell]$, where $U_i \leftarrow_R \mathsf{Z}_N^*$. It then encrypts the evaluations of $\ell + 1$ randomly sampled values: $T_i = (N+1)^{r_i} S_i^N \mod N^2$ for $r_i \leftarrow_R [0, A)$ and $S_i \leftarrow_R \mathsf{Z}_N^*$ and generates a KZG commitment T to the polynomial with evaluations r_i. After computing a random challenge $c = H(\mathsf{vk}, \{\mathsf{ct}_i\}, \{T_i\}, T)$, it finds $W_i = S_i U_i^c \mod N^2$ and $z_i = r_i + c\phi(i) \in \mathsf{Z}$, and sends ct_i, z_i and W_i to the verifier. The verifier reconstructs T_i and T, and then checks $c = H(\mathsf{vk}, \{\mathsf{ct}_i\}, \{T_i\}, T)$. Note this protocol, for technical reasons, uses a crs that commits to Lagrange-basis polynomials instead of the more common monomial basis. It also uses an adaptation of standard discrete logarithm equality proofs to the groups of unequal order.

Next, we prove that the protocol described above and formalized by Figure 4 is a secure VECK protocol.

Theorem 5.2. *The protocol in Figure 4 is a secure VECK in the random oracle and algebraic group models for function F defined in Equation* (5.1).

We give the proof of this theorem in Appendix C.3.

The protocol in Figure 4 has a negligible probability of correctness failure. It can be modified to achieve perfect correctness by restarting the encryptor until all of the z values are in the range $[0, A)$. However, this process would result in a significant slowdown of the encryption. For practicality, we suggest the version with a negligible probability of correctness failure.

Concrete parameters: For an instantiation of this protocol at λ-bits security level (*e.g.*, $\lambda = 128$), we would set $B \approx 2^{2\lambda}$ to achieve collision resistance for the hash function at λ-bits security, and $p = 2^{2\lambda}$ for the hardness of the discrete logarithm problem to be at λ-bits security level in the elliptic curve groups. Then, for any practical vector length $n \ll 2^\lambda$, we get $A \geq 2^{6\lambda}$ and $N \geq 2^{8\lambda+2}$. Therefore, for $\lambda = 128$, the length of the modulus N should be at least 2050 bits, which is a reasonable size RSA modulus widely used in production today.

Optimizations: Note that computing $(N+1)^a \mod N^2 = 1 + aN \mod N^2$ is very cheap. Computing $U_i^N \mod N^2$ is expensive but can be done in advance. Decryption can be calculated twice: once $\mod p^2$ and once $\mod q^2$ instead of $\mod N^2$ by using $L_p(x) = (x-1)/p$ and $L_q(x) = (x-1)/q$ instead of normal $L(\cdot)$ respectively. These two "partial decryptions" can be combined into m using the Chinese Remainder Theorem.

5.3 VECK for subset openings of KZG commitments

Here we show how to build VECK for encrypting *subvectors* of evaluations of the committed polynomial, namely VECK for the function $F_S : \mathsf{F}_p^\ell[X] \to \mathsf{F}_p^{|S|}$, where $S \subseteq \mathsf{F}_p$, $|S| \leq \ell + 1$:

$$F_S(\phi) = \{\phi(i)\}_{i\in S} \tag{5.2}$$

In the previous sections we showed constructions for the case of $S = [\ell] = \{0, 1, \ldots, \ell\}$, we note that they trivially generalize to any arbitrary set S of the same size $|S| = \ell + 1$. We refer to such scheme as a VECK for the full opening function $F^{\text{full-eval}}$. We now show how to support smaller sets (compared to the degree of the committed polynomial, ℓ) using the full opening VECK as subroutine.

At a high level, we generate a polynomial $V_S(X) = \prod_{i\in S}(X - i)$ of degree $|S| - 1$ which vanishes on the set S, and we generate a polynomial $\phi_S(X) = \sum_{i\in S} \phi(i) L_{i,S}(X)$ of degree $|S| - 1$ that agrees with $\phi(X)$ on the set S. We next sample a random $t \leftarrow_R \mathsf{F}_p$ and create a blinded polynomial $\phi'_S(X) = \phi_S(X) + tV_S(X)$ of degree $|S| - 1$. This polynomial agrees with $\phi(X)$ on the set S. Therefore

Gen(crs) $\to$ pp**:** On input crs $= (\mathcal{G}, \{g_1^{\tau^i}\}_{i=1}^n, \{g_2^{\tau^i}\}_{i=1}^n)$, generate random group elements with unknown discrete logarithms $h_i \leftarrow_R \mathsf{G}_1, h \leftarrow_R \mathsf{G}_1$ for $i \in [n]$. In practice, these can be generated using a Hash-to-Curve [8] function $H_{\mathsf{curv}} : \mathsf{F}_p \to \mathsf{G}_1, h_i := H_{\mathsf{curv}}(i)$. Output pp $= (\mathsf{crs}, \{h_i\}_{i=0}^n, h)$.

Enc$(F^{\textbf{full-eval}}_{[\ell]}, C_\phi, \phi(X)) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$:

(1) Sample $s \leftarrow_R \mathsf{F}_p$, set sk $:= s$, vk $= h^s$.
(2) Compute the encryptions $\{\mathsf{ct}_i := h_i^s \cdot g_1^{\phi(i)}\}_{i=0}^{\ell}$.
(3) Generate a Fiat-Shamir pseudo-random challenge $\alpha := \mathsf{H}(C_\phi, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell, [\ell])$.[a]
(4) Compute $C_\alpha := g_1^{\phi(\alpha)+s(\tau-\alpha)}$.
(5) Compute the opening proof for α: $\pi_\alpha := \textsc{Open}(\mathsf{crs}, \alpha, \phi(X) - s(X-\alpha))$ (note that $\pi_\alpha = g_1^{(\phi(\tau)-\phi(\alpha))/(\tau-\alpha)-s}$).
(6) Compute π_{LIN} to prove that C_α is indeed of the form g_1 raised to a known power, multiplied by $g_1^{\tau-\alpha}$ raised to the secret s of the verification key, *i.e.*, we compute the proof for the relation $\mathcal{R}_{\mathsf{LIN}}$ as described in the Appendix A.2, Equation (A.4):

$$\mathcal{R}_{\mathsf{LIN}} = \begin{array}{lll} g_{1,1} = g_1 & g_{1,2} = g_1^{\tau-\alpha} & u_1 = C_\alpha \\ g_{2,1} = 1 & g_{2,2} = h & u_2 = \mathsf{vk} \end{array} ; \quad \begin{array}{l} x_1 = \phi(\alpha) \\ x_2 = s \end{array} \quad g_{1,1}^{x_1} g_{1,2}^{x_2} = u_1 \wedge g_{2,1}^{x_1} g_{2,2}^{x_2} = u_2 \ .$$

(7) Compute $Q := \prod_{i=0}^{\ell} h_i^{L_{i,\ell}(\alpha)} \cdot g_1^{-(\tau-\alpha)}$.
(8) Compute discrete logarithm equality π_{DLeq} with respect to (Q, Q^s, h, vk) for the witness s, *i.e.*, we compute the proof for the relation $\mathcal{R}_{\mathsf{DLeq}}$ as described in the Appendix A.2, Equation (A.3):

$$\mathcal{R}_{\mathsf{DLeq}} := \{((g = Q, h, a = Q^s, b = \mathsf{vk}); \ \ x = s) \ \mid \ a = g^x \wedge b = h^x\}.$$

(9) Output $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct} = [\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell], \pi = (C_\alpha, \pi_\alpha, \pi_{\mathsf{LIN}}, \pi_{\mathsf{DLeq}}))$.

Ver$_{\mathsf{ct}}(F^{\textbf{full-eval}}_{[\ell]}, C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \to 0/1$:

(1) Parse $\mathsf{ct} = (\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$ and the server's proofs $\pi = (C_\alpha, \pi_\alpha, \pi_{\mathsf{LIN}}, \pi_{\mathsf{DLeq}})$.
(2) Verify π_{LIN} for C_α against $g_1, g_1^{\tau-\alpha}, h, C_\alpha, \mathsf{vk}$.
(3) Compute the Fiat-Shamir challenge: $\alpha := \mathsf{H}(C_\phi, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$.
(4) Verify the opening proof π_α: $e(C_\phi/C_\alpha, g_2) = e(\pi_\alpha, g_2^{\tau-\alpha})$.
(5) Compute Q as above for encryption.
(6) Compute $\mathsf{ct} := \prod_{i=0}^{\ell} \mathsf{ct}_i^{L_{i,\ell}(\alpha)} \in \mathsf{G}_1$, and $Q^* := \mathsf{ct}/C_\alpha$.
(7) Verify π_{DLeq} for the relation (Q, Q^*, h, vk).
(8) If any of the verification checks fail, output 0. Otherwise, output 1.

Ver$_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) \to 0/1$: For sk $= s \in \mathsf{F}_p$, return 1 if and only if vk $= h^s$.

Dec$(F^{\textbf{full-eval}}_{[\ell]}, \mathsf{sk}, \mathsf{ct}) \to \{\phi(i)\}_{i=0}^{\ell}$: For sk $= s \in \mathsf{F}_p$, for each $i \in [\ell]$, recover the value $x = \phi(i)$ by brute-forcing $x \in [\mathcal{B}]$ in $g_1^x = \mathsf{ct}_i/h_i^s$. Pollard's rho algorithm for logarithms [58], Shank's baby-step-giant-step algorithm [64] or brute-force with pre-computed tables [21] could be helpful to find the right trade-off between decryption memory and time.

[a]Generalizing to more general subsets S other than $[\ell]$, the description of the subset S would be part of the input of $\mathsf{H}(\cdot)$ generating the Fiat-Shamir challenge.

Figure 2: A VECK protocol instantiated with the exponential ElGamal encryption and KZG polynomial commitment schemes for the function F defined in Equation (5.1).

we can use full-opening VECK for polynomial $\phi_S(X)$ on the set $S' = S \cup \{-1\}$ to get the encryptions of values $\phi(i)$ for $i \in S$ and, additionally, an encryption of its value at (-1) (evaluation of $\phi_S(X)$ on point (-1)) to assist with VECK ciphertext verification. By the zero-knowledge property of the full-opening VECK, this encryption would leak no additional information about the polynomial.

An important property of our scheme, is that the output of encryption can also be computed without knowing the full polynomial $\phi(X)$, but only knowing the subset of evaluations: $\{\phi(i)\}_{i\in S}$ and the batch opening proof for this subset. It makes our scheme also applicable to distributed data-storage, where independent servers store the commitment C_ϕ, subsets of evaluations and the batch-proof (e.g. Danksharding see the discussion Section 8 and Appendix A.3 for more details).

Theorem 5.3. *The protocol described in Figure 5 is a secure VECK for function F defined in Equation Equation* (5.2).

We give the proof in Appendix C.4.

6 PERFORMANCE EVALUATION

In this section, we report on the asymptotic and concrete performance metrics of an implementation of our FDE protocols.

6.1 Theoretical performance

Our FDE protocols consist of three rounds: first, the server sends the ciphertexts and proofs. Second, the client locks money on-chain, and finally, the server reveals its VECK decryption key. Note that the withdrawal rounds can be amortized over multiple protocol runs as one does not necessarily need to withdraw their earned (or locked)

Gen(crs) $\rightarrow$ pp: Let the bound be $\mathcal{B} = \mathrm{poly}(\lambda)$.

Let k be such that $\mathcal{B} = 2^{\lceil \frac{\log p}{k} \rceil}$.

Generate $h_{i,j} \leftarrow_R \mathsf{G}_1$ for $i \in [n], j \in [k]$.

Enc($F^{\textbf{full-eval}}_{[\ell]}, C_\phi, \phi(X)) \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$:

(2) For $\forall i \in [\ell]$, let $\phi(i) = \phi_{i,0} + \phi_{i,1}\mathcal{B} + \ldots + \phi_{i,k}\mathcal{B}^k$,
for $\phi_{i,j} \in [\mathcal{B}]$ for $\forall j \in [k]$.
Compute $\{\mathsf{ct}_{i,j} := h^s_{i,j} \cdot g_1^{\phi_{i,j}}\}_{j=0}^k$.
Compute range-proof of encrypted values:
π_{range} (attesting to $\phi_{i,j} \in [\mathcal{B}]$).

(3) $\alpha := \mathsf{H}(C_\phi, \{\mathsf{ct}_{i,j}\}_{i\in[\ell], j\in[k]})$.

(7) $Q := \prod_{i=0}^{\ell} \prod_{j=0}^{k} h_{i,j}^{\mathcal{B}^j \, L_{i,\ell}(\alpha)} \cdot g_1^{-(\tau-\alpha)}$.

(9) Output $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct} = \{\mathsf{ct}_{i,j}\}_{(i\in[\ell], j\in[k])}$,
$\pi = (C_\alpha, \pi_\alpha, \pi_{\mathsf{LIN}}, \pi_{\mathsf{DLeq}}, \pi_{\mathsf{range}})$.

Ver$_{\mathsf{ct}}(F^{\textbf{full-eval}}_{[\ell]}, C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \rightarrow 0/1$:

(6) Compute $\mathsf{ct} := \prod_{i=0}^{\ell} \prod_{j=0}^{k} \mathsf{ct}_{i,j}^{\mathcal{B}^j \, L_{i,\ell}(\alpha)}$,
$Q^* := \mathsf{ct}/C_\alpha$,
check π_{range} against $\{\mathsf{ct}_{i,j}\}_{(i\in[\ell], j\in[k])}$.

Figure 3: Modifications to Figure 2 to remove the bound on the evaluations $\phi(i)$ and support arbitrary polynomials of degree ℓ. The precise relation for the range proofs is provided in Equation (A.5) in Appendix A.2.

coins after each exchange. The server's proofs are constant-sized in the exponential ElGamal-based protocol, *cf.* Figure 2 and linear in the Paillier-based protocol, *cf.* Figure 4. In both protocols, the client needs to submit only a single signature. The on-chain footprint of our protocols consists of three signatures (two transactions from the server, one from the client) and two group elements (sk and vk of the underlying VECK scheme). We compare our asymptotic performance with related work in Table 1.

6.2 Implementation performance

To measure concrete performance, we created a proof of concept implementation of our protocols using Rust v1.74.0 and Solidity v0.8.13. All our source code is publicly available[7].

All experiments were run on a consumer-grade PC with an AMD Ryzen 5 3600 (6-core) CPU and 8GB RAM. We used the Criterion benchmarking crate[8] to measure the execution time of the prover and the verifier in our protocols. Each measurement was repeated 10 times, and below, we report the mean of these protocol runs.

6.2.1 *Off-chain costs.*

Prover time. In the exponential ElGamal scheme, we used $k = 8$, that is, each BLS12-381 scalar is split into k smaller plaintexts. Range proofs and exponential ElGamal encryptions are computed in 89 s for 4, 096 exchanged BLS12-381 points (*cf.* Figure 6). This is a one-time cost for the prover and can serve multiple clients who request the same data with amortized prover costs, *cf.* Section 7. The prover's amortized overhead for proving the consistency of 4, 096 BLS12-381 field elements ($\approx$ 128 KiB) with respect to ElGamal ciphertexts takes less than 40 ms. Prover time in the exponential ElGamal protocol for a small number of exchanged points is dominated by the computation of the FFTs (*i.e.*, computing the quotient polynomial $q(X)$ in Step 5 of the protocol in Figure 2). Therefore, we observe that the proving overhead is larger for transferring fewer BLS12-381 scalar field elements. However, for larger exchanged data, ElGamal encryption and the range proof generation dominate our ElGamal prover time. On the other hand, proof generation in the Paillier-based protocol (*cf.* Figure 4) is monotonically increasing in the number of transferred data points. For 4, 096 BLS scalar field elements, proof generation takes on average $\approx$ 5.09 s.

Proof size We implemented our exponential ElGamal protocol (Section 5.1) with the range proof derived from the homomorphic polynomial commitment schemes by Boneh et al. [10]. These range proofs can be batched to improve the proof size (two G_1 elements for any number of ranges) and the verifier's concrete efficiency by replacing $O(n)$ pairing computations with $O(n)$ MSMs in G_1. We leave this optimization for future work. To exchange 4, 096 BLS scalar field elements (0.13 MB), the total bandwidth (cipher texts and proofs) is 1, 56 MB in the exponential ElGamal protocol (with $k = 8$ chunks per BLS field element), while 6, 55 MB in the Paillier-based protocol ($\lambda = 128$, thus, $\log_2(N) = 3072$). This constitutes a 11, 95$\times$ (and 50, 18$\times$) factor bandwidth overhead in our protocols, respectively, in comparison to the size of the exchanged data. In conclusion, we observe that the Paillier-based protocol does not improve concretely the bandwidth costs of the exponential ElGamal protocol (*cf.* Figure 2) due to its larger cryptographic groups and the linear-sized VECK proofs (*cf.* Figure 4). It is an interesting future direction to design constant-sized VECK proofs with the Paillier-encryption scheme.

Verifier time Verifying the proofs is slightly more expensive but still efficient. In particular, verifying the correctness of 4, 096 ElGamal ciphertexts with respect to a KZG commitment using the protocol in Figure 2 takes around 34.15 s. The verifier's time strictly increases in the number of opened points as it is dominated by multi-scalar multiplications that have sizes proportional to the number of exchanged data points. It should be noted that the verifier time is dominated by the split scalar encryption verification, meaning that the verifier needs to check whether an encrypted field element is properly split into smaller field elements within the brute-forceable range. With lookup tables, decryption of ElGamal ciphertexts would be quick and negligible in terms of compute costs. Verifying the proof of correctness in the Paillier-based protocol for 4, 096 BLS scalars takes roughly 19.45 s. Decrypting the 4, 096 Paillier ciphertexts takes $\approx$ 9.54 s.

6.2.2 *On-chain costs.*

Bitcoin. We include the Bitcoin Script corresponding to the bonding contract of Section 4.3 in Appendix B.1. The script contains two conditional executions. One execution uses a timelock and enables the spending of the locked tokens by the client after a timeout period. The other enables the spending of the funds by any transaction carrying the adaptor signature and the server's signature.

[7] https://github.com/PopcornPaws/fde.
[8] https://crates.io/crates/criterion.

Gen$(\mathsf{crs}) \rightarrow \mathsf{pp}$: On input $\mathsf{crs} = (\mathcal{G}, \{g_1^{\tau^i}\}_{i=1}^n, \{g_2^{\tau^i}\}_{i=1}^n)$, where τ is the secret trapdoor, let $\mathsf{H} : \{0,1\}^* \rightarrow [0, B)$ denote a hash function, and $A := Bp\ell 2^\lambda$.

Enc$(F_{[\ell]}^{\textbf{full-eval}}, C_\phi, \phi(X)) \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$:

(1) Generate keys for Paillier encryption: let $p', q' \leftarrow_R \mathsf{Primes}(1^\lambda)$, and compute the modulus $N := p'q'$, s.t. $N \geq 2\sqrt{2}AB$. Let $\mu := \mathsf{lcm}(p'-1, q'-1)$, where $\mathsf{lcm}(\cdot,\cdot)$ returns the least common multiple of two integers. We set the Paillier public key $\mathsf{vk} := N$, and the secret key $\mathsf{sk} := (p', q', \mu)$.
(2) Calculate the encryptions of $\{\phi(i)\}_{i\in[\ell]}$: for each $i \in [\ell]$ sample $U_i \leftarrow_R \mathsf{Z}_N^*$ and compute $\mathsf{ct}_i := (N+1)^{\phi(i)} U_i^N \mod N^2$.
(3) For each $i \in [\ell]$ sample $S_i \leftarrow_R \mathsf{Z}_N^*, r_i \leftarrow_R [0, A)$ and compute $T_i := (N+1)^{r_i} S_i^N \mod N^2$. Let $T := \prod_{i\in[\ell]} g_1^{L_{i,[\ell]}(\tau)\, r_i}$.
(4) Sample a random challenge $c := \mathsf{H}(\mathsf{vk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell, C_\phi, T_0, \ldots, T_\ell, T)$ using the Fiat-Shamir transformation.
(5) Compute $\forall i \in [\ell] : W_i := S_i U_i^c \mod N^2$ and $z_i := r_i + c\phi(i)$ over the integers ($z_i \in \mathsf{Z}$).
(6) Output $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct} = (\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell), \pi = (c, W_0, \ldots, W_\ell, z_0, \ldots, z_\ell))$.

Ver$_{\mathsf{ct}}(F_{[\ell]}^{\textbf{full-eval}}, C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \rightarrow 0/1$:

(1) Parse the input $\mathsf{ct} = (\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$ and $\pi = (c, W_0, \ldots, W_\ell, z_0, \ldots, z_\ell))$.
(2) Reject the proof if any of the z values are not within $[0, A)$.
(3) Compute $\forall i \in [\ell] : T_i := (N+1)^{z_i} W_i^N \mathsf{ct}_i^{-c} \mod N^2$, and $T := \prod_{i\in[\ell]} g_1^{L_{i,[\ell]}(\tau)\, z_i} / C_\phi^c$.
(4) Accept if $c = \mathsf{H}(\mathsf{vk}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell, C_\phi, T_0, \ldots, T_\ell, T)$.

Ver$_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) \rightarrow 0/1$: Return 1 if and only if $\mathsf{vk} = N = p'q'$ for $\mathsf{sk} = (p', q')$.

Dec$(F_{[\ell]}^{\textbf{full-eval}}, \mathsf{sk}, \mathsf{ct}) \rightarrow \{m_i\}_{i\in[\ell]}$: Compute $\forall i \in [\ell] : m_i = \frac{L(\mathsf{ct}_i^\mu \mod N^2)}{L((N+1)^\mu \mod N^2)} \mod N$, where $L(x) = \frac{x-1}{N}$ for $x \ddot{Y} N^2 \wedge x \equiv 1 \mod N$. Construct a polynomial $m(X) = \sum_{i\in[\ell]} m_i L_{i,\ell}(X)$ and check that $C_\phi = \mathsf{KZG.Commit}(\mathsf{crs}, m(X))$, if equality holds, we are done. Otherwise, following [37] for each m_i, we search the values (σ_1, σ_2) that constitute the smallest vector in the lattice of dimension two with basis $((N, 0), (m_i, 1))$. The smallest vector can be found in time $O(\log N)$ using Gauss algorithm (see [25], p.23). We then set $m_i := \sigma_1/\sigma_2 \mod p$.

Figure 4: A VECK protocol instantiated with the Paillier encryption and the KZG polynomial commitment scheme for encrypting polynomial's evaluations.

We expect the transaction fee of the bonding contract ($\approx$ 231 Bytes) to be below \$10 for a confirmation time of 1 hour on Bitcoin [9] *regardless of the data size.*

Ethereum All of our protocols in Section 4, except the protocol in Section 4.3, rely on the same on-chain smart contract logic that guarantees the atomicity of our fair exchange protocols. We implemented this FDE bonding logic for the Ethereum Virtual Machine (EVM), compiled from Solidity (our full source is included in Appendix B.2). The smart contract implements four functionalities (cf. Figure 1), and we report the EVM gas costs for each in Table 2. First, the server registers its public key on the blockchain in a compressed form, *i.e.*, only the x-coordinate of its public key by calling the serverSendsPubKey function. Additionally, the server sets the agreed price of the exchanged data. Afterward, the client locks her payment by calling the clientLocksPayment function according to the previously agreed price. Third, the server sends the decryption key of the VECK scheme and calls the serverSendsSecKey function. This function ensures that the contract verifies that the provided decryption key sk matches the public key vk the server submitted in the first step. Finally, the parties can withdraw their money with the withdrawPayment function: the server can withdraw if it has provided the decryption key, while the client can withdraw its locked payment after a timeout if the server fails to reveal the decryption key sk for the data. We observe in Table 2, that all the aforementioned operations are highly efficient and affordable on today's Ethereum in both of our protocols.

Transaction	**Gas cost**		**USD cost**	
	ElGamal	**Paillier**	**ElG.**	**Pail.**
serverSendsPubKey	158,449	176,296	5.11 \$	5.68 \$
clientLocksPayment	30,521	30,521	0.98 \$	0.98 \$
serverSendsSecKey	73,692	82,475	2.37 \$	2.65 \$
withdrawPayment	43,836	43,836	1.41 \$	1.41 \$

Table 2: EVM gas costs of the smart contract components for $\lambda = 128$ bits of security. Note these costs are constant in the size of the exchanged data. (*cf.* Appendix B.2), in the Exponential Elgamal (*cf.* Figure 2) and Paillier encryption-based FDE schemes (*cf.* Figure 4). For the USD costs, we used the transaction fee 14 GWei and 2,302.35 USD/ETH exchange rate as of 2024 February 3rd.

7 MULTI-CLIENT MODEL

In certain applications of the FDE protocol, the server is expected to provide the same data to multiple clients over time. For instance, in blockchain applications, light clients might query the same blocks

$\textsc{Gen}(\mathsf{crs}) \to \mathsf{pp}$: Output the result of $\mathrm{F}\textsc{Gen}(\mathsf{crs})$.
$\textsc{Enc}(F_S, C_\phi, \phi(X)) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$:
(1) Sample $t \leftarrow_R \mathsf{F}_p$ and construct the following polynomials in $\mathsf{F}_p[X]$:
- $V_S(X) := \prod_{i \in S}(X - i)$,
- $\phi_S(X) := \sum_{i \in S} \phi(i) L_{i,S}(X)$
- $\phi'_S(X) := tV_S(X) + \phi_S(X)$,

(2) Compute $C_S := \textsc{Commit}(\mathsf{crs}, \phi'_S(X))$.
(3) Compute $\pi_S := \textsc{Commit}(\mathsf{crs}, (\phi(X) \text{-} \phi'_S(X))/V_S(X))$.
(4) Let $S' := S \cup \{-1\}$
(5) Run $\textsc{Enc}(F^{\text{full-eval}}_{S'}, C_S, \phi'_S(X)) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi')$
(6) Output $(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi = (C_S, \pi_S, \pi'))$.

$\textsc{Ver}_{\mathsf{ct}}(F_S, C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \to 0/1$:
(1) Parse π as (C_S, π_S, π').
(2) Verify $e(C_\phi / C_S, g_2) = e(\pi_S, g_2^{V_S(\tau)})$.
(3) Run $\textsc{Ver}_{\mathsf{ct}}(F^{\text{full-eval}}_{S'}, C_S, \mathsf{vk}, \mathsf{ct}, \pi')$, where S' is defined above.

$\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) \to 0/1$: Output the result of $\mathrm{F}\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk})$.
$\textsc{Dec}(F_S, \mathsf{sk}, \mathsf{ct}) \to \{\phi(i)\}_{i \in S}$: Output the result of $\mathrm{F}\textsc{Dec}(F_{S'}, \mathsf{sk}, \mathsf{ct})$ (S' is defined above).

Figure 5: VECK protocol for encryptions of subsets of committed vectors, *i.e.*, function of Equation (5.2).

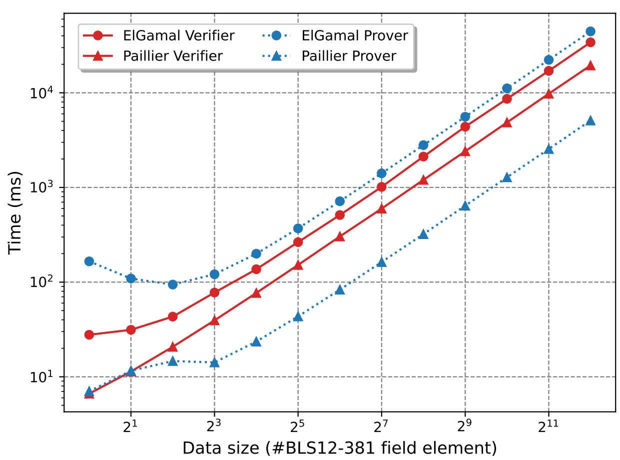

Figure 6: Prover and verifier times in the VECK schemes described in Figure 2 and Figure 4. Interestingly, in the exponential ElGamal VECK scheme, the prover time is dominated by the FFT computations. Hence, if smaller sets $S \subset [1, n]$ are opened, then we observe larger prover time due to the larger degree of the quotient polynomial $q(X)$, cf. Step 5 in Figure 2.

from the validators. We next formulate a model for this setting and construct a VECK protocol that minimizes the server's work.

7.1 Problem Setup

Consider a single server $\mathcal{S}$ with the messages $(m_i)_{i \in [n]}$ from some message space $\mathcal{M}$, and multiple clients $(C_i)_{i=1,\ldots,K}$ that query $\mathcal{S}$ for the messages. In the multi-client model, we would like to guarantee that a client cannot obtain the messages without getting the ciphertexts and the decryption key from the server, except when it downloads the data from one of the other clients. Clients can communicate with each other, but the total number of bits, received by a client from the other clients cannot exceed a constant L.

In the setting of multiple clients, we can save the server some work by moving parts of the proof generation to a *preprocessing* step that is amortized across all clients querying the same data. Thus, we distinguish between the preprocessing phase of the protocol and the online phase that is repeated for each client. To illustrate this distinction, consider an example with $L = 0$. In this case, $\mathcal{S}$ has to generate an encryption of the messages and the proof only once, and can reuse the same ciphertexts and the proof across all clients. Since the proof is generated only once, $\mathcal{S}$ can pick a symmetric key encryption scheme with secret key α (succinct in n)[9] that has an efficient decryption algorithm and a small message complexity for the ciphertexts, but not necessarily a fast algorithm for generating a VECK proof. Then, it can leverage the power of a generic SNARK with an efficient constant time verifier and a small constant size proof (e.g., [40, 41]) to generate the proof. As the encryption and the proof generation are *not* repeated per client, this scheme achieves a small amortized proving cost for large K and reduces the ciphertext decryption time and proof verification time.

In practice, clients do communicate with each other. When they are able to convey each other the whole length N message sequence, *i.e.*, $L = O(N \log(|\mathcal{M}|))$ bits, $\mathcal{S}$ cannot expect to profit from its interaction with the clients, since the first client to receive the messages could broadcast it to the others, making any further communication obsolete. However, this implicitly assumes that clients can act as servers themselves, which contradicts the premise of the FDE problem in the first place. Thus, in this section, we assume that $L = o(N \log(|\mathcal{M}|) + \lambda)$. Note that a small L still yields a non-trivial problem, as $\mathcal{S}$ can no longer use the same secret key α for all clients, lest they share the key with each other.

7.2 Security Definition

We now define the security properties for MC-VECK.

Definition 7.1 (multi-client-VECK). *A non-interactive Multi-Client-VECK (MC-VECK) scheme for a function F augments a VECK scheme from Definition 3.1 with an additional preprocessing algorithm that outputs a secret and a public state for the server that is used in the encryption. Namely, MC-VECK is a tuple of algorithms $\Pi = (\textsc{Gen}, \textsc{Prep}, \textsc{Enc}, \textsc{Ver}_{\mathsf{ct}}, \textsc{Ver}_{\mathsf{key}}, \textsc{Dec})$ defined for a class of functions $\mathcal{F} = \{F : \mathcal{W} \to \mathcal{V}\}$ and a commitment scheme $(\textsc{Setup}, \textsc{Commit})$ over the space $\mathcal{W}$. The only algorithms whose input-output formats are different are $\textsc{Prep}$ and $\textsc{Enc}$ defined as follows:*
- *$\textsc{Prep}(\mathsf{pp}, C_w, w) \to (\mathsf{aux}, \mathsf{msk})$: Probabilistic polynomial-time algorithm that outputs a public value* aux *and a server's master secret key* msk *(*pp *will be omitted from the algorithms for brevity).*
- *$\textsc{Enc}(\mathsf{pp}, F, \mathsf{aux}, \mathsf{msk}) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$: Probabilistic polynomial-time algorithm run by the server, it takes as input the value* aux *and a master secret key* msk*, and outputs a verification*

[9] Otherwise, the scheme violates the succinctness assumption.

key vk*, a secret key* sk*, an encryption* ct *of* $F(w)$ *and a proof* π*.*

An MC-VECK protocol $\Pi = (\textsc{Gen}, \textsc{Prep}, \textsc{Enc}, \textsc{Ver}_{\mathsf{ct}}, \textsc{Ver}_{\mathsf{key}}, \textsc{Dec})$ *is correct and sound if the VECK* $\Pi' = (\textsc{Gen}, \textsc{Enc}', \textsc{Ver}_{\mathsf{ct}}, \textsc{Ver}_{\mathsf{key}}, \textsc{Dec})$ *where encryption is derived as follows is correct and sound.*

$\textsc{Enc}'(\mathsf{pp}, F, C_w, w)$:

- *Run* $\textsc{Prep}(\mathsf{pp}, F, C_w, w) \rightarrow (\mathsf{aux}, \mathsf{msk})$.
- *Run* $\textsc{Enc}(\mathsf{pp}, F, \mathsf{aux}, \mathsf{msk}) \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$
- *Output* $(\mathsf{vk}' = (\mathsf{vk}, \mathsf{aux}), \mathsf{sk}, \mathsf{ct}, \pi)$.

We additionally require the MC-VECK scheme to satisfy computational $\mathcal{L}$-bits zero-knowledge*. This property implies that one client, Alice, who successfully completed the data-exchange, won't be able to help the other client, Bob, who only downloaded the encryption, learn anything about the encrypted data, other than the bits that Alice sends Bob directly. With this property, it is guaranteed that in order to obtain the data, Bob has to either complete the data-exchange with the server or download full data from Alice, as no partial help from Alice would help Bob avoid paying the server for the decryption key in our FDE protocol.*

Computational $\mathcal{L}$*-bits Zero-Knowledge: We generalize zero-knowledge in order to prevent the attacker who learns some* $\mathcal{L}$*-bits from other clients from learning anything else about the data. For any PPT algorithms* $\mathcal{A}_1, \mathcal{A}_2$*, where* $\mathcal{A}_1$ *outputs at most* $\mathcal{L}$*-bits hint, there exists a PPT simulator* Sim*, there is a negligible function* $\mu(\cdot)$ *such that for all* $w \in \mathcal{W}$*, the following probability is less than* $\frac{1}{2} + \mu(\lambda)$*:*

$$\Pr\left[\begin{array}{l|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ & pp \leftarrow \textsc{Gen}(\mathsf{crs}) \\ & (\mathsf{aux}, \mathsf{msk}) \leftarrow \textsc{Prep}(F, C_w, w) \\ \mathcal{A}_2(\mathsf{pp}, F, C_w, \mathsf{hint} & (\mathsf{vk}_*, \mathsf{sk}_*, \mathsf{ct}_*, \pi_*) \leftarrow \textsc{Enc}(F, \mathsf{aux}, \mathsf{msk}) \\ \mathsf{vk}_b, \mathsf{ct}_b, \pi_b) = b & \mathsf{hint} \leftarrow \mathcal{A}_1(\mathsf{vk}_*, \mathsf{sk}_*, \mathsf{ct}_*, \pi_*) \\ & \\ & (\mathsf{vk}_0, \mathsf{sk}_0, \mathsf{ct}_0, \pi_0) \leftarrow \textsc{Enc}(F, \mathsf{aux}, \mathsf{msk}) \\ & (\mathsf{vk}_1, \mathsf{ct}_1, \pi_1) \leftarrow \mathsf{Sim}(\mathsf{pp}, C_w, \mathsf{hint}) \\ & b \leftarrow_R \{0, 1\} \end{array}\right]$$

7.3 MC-VECK for ElGamal Encryption

We next describe an MC-VECK protocol for exponential ElGamal encryption that extends the VECK scheme in Section 5.1 (Figure 2). The protocol consists of an offline phase and an online phase (Figure 7). Recall that $(h_i)_{i\in[n]} \in \mathsf{G}_1^{n+1}$ denote the public parameters of the ElGamal encryption.

Preprocessing Step. In the preprocessing step, $\mathcal{S}$ runs the $\textsc{Enc}$ algorithm of the original VECK scheme (*cf.* Definition 7.1) and outputs the verification key vk, the ciphertexts ct and the VECK proof π for the function F as part of the preprocessing output aux and outputs the decryption key sk as the master secret key msk (*cf.* Figure 2 and Figure 7). Let $H\colon \{0,1\}^* \rightarrow \mathsf{F}_p$ be a hash function modelled as a random oracle.

Online Step of MC-VECK. When $\mathcal{S}$ starts interacting with a new client C, it samples a new key δ_C u.a.r., and calculates the commitments

$$D_C = h^{\delta_C} \text{ and } h_{C,i} = (h_i^{\delta_C})_{i\in[n]}.$$

Construction 1: ElGamal MC-VECK for the function $F := F^{\text{full-eval}}_{[n]}$. It readily generalizes to subset openings since the protocol for subset openings use a VECK protocol for $F^{\text{full-eval}}_{[n]}$ as blackbox.

$\textsc{Gen}(\mathsf{crs}) \rightarrow \mathsf{pp}$: Output $\textsc{Setup}(\mathsf{crs}) \rightarrow \mathsf{pp}$ as in Figure 2.

$\textsc{Prep}(F, C_\phi, \phi) \rightarrow (\mathsf{aux}, \mathsf{msk})$:

1. Run $\textsc{Enc}(F, C_\phi, \phi) \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$ as in Figure 2.
2. Output $\mathsf{aux} = (\mathsf{vk}, \mathsf{ct}, \pi)$.
3. Output $\mathsf{msk} = \mathsf{sk}$.

$\textsc{Enc}(F, \mathsf{aux}, \mathsf{msk}) \rightarrow (\mathsf{vk}_C, \mathsf{sk}_C, \mathsf{ct}, \pi_C)$:

1. Parse $\mathsf{aux}, \mathsf{msk} \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$.
2. Sample $\delta_C \leftarrow_R \mathsf{F}_p$.
3. Set $D_C = h^{\delta_C}$, $h_{C,i} = (h_i^{\delta_C})_{i\in[n]}$.
4. Set $e_i = H(D_C, i)$ for all $i \in [n]$.
5. Calculate $Q = \prod_{i\in[n]} h_i^{e_i}$.
6. Compute discrete logarithm equality π_{DLEq} for $(Q, Q^{\delta_C}, h, D_C)$ with the witness δ_C [22].
7. Set
   - $\mathsf{vk}_C = \mathsf{vk} \cdot D_C$.
   - $\mathsf{sk}_C = \mathsf{sk} + \delta_C$.
   - $\pi_C = (\pi, D_C, \pi_{\mathsf{DLEq}}, (h_{C,i})_{i\in[n]})$.
8. Output $(\mathsf{vk}_C, \mathsf{sk}_C, \mathsf{ct}, \pi_C)$.

$\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C, \mathsf{ct}, \pi_C) \rightarrow 0/1$:

1. Parse $\pi_C \rightarrow (\pi, D_C, \pi_{\mathsf{DLEq}}, (h_{C,i})_{i\in[n]})$.
2. Verify if $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C/D_C, \mathsf{ct}, \pi) \rightarrow 1$ as in Figure 2.
3. Set $e_i = H(D_C, i)$ for all $i \in [n]$.
4. Compute Q as above.
5. Compute $Q^* = \prod_{i\in[n]} h_{C,i}^{e_i}$.
6. Verify π_{DLEq} for the relation (Q, Q^*, h, D_C).
7. Return 1 if the steps above succeed, 0 o.w.

$\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}_C, \mathsf{sk}_C) \rightarrow 0/1$: Return 1 if $\mathsf{vk}_C = h^{\mathsf{sk}_C}$, 0 o.w.

$\textsc{Dec}(F, \mathsf{sk}_C, (\mathsf{ct}, (h_{C,i})_{i\in[n]})) \rightarrow \phi(i)$:

1. Parse $\mathsf{ct} \rightarrow (\mathsf{ct}_i)_{i\in[n]}$.
2. Set $\mathsf{ct}_C = (\mathsf{ct}_i \cdot h_{C,i})_{i\in[n]}$.
3. Output $\textsc{Dec}(F, \mathsf{sk}_C, \mathsf{ct}_C)$ as in Figure 2.

Figure 7: The MC-VECK protocol for ElGamal encryption. The parameters with subscript C are generated per client C.

It then calculates $Q = h_i^{e_i}$, where $e_i = H(D_C, i)$ for all $i \in [n]$, and computes a single discrete logarithm equality proof π_{DLEQ} with respect to $(Q, Q^{\delta_C}, h, D_C)$. Informally, the values $\{h_{C,i}\}$ will be used to rerandomized the ElGamal encryption so that the corresponding decryption key becomes $\mathsf{sk}_C = \mathsf{msk} + \delta_C$. The proof π_{DLEQ}, on the other hand, will attest to the correct rerandomization. The final proof π_C includes π, D_C, π_{DLEq} and $(h_{C,i})_{i\in[n]}$.

Upon receiving vk_C, ct and $\pi_C = (\pi, D_C, \pi_{\mathsf{DLEq}}, (h_{C,i})_{i\in[n]})$, the client C first verifies the VECK proof π with respect to C_ϕ, vk_C/D_C and ct. It then finds $(e_i)_{i\in[n]}$ and Q as described above and checks the proof π_{DLEQ} against $(Q, \prod_{i\in[n]} h_{C,i}^{e_i}, h, D_C)$. If the steps succeed in C's view, it accepts the proof π_C.

Once the decryption key sk_C is published, its correctness is verified by checking if $h^{\mathsf{sk}_C} = \mathsf{vk}_C$.

Finally, to decrypt ct, C calculates $\mathsf{ct}_C = (\mathsf{ct}_i \cdot h_{C,i})_{i \in [n]}$ as the rerandomized ciphertexts in the target group, and outputs $\textsc{Dec}(\mathsf{sk}_C, \mathsf{ct}_C)$ as in Figure 2 by running the decryption algorithm in the target group.

Discussion. The MC-VECK protocol enables the prover to reuse the original ciphertexts and the VECK proof π across all clients (*e.g.*, the proofs π_α, π_{LIN}, π_{DLeq} and most importantly the range proofs). As generating the ciphertexts and the range proofs constitute the bulk of the proving time, this saves the prover considerable computation (*cf.* Section 6).

To minimize the client's verification work and the server's communication complexity, the server $\mathcal{S}$ could also post C_ϕ, the verification key vk, ciphertexts ct and proof π to a smart contract on the blockchain after the preprocessing step. The contract then runs $\textsc{Ver}_{\mathsf{ct}}(C_\phi, \mathsf{vk}, \mathsf{ct}, \pi)$, thus removing the need for the clients to later individually verify the proof π. Moreover, in any future interaction with a client C, $\mathcal{S}$ would no longer have to send ct and π to C; since π would already be verified by the contract, and C would be able to retrieve ct and π from the blockchain. This would improve applications where the total cost of server's communication and the clients' verification dominates the cost of verifying the data once on the blockchain.

Finally, the MC-VECK protocol above can be readily modified to support subset openings of KZG commitments as the protocol in Section 5.3 makes use of a VECK protocol for the function $F^{\text{full-eval}}$ as blackbox.

Analysis. Security of MC-VECK protocols is stated below.

Theorem 7.1. *Given H modelled as a random oracle, the protocol in Figure 7 is a secure MC-VECK protocol in the random oracle and algebraic group models.*

Proof is given in Appendix C.5. Intuitively, correctness and soundness follow from the same properties of the VECK protocols used in the preprocessing step.

One important property inherited from the zero-knowledge of VECK protocols is that given any two values $w_0 < w_1$ committed by the same C_w, no PPT adversary would be able to distinguish their encryptions. Although many such values exist, certainly no PPT algorithm can break binding and actually find them (assuming that the crs was generated securely), so we talk about them as barely hypothetical values. Then, even if the adversary knows the value w_0, it would not be able to distinguish the encryption of $F(w_0)$ from the encryption of $F(w_1)$ until the decryption keys are revealed. Since this is true for the adversary that might know w_0, it is also true for the adversary that holds partial information about $F(w_0)$ or w_0 itself, *e.g.*, a hint based on w_0. Then, assuming multiple (possibly colluding) clients downloading the same data, if one client, who already holds $F(w)$, tries to help the other client, who only downloaded the encryption of $F(w)$ from the server without yet obtaining the decryption key, it won't be able to do so, since the communicated hint reveals no additional information about $F(w)$. This would imply L-bits zero-knowledge for the MC-VECK protocols.

8 DISCUSSION AND FUTURE DIRECTIONS

We leave the following open problems, extensions, and possible improvements for future work.

Optimizations. If a client requests multiple batches of data under different commitments, the server can reuse the same decryption key for the ciphertexts, reducing the on-chain footprint of the protocol. In the multi-client model, the server currently generates and remembers a different decryption key to interact with each client. As an optimization, the server could instead use a PRF function $f_k(\cdot)$, and compute the i-th client's decryption key as $\mathsf{sk}_i := f_k(i)$; the server would then only store an index for each client.

Optimizing bandwidth. Both our protocols concretely incur at least a 10×-factor of bandwidth blowup (*i.e.* ciphertexts and proofs) in comparison to just sending the plaintext data, *cf.* Section 6. Thus, it is an interesting open problem to design a VECK proof system with smaller overhead and for the KZG commitment and Paillier encryption schemes, with constant proof size. One approach for smaller overhead would be "packing" multiple data points $\{\phi(i)\}_{i=1}^k \in \mathsf{F}_p^k$ into a single Paillier ciphertext $c \in \mathsf{Z}_{N^2}^*$.

Distributed data-storage. In Danksharding [15] the blockchain data is erasure coded using Reed-Solomon codes [60] and stored on multiple different servers. Hence, when a client wishes to pay for certain data, it needs to fetch the data and the accompanying proofs from multiple servers. Our FDE protocol with subset openings readily generalizes to this setting; so that the client can engage with multiple servers independently to reconstruct the original data after fairly obtaining all the fragments required for reconstruction. We provide technical details on Danksharding in Appendix A.3.

An intermediate step towards Danksharding is EIP-4844 [15], which provides the functionality of persisting data on-chain for a predetermined period of 1-2 months. After the data expires, the validators are no longer obliged to store it, although they keep the KZG commitments to the data. Our protocol can be used to pay archival nodes for accessing the expired data in a fair and trust-minimized way, thus bringing in financial incentives to ensure that the data continues to be available.

Server griefing. In our current design, a malicious client could grief the server by having it produce the ciphertext (and the correctness proofs) but never requesting the decryption key. While this can be mitigated in a similar way to standard denial-of-service attacks, a promising alternative approach is to split the client's payment into two parts as follows: a first small payment is provided with the request, essentially to reimburse the server for its computation cost; the second payment is provided as before—as an exchange for the VECK decryption key. While this can allow the server to simply pocket the first payment, assuming that a competitive market exists for providing the data, users will simply choose a different server, and the servers will not risk their reputation to steal the first payment, which would typically be small compared to the price of the content. Moreover, faced with a grieving attack, the server can reuse the same ciphertext, the proof and the decryption key in its interaction with different clients willing to buy the same data, thus mitigating the attack in a practical sense (despite saving on compute, the server would not save on bandwidth as it would still have to send the encryptions).

Market pricing. We envision a marketplace for committed data with multiple servers and clients, where servers compete to fulfill clients' requests for data. In our proof of concept implementations, the server $\mathcal{S}$ sets explicitly the agreed price for the data exchange. We envision that real-world deployments of our protocols will employ other (automated) market-making mechanisms, *e.g.*, the constant product formula of Uniswap [2], to establish automatically and trustlessly the price of the subsequent fair data exchange. Specifically, the pricing function is $f(x, y) = x \cdot y = const$, where x is the amount of data waiting to be exchanged in the FDE protocol, while y is the number of clients wishing to download committed data. Future FDE applications could also apply more sophisticated market-making mechanisms.

Other commitment and encryption schemes. We believe that we only scratched the surface of FDE protocols' design space. In particular, we leave it to future work to explore other combinations of encryption (or functional encryption) and commitment schemes, such as a recent FRI-based commitment schemes for distributed data storage [43]. The design principles behind VECK schemes and FDE protocols can also be generalized to functions beyond data exchange such as exchanging the result of a generic computation; however, the applications explored in this paper focus on functions with non-succinct outputs, enabling the use of more communication-permissive techniques.

9 ACKNOWLEDGMENTS

We thank Dan Boneh for suggesting us to look at the multi-client setting for FDE and for helping us start working on that direction. We also thank Robin Linus for his help with the Bitcoin script. Ertem Nusret Tas was partially supported by the Stanford Center for Blockchain Research.

REFERENCES

[1] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. 2023. Recurring Contingent Service Payment. In *EuroS&P*. IEEE, 724–756.
[2] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. 2021. An analysis of Uniswap markets. *Cryptoeconomic Systems Journal* (2021).
[3] Nadarajah Asokan, Victor Shoup, and Michael Waidner. 2000. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in communications* 18, 4 (2000), 593–610.
[4] N. Asokan, Victor Shoup, and Michael Waidner. 2000. Optimistic fair exchange of digital signatures. *IEEE J. Sel. Areas Commun.* 18, 4 (2000), 593–610.
[5] Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Garth R. Goodson, and Bianca Schroeder. 2008. An analysis of data corruption in the storage stack. *ACM Trans. Storage* 4, 3 (2008), 8:1–8:28.
[6] Waclaw Banasik, Stefan Dziembowski, and Daniel Malinowski. 2016. Efficient Zero-Knowledge Contingent Payments in Cryptocurrencies Without Scripts. In *ESORICS (2) (Lecture Notes in Computer Science, Vol. 9879)*. Springer, 261–280.
[7] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. 2015. Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 287–304.
[8] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *CCS*. ACM, 967–980.
[9] Bitcoin Fee Calculator 2024. *Bitcoin Fee Calculator*. https://bitcoinjobs.com/fee-calculator
[10] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. 2020. Efficient polynomial commitment schemes for multiple points and polynomials. *IACR Cryptol. ePrint Arch.* (2020), 81.
[11] Dan Boneh and Victor Shoup. 2023. A graduate course in applied cryptography. *Draft 0.6* (2023).
[12] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. *IACR Cryptol. ePrint Arch.* (2017), 1050.
[13] Kevin D. Bowers, Ari Juels, and Alina Oprea. 2009. Proofs of retrievability: theory and implementation. In *CCSW*. ACM, 43–54.
[14] Vitalik Buterin. 2017. *Sharding FAQ*. https://vitalik.eth.limo/general/2017/12/31/sharding_faq.html
[15] Vitalik Buterin, DL Dankrad Feist, G Kadianakis, M Garnett, and A Dietrichs. 2022. Eip-4844: Shard blob transactions. *EIPS/eip-4844* (2022).
[16] Jan Camenisch and Victor Shoup. 2003. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *CRYPTO (Lecture Notes in Computer Science, Vol. 2729)*. Springer, 126–144.
[17] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *CCS*. ACM, 2075–2092.
[18] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services. In *CCS*. ACM, 229–243.
[19] Statistics Canada. 2019. Study: The value of data in Canada: Experimental estimates. https://www150.statcan.gc.ca/n1/en/daily-quotidien/190710/dq190710a-eng.pdf.
[20] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *Public Key Cryptography (Lecture Notes in Computer Science, Vol. 7778)*. Springer, 55–72.
[21] Panagiotis Chatzigiannis, Konstantinos Chalkias, and Valeria Nikolaenko. 2021. Homomorphic Decryption in Blockchains via Compressed Discrete-Log Lookup Tables. In *DPM/CBT@ESORICS (Lecture Notes in Computer Science, Vol. 13140)*. Springer, 328–339.
[22] David Chaum and Torben P. Pedersen. 1992. Wallet Databases with Observers. In *CRYPTO (Lecture Notes in Computer Science, Vol. 740)*. Springer, 89–105.
[23] Jeremy Clark and Aleksander Essex. 2012. CommitCoin: Carbon Dating Commitments with Bitcoin - (Short Paper). In *Financial Cryptography (Lecture Notes in Computer Science, Vol. 7397)*. Springer, 390–398.
[24] Bram Cohen. 2003. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*.
[25] Henri Cohen. 2013. *A course in computational algebraic number theory*. Vol. 138. Springer Science & Business Media.
[26] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. 2013. Election Verifiability or Ballot Privacy: Do We Need to Choose?. In *ESORICS (Lecture Notes in Computer Science, Vol. 8134)*. Springer, 481–498.
[27] Wei Dai, Tatsuaki Okamoto, and Go Yamamoto. 2022. Stronger Security and Generic Constructions for Adaptor Signatures. In *INDOCRYPT (Lecture Notes in Computer Science, Vol. 13774)*. Springer, 52–77.
[28] Ivan Bjerre Damgård. 1989. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*. Springer, 416–427.
[29] Danksharding 2024. *Danksharding*. https://ethereum.org/roadmap/danksharding
[30] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. FairSwap: How To Fairly Exchange Digital Goods. In *CCS*. ACM, 967–984.
[31] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. 2020. OptiSwap: Fast Optimistic Fair Exchange. In *AsiaCCS*. ACM, 543–557.
[32] Dankrad Feist and Dmitry Khovratovich. 2023. Fast amortized KZG proofs. *IACR Cryptol. ePrint Arch.* (2023), 33.
[33] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO (Lecture Notes in Computer Science, Vol. 263)*. Springer, 186–194.
[34] Filecoin 2024. *Filecoin*. https://filecoin.io/
[35] Ben Fisch. 2019. Tight Proofs of Space and Replication. In *EUROCRYPT (2) (Lecture Notes in Computer Science, Vol. 11477)*. Springer, 324–348.
[36] Ben Fisch, Joseph Bonneau, Nicola Greco, and Juan Benet. 2018. *Scaling proof-of-replication for Filecoin mining*. Technical Report. Protocol Labs.
[37] Pierre-Alain Fouque and Jacques Stern. 2001. One Round Threshold Discrete-Log Key Generation without Private Channels. In *Public Key Cryptography (Lecture Notes in Computer Science, Vol. 1992)*. Springer, 300–316.
[38] Georg Fuchsbauer. 2019. WI Is Not Enough: Zero-Knowledge Contingent (Service) Payments Revisited. In *CCS*. ACM, 49–62.
[39] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The Algebraic Group Model and its Applications. In *CRYPTO (2) (Lecture Notes in Computer Science, Vol. 10992)*. Springer, 33–62.
[40] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* (2019), 953.
[41] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT (2) (Lecture Notes in Computer Science, Vol. 9666)*. Springer, 305–326.
[42] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. 2023. Foundations of Data Availability Sampling. *IACR Cryptol. ePrint Arch.* (2023), 1079.
[43] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. 2024. FRIDA: Data Availability Sampling from FRI. *Cryptology ePrint Archive* (2024).
[44] Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, and Chase Qishi Wu. 2021. Fair Peer-to-Peer Content Delivery via Blockchain. In *ESORICS (1) (Lecture Notes in Computer Science, Vol. 12972)*. Springer, 348–369.

[45] Economist Impact. 2022. The future of Europe's data economy. https://impact.economist.com/perspectives/sites/default/files/ei233_msft_futuredata_report_-_v7.pdf.
[46] Simon Janin, Kaihua Qin, Akaki Mamageishvili, and Arthur Gervais. 2020. FileBounty: Fair Data Exchange. In *EuroS&P Workshops*. IEEE, 357–366.
[47] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 6477)*. Springer, 177–194.
[48] Jiwon Lee, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. 2019. SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. *IACR Cryptol. ePrint Arch.* (2019), 1270. Financial Crypto 2024.
[49] Robin Linus. 2023. *BitStream: Decentralized File Hosting Incentivised via Bitcoin Payments.* https://robinlinus.com/bitstream.pdf
[50] Markets and Markets. 2023. Cloud Storage Market. https://www.marketsandmarkets.com/Market-Reports/cloud-storage-market-902.html.
[51] Gregory Maxwell. 2011. Zero Knowledge Contingent Payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.
[52] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review* (2008).
[53] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. 2024. Powers-of-Tau to the People: Decentralizing Setup Ceremonies. 14585 (2024), 105–134.
[54] Henning Pagnia and Felix C Gärtner. 1999. *On the impossibility of fair exchange without a trusted third party.* Technical Report. Citeseer.
[55] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.
[56] The European Parliament and the Council of the European Union. 2024. Data Act. https://digital-strategy.ec.europa.eu/en/policies/data-act.
[57] Olivier Pereira. 2014. Verifiable Elections with Commitment Consistent Encryption - A Primer. *CoRR* abs/1412.7358 (2014).
[58] John M Pollard. 1978. Monte Carlo methods for index computation. *Mathematics of computation* 32, 143 (1978), 918–924.
[59] Guillaume Poupard and Jacques Stern. 2000. Fair Encryption of RSA Keys. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 1807)*. Springer, 172–189.
[60] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.
[61] David Reinsel-John Gantz-John Rydning, John Reinsel, and John Gantz. 2018. The digitization of the world from edge to core. *Framingham: International Data Corporation* 16 (2018), 1–28.
[62] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4 (1991), 161–174.
[63] Scroll 2024. *Scroll.* https://scroll.io/
[64] Daniel Shanks. 1971. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc., 1971*, Vol. 20. 415–440.
[65] Ludwig Siegele. 2020. A deluge of data is giving rise to a new economy. *The Economist* 5 (2020), 2–7.
[66] Markus Stadler. 1996. Publicly Verifiable Secret Sharing. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 1070)*. Springer, 190–199.
[67] Starkware 2024. *Starkware.* https://starkware.co/
[68] Elias Strehle and Fred Steinmetz. 2020. Dominating OP Returns: The Impact of Omni and Veriblock on Bitcoin. *J. Grid Comput.* 18, 4 (2020), 575–592.
[69] Ertem Nusret Tas and Dan Boneh. 2023. Cryptoeconomic Security for Data Availability Committees. In *FC (Lecture Notes in Computer Science, Vol. 13951)*. Springer, 310–326.
[70] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.
[71] zkSync 2024. *zkSync.* https://zksync.io/

A EXTENDED PRELIMINARIES

A.1 Cryptographic assumptions and models

For the sake of completeness, we enclose the definitions of the applied cryptographic assumptions.

A.1.1 The DDH assumption.

Definition A.1. *The* Decisional Diffie-Hellman (DDH) assumption *holds for the group generator $GGen(\cdot)$ that outputs groups of order p if there is no efficient adversary $\mathcal{A}$ that can distinguish the ensembles (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^r), for group generator $g \leftarrow_R \mathsf{G}$, and $a, b, r \leftarrow_R \mathsf{F}_p$. More formally, we require that the following holds for the group G output by $GGen(\cdot)$:*

$$|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^r) = 1]| \leq \mathsf{negl}(\lambda). \tag{A.1}$$

The probability is over the random choice of g, G according to the distribution induced by $GGen(\cdot)$, the random choice of $a, b, r \in \mathsf{F}_p$, and the random bits used by $\mathcal{A}$. The group family G satisfies the DDH assumption if there is no DDH algorithm for G.

A.1.2 The DCR assumption.

Definition A.2. *Given the security parameter λ, the set $S(p)$ of p bit primes such that $p = 2p' + 1$ for some prime p' and some $\chi = \mathsf{poly}(p)$, the* decisional composite residuosity (DCR) assumption *states that the tuples (N, u) and (N, v) are computationally indistinguishable for $N = p'q'$, where $p', q' \leftarrow_R S(p)$, $u \leftarrow_R \mathsf{QR}_{N^{\chi+1}} = \{a^2 | a \in \mathsf{Z}^*_{N^{\chi+1}}\}$ and $v \leftarrow_R \mathsf{HC}_{N^{\chi+1}} = \{a^{2N^{\chi}} | a \in \mathsf{Z}^*_{N^{\chi+1}}\}$. Here, $\mathsf{QR}_{N^{\chi+1}}$ denotes the subgroup of quadratic residues and $\mathsf{HC}_{N^{\chi+1}}$ denotes the subgroup of N^{χ}-th* residues *modulo $N^{\chi+1}$.*

A.1.3 The algebraic group model (AGM).

We prove the security of our schemes in the algebraic group model introduced by Fuchsbauer, Kiltz, and Loss [39]. The algebraic group model is an abstraction of an adversary that can only use algebraic algorithms in its attacks.

Definition A.3 (Algebraic algorithm [39]). *An algorithm $\mathcal{A}$ is* algebraic *if $\forall z \in \mathsf{G}$ that is output by $\mathcal{A}$ (either as returned by $\mathcal{A}$ or by invoking an oracle), $\mathcal{A}$ also provides the representation of z with respect to previously received group elements from G. More formally, if* $\mathsf{elems} \in \mathsf{G}^n$ *a vector of previously received group elements, then $\mathcal{A}$ must be able to provide a vector* r *representing z in the group,* i.e., *it holds that $z = \langle \mathsf{elems}, \mathsf{r} \rangle$.*

A.2 Applied Sigma-protocols and proof systems

In this section we state all the Σ-protocols used in our constructions. We remind that a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$ ($\mathcal{X}$ are the statements and $\mathcal{W}$ are the witnesses) is a three-move protocol, where the prover P holds (x, w) and starts the protocol, the verifier V holds x and responds with a random message from the challenge space $c \leftarrow_R C$, and after P's response, V accepts or rejects. Although we describe interactive variants, in our protocols, we use their non-interactive counterparts that can be obtained via a standard Fiat-Shamir transformation, where the verifier's challenge is replaced with a hash of the prover's first message.

We first introduce a protocol of Schnorr [62] as an identification scheme to prove the knowledge of a discrete logarithm in a prime-order group where the discrete logarithm problem is hard. More formally, Schnorr is a protocol with special soundness and special HVZK for the following relation in a group G ($|\mathsf{G}| = p$) with group generator $g \leftarrow_R \mathsf{G}$.

$$\mathcal{R}_{\mathsf{DL}} := \{(h, x) | h = g^x\}. \tag{A.2}$$

The proof system proceeds as follows.

- The prover samples $r \leftarrow_R \mathsf{F}_p$, and sends $R := g^r \in \mathsf{G}$ to the verifier.

- The verifier samples a random challenge $c \leftarrow_R \mathsf{F}_p$, and sends it to the prover.
- The prover replies with $z = cx + r \in \mathsf{F}_p$.

The proof π_{DL} consists of $\pi_{\mathsf{DL}} = (R, z)$. The verifier accepts the proof if $Rh^c = g^z$ holds. It can be shown that this proof system satisfies correctness, (knowledge) soundness, and zero-knowledge. Naturally, this proof system can be made non-interactive using the Fiat-Shamir transformation [33].

The following protocol is used to prove that two group elements have the same discrete logarithm with respect to different bases g and h [22]. We use the following protocol that satisfies special soundness and special HVZK:

$$\mathcal{R}_{\mathsf{DLeq}} := \{((g, h, a, b), x) | a = g^x \wedge b = h^x\}. \tag{A.3}$$

The proof system proceeds as follows. The prover holds $P(g, h, a, b, x)$ and the verifier holds $V(g, h, a, b)$.

- The prover samples $r \leftarrow_R \mathsf{F}_p$, and sends the group elements $(R, R') := (g^r, h^r) \in \mathsf{G}^2$ to the verifier.
- The verifier samples a random challenge $c \leftarrow_R \mathsf{F}_p$, and sends it to the prover.
- The prover replies with $z = cx + r \in \mathsf{F}_p$.

The proof π_{DLeq} consists of $(R, R', z) \in \mathsf{G}^2 \times \mathsf{F}_p$. The verifier accepts the proof if $g^z = Ra^c \wedge h^z = R'b^c$ holds.

We also use a generalization of the protocol above for proving linear relations [11] (Section 19.5.3) with special soundness and special HVZK:

$$\mathcal{R}_{\mathsf{LIN}} := \left\{ (\{g_{i,j}, u_i\}_{i \in [m] j \in [n]}; \{x_j\}_{j \in [n]}) \;\middle|\; \begin{array}{l} \prod_{j=1}^{n} g_{1,j}^{x_j} = u_1 \wedge \\ \cdots \\ \wedge \prod_{j=1}^{n} g_{m,j}^{x_j} = u_m \end{array} \right\} \tag{A.4}$$

- The prover samples $\alpha_j \leftarrow_R \mathsf{F}_p$ for $j \in [n]$, sets $v_i := \prod_{j=1}^{n} g_{ij}^{\alpha_j}$ for $i \in [m]$ and sends the elements $\{v_i\}_{i \in [m]}$ to the verifier.
- The verifier samples a random challenge $c \leftarrow_R \mathsf{F}_p$, and sends it to the prover.
- The prover computes $\beta_j = \alpha_j + x_j \cdot c$ for $j \in [n]$ and sends $\{\beta_j\}_{j \in [n]}$ to the verifier.
- The verifier checks that $\forall i \in [m]$, the following equations hold: $\prod_{j=1}^{n} g_{ij}^{\beta_j} = v_i \cdot u_i^c$.

Definition A.4 (Special soundness ([11], Definition 19.4)). . *Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$. We say that (P, V) provides* special soundness *if there is an efficient deterministic algorithm* Ext*, called a witness extractor, with the following property: on input $x \in \mathcal{X}$ and two accepting conversations (t, c, z) and (t, c', z') with $c < c'$, algorithm* Ext *always outputs $w \in \mathcal{W}$ such that $(x, w) \in \mathcal{R}$.*

Definition A.5 (Special Honest Verifier Zero Knowledge (HVZK) ([11], Definition 19.5)). *Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$ with challenge space C. We say that (P, V) is* special honest verifier zero knowledge*, or special HVZK, if there exists an efficient probabilistic algorithm* Sim *that takes as input $(x, c) \in \mathcal{X} \times C$, and satisfies the following properties:*

(i) for all inputs $(x, c) \in \mathcal{X} \times C$, algorithm Sim *always outputs a pair (t, z) such that (t, c, z) is an accepting conversation for x;*

(ii) for all $(x, w) \in \mathcal{R}$, if we compute

$$c \leftarrow_R C, (t, z) \leftarrow \mathsf{Sim}(x, c),$$

then (t, c, z) has the same distribution as that of a transcript of a conversation between $P(x, w)$ and $V(x)$.

The word "special" in the definition means that (i) the simulator may simulate around a given challenge c, and (ii) the simulator produces an accepting conversation even when the statement x does not have a witness.

In Figure 3, we created a VECK protocol with the exponential ElGamal and the KZG commitment scheme that could support arbitrary, *i.e.*, high-entropy data, by using range proofs. Next, we formulate the range-proof relation, necessary to support high-entropy data.

$$\begin{aligned} &\mathcal{R}_{\mathsf{range}} := \{(g_1, h, \{h_i\}, \{\mathsf{ct}_{i,j}\}, h^s);\ (\{\phi_{i,j}\}, s)) \mid \\ &\forall i \in [\ell], j \in [k] : \mathsf{ct}_{i,j} = h_i^s g_1^{\phi_{i,j}} \wedge 0 \leq \phi_{i,j} \text{ Ÿ } \mathcal{B}\}, \end{aligned} \tag{A.5}$$

where $\mathcal{B} \in \mathsf{F}_p$ is a predefined bound. An HVZK instantiation of this proof system can be found in [10] that we apply in our implementation, *cf.* Section 6.

A.3 Danksharding

Danksharding is a scaling solution for the data posted to Ethereum. It enables each Ethereum block to have a capacity of up to 256 data blobs, each blob consisting of a vector of 4096 elements in F_p. The data availability sampling (DAS) method employed by danksharding sets each row of 16 field elements as a single *sample*. This data is organized into a 256×4096 data matrix B.

Danksharding uses a technique called *data availability sampling (DAS)* [42], where no Ethereum validator downloads all of a proposed block but instead verifies its availability by sampling pieces of the data matrix. To ensure that the received samples can be verified and the block can be recovered when a sufficient fraction of the validators receive samples, DAS methods require encoding the data with erasure-code and publishing commitments to the coded data. Towards this goal, the block builder fits a degree (d_x, d_y) bivariate polynomial $f(X, Y)$ to the data matrix such that $f(i, j) = B[i, j]$ for $i = 0, \ldots, 255$ and $j = 0, \ldots, 4095$ for d_x Ÿ 256 and d_y Ÿ 4096. It then expands the matrix B towards the bottom and right to form an extended 512×8192 data matrix E of field elements, where $E[i, j] = f(i, j)$ for $i = 0, \ldots, 511$ and $j = 0, \ldots, 8191$. Finally, it publishes KZG commitments C_i to the univariate polynomials $f_i(Y) = f(i, Y)$, for all $i = 0, \ldots, 255$. These commitments are global knowledge across all Ethereum clients.

Since d_x Ÿ 256, for any given $x \geq 256$, there are constants $\lambda_0(x), \ldots, \lambda_{255}(x)$ depending only on x such that the KZG commitment to $f_x(Y)$, namely C_x, can be computed as $C_x = \prod_{i=0}^{255} \lambda_i(x) C_i$. Hence, the block builder can publish $\overline{C} = (C_0, \ldots, C_{255})$ as the commitment to all of the extended data matrix E.

The DAS method employed by danksharding sets each row of 16 field elements as a single *sample*. Thus, the matrix E consists of 512×512 samples arranged as a square matrix. To distribute the block data to the validators, the builder splits the matrix E

into multiple groups $(P_i)_i$, containing exactly two rows and two columns of samples from E. Each group is sent to a distinct validator, and upon receiving the assigned rows and columns, the validator acts as a data provider to the Ethereum clients.

A.4 Paillier Encryption

In this section, we review the additively homomorphic Paillier public-key encryption scheme [55]. In our application, a distinguishing feature of the Paillier scheme is that it has a low ciphertext expansion, *i.e.*, $|\mathsf{ct}| = 2|m|$, in contrast to the Exponential Elgamal encryption scheme, which for parameters of interest has $|\mathsf{ct}| \approx 8|m|$.

Paillier's public key encryption scheme [55] consists of the following three efficient algorithms:

Gen(1^ℓ): Generates two ℓ-bits safe primes p' and q', and sets the RSA modulus as $N = p'q'$. Samples a uniformly random $x \leftarrow_R \mathbb{Z}_N$, and sets $G := (N+1) \cdot x \mod N^2$ (G has order N in $\mathbb{Z}^*_{N^2}$, *i.e.*, $G^N = 1 \mod N^2$). Sets λ to be Carmichael's lambda function: $\phi = \mathsf{lcm}(p'-1, q'-1)$. Outputs the public key $\mathsf{pk} = (N, G)$ and the secret key $\mathsf{sk} = \lambda$.

Enc(pk, m): To encrypt a message $m \in \mathbb{Z}_N$, randomly chooses $U \leftarrow_R \mathbb{Z}^*_N$ and outputs the ciphertext $c = G^m U^N \mod N^2$.

Dec(sk, c): To decrypt $c \in \mathbb{Z}^*_{N^2}$, computes $m = \frac{L(c^\lambda \mod N^2)}{L(G^\lambda \mod N^2)} \mod N$, where $L(x)$ takes input from $S = \{x < N^2 | x = 1 \mod N\}$ and $L(x) = \frac{x-1}{N}$.

Correctness: The following holds for any $w \in \mathbb{Z}^*_{N^2}$: $w^\lambda = 1 \mod N$ and thus $w^{\lambda N} = 1 \mod N^2$. Hence, $(c^\lambda \mod N^2)$ and $(G^\lambda \mod N^2)$ are equal to 1 when they are raised to the power N; so they are N-th roots of unity, and each of them can be represented as $1 + \beta N \mod N^2$. Then, the function L outputs the value $(\beta \mod N)$. Therefore $L((G^m)^\lambda \mod N^2) = m \cdot L(G^\lambda \mod N^2) \mod N$. Note that $L(G^\lambda \mod N^2) < 0 \mod N$ as otherwise the order of G would be λ which is smaller than N, and this would contradict the way G is chosen in the construction.

Security: The scheme is semantically secure based on the hardness of breaking the DCR assumption.

A.5 Adaptor Signatures

Definition A.6 (Adaptor Signatures). *Consider a signature scheme $\Sigma = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify})$ and a hard relation $\mathcal{R}$. Let $(\mathsf{pk}_\sigma, \mathsf{sk}_\sigma) \leftarrow \textsc{KeyGen}(1^\lambda)$ and $(Y, y) \in \mathcal{R}$. An adaptor signature* Sig *scheme with respect to Σ and Y consists of the following four algorithms* (cf. *[27]):*

$\hat{\sigma} \leftarrow \textsc{pSign}(\mathsf{sk}_\sigma, m, Y)$: *The pre-signing algorithm is a PPT algorithm that takes sk_σ, a message $m \in \{0,1\}^\ell$ and a statement Y, and generates a pre-signature $\hat{\sigma}$.*

$b \leftarrow \textsc{pVerify}(\mathsf{pk}_\sigma, m, \hat{\sigma}, Y)$: *The pre-verification algorithm is a deterministic algorithm that takes pk_σ, a message $m \in \{0,1\}^\ell$, a pre-signature $\hat{\sigma}$, a statement Y, and returns* $0/1$.

$\sigma \leftarrow \textsc{Adapt}(\mathsf{pk}_\sigma, \hat{\sigma}, y)$: *The adapt algorithm is a PPT algorithm that takes pk_σ, a pre-signature $\hat{\sigma}$ and the witness y for the statement Y in $\mathcal{R}$, and generates an* adapted signature σ.

$y \leftarrow \textsc{Extract}(\sigma, \hat{\sigma}, Y)$: *The extract algorithm is a deterministic algorithm that takes an adapted signature σ, a pre-signature $\hat{\sigma}$ and the statement Y, and returns the witness y such that $(Y, y) \in \mathcal{R}$ or $\perp$.*

An adaptor signature scheme must fulfill the following security and privacy requirements:

Definition A.7. *(Security and Privacy for Adaptor Signatures) An adaptor signature scheme $\Xi_{R,\Sigma} = (\textsc{pSign}, \textsc{Adapt}, \textsc{pVerify}, \textsc{Extract})$ for a relation $\mathcal{R}$ and signature scheme Σ satisfies these properties:*

- ***Pre-signature correctness:*** *For $\forall m \in \{0,1\}^*, \forall (Y, y) \in \mathcal{R}$;*

$$\Pr\left[\begin{array}{l|l} \mathsf{pVrfy}(\mathsf{pk}_\sigma, m, Y, \hat{\sigma}) = 1 \wedge & (\mathsf{sk}, \mathsf{pk}_\sigma) \leftarrow \mathsf{Gen}(1^\lambda), \hat{\sigma} \leftarrow \textsc{pSign}(\mathsf{sk}, m, Y) \\ \mathsf{Vrfy}(\mathsf{pk}_\sigma, m, \sigma) = 1 \wedge (Y, y') \in R & \sigma := \mathsf{Adapt}(\mathsf{pk}_\sigma, \hat{\sigma}, y), y' := \mathsf{Ext}(\mathsf{pk}_\sigma, \sigma, \hat{\sigma}, Y) \end{array}\right] = 1$$

- ***Existential unforgeability:*** *The scheme $\Xi_{R,\Sigma}$ is* aEUF-CMA *secure if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, it holds that* $\Pr[\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \sigma(\lambda)$, *where the experiment* $\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}$ *is defined as follows:*

```
aSigForge_{A,Ξ_{R,Σ}}(λ)
  Q := ∅, (sk_σ, pk_σ) ← Gen(λ), (Y, y) ← RelationGen(λ)
  (m, st) ← A_1^{O_S(·),O_pS(·,·)}(pk_σ, Y)
  σ̂ ← pSign(sk, m, Y)
  σ ← A_2^{O_S(·),O_pS(·,·)}(σ̂, st)
  return (m ∉ Q ∧ Vrfy(pk_σ, m, σ))
```

```
O_S(m)                      O_pS(m, Y)
  σ ← Sign(sk, m)             σ̂ ← pSign(sk, m, Y)
  Q := Q ∪ {m}                Q := Q ∪ {m}
  return σ                    return σ̂
```

- ***Pre-signature adaptability:*** *For $\forall m \in \{0,1\}^*, \forall (Y, y) \in \mathcal{R}, \forall \mathsf{pk}_\sigma, \forall \hat{\sigma} \in \{0,1\}^*$ the following probability is* 1*:*

$$\Pr[\mathsf{Vrfy}(\mathsf{pk}_\sigma, m, \textsc{Adapt}(\mathsf{pk}_\sigma, \hat{\sigma}, y)) = 1 | \textsc{pVerify}(\mathsf{pk}_\sigma, m, Y, \hat{\sigma}) = 1]$$

- ***Witness extractability:*** *The scheme $\Xi_{R,\Sigma}$ is witness extractable if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,* $\Pr[\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \sigma(\lambda)$, *where the experiment* $\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}$ *is defined as follows:*

```
aWitExt_{A,Ξ_{R,Σ}}(λ)
  Q := ∅, (sk_σ, pk_σ) ← Gen(λ)
  (m, Y, st) ← A_1^{O_S(·),O_pS(·,·)}(pk_σ)
  σ̂ ← pSign(sk, m, Y)
  σ ← A_2^{O_S(·),O_pS(·,·)}(σ̂, st)
  return ((Y, Ext(pk_σ, σ, σ̂, Y)) ∉ R ∧ m ∉ Q ∧ Vrfy(pk_σ, m, σ))
```

```
O_S(m)                      O_pS(m, Y)
  σ ← Sign(sk, m)             σ̂ ← pSign(sk, m, Y)
  Q := Q ∪ {m}                Q := Q ∪ {m}
  return σ                    return σ̂
```

B THE SCRIPTS OF OUR ON-CHAIN BONDING CONTRACTS

B.1 Bitcoin Script

Algorithm 1 The bonding contract. Here, ⟨client_public_key⟩, ⟨server_public_key⟩ and ⟨adaptor_signature_public_key⟩ denote the client's public key pk_C, the server's public key pk_S and the adaptor signature's public key $\mathsf{pk}_\sigma \cdot \mathsf{vk}$ respectively. Before the timelock expires (set to 1 month here), it allows spending of the funds by any transaction with two signatures that verify with respect to pk_S and $\mathsf{pk}_\sigma \cdot \mathsf{vk}$. After the timelock expires, it allows spending of the funds by any transaction with a signature that verifies with respect to pk_C.

```
OP_IF
    <1 month>
    OP_CHECKLOCKTIMEVERIFY OP_DROP
    <client_public_key>
    OP_CHECKSIGVERIFY
OP_ELSE
    <server_public_key>
    OP_CHECKSIGVERIFY
    <adaptor_signature_public_key>
    OP_CHECKSIGVERIFY
OP_ENDIF
```

B.2 Solidity smart contract

Our Ethereum-based protocols, *cf.* Section 4.2, use smart contracts to achieve fairness and atomicity. We enclose below our smart contract code developed in Solidity for the Ethereum Virtual Machine. The following contract is written for the protocol in Section 5.1. A very similar contract works also for the Paillier-based protocol, *cf.* Section 5.2. For brevity, we omit the elliptic curve cryptography libraries, i.e., BN254 curve arithmetic, from the contract code.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

import { BN254 } from "./BN254.sol";
import { Types } from "./Types.sol";
import { Constants } from "./Constants.sol";

contract FDE is BN254 {

    struct agreedPurchase {
        uint256 timeOut; // The protocol after this timestamp, simply aborts and returns funds.
        uint256 agreedPrice;
        Types.G1Point sellerPubKey;
        bool  secretKeySent;
        bool ongoingPurchase;
    }

    // We assume that for a given seller-buyer pair, there is only a single purchase at any given time
    // Maps seller (server addresses) to buyer (client addresses) which in turn are mapped to tx details
    mapping(address => mapping(address => agreedPurchase)) public orderBook; // Privacy is out of scope for now
    mapping(address => uint256) balances; //stores the Eth balances of sellers

    // Events
    event BroadcastPubKey(address indexed _seller, address indexed _buyer, uint256 _pubKeyX, uint256 _timeOut, uint256 _agreedPrice);
    event BroadcastSecKey(address indexed _seller, address indexed _buyer, uint256 _secKey);

    constructor (
    ) {
    }

    // Agreed price could be set by the contract akin to Uniswap whereby price would be dynamically changing
    // according to a constant product formula given the current number of sellers and buyers (assuming
    // that each tx in the orderBook has the same volume)
    function sellerSendsPubKey(
        uint256 _timeOut,
        uint256 _agreedPrice,
        uint256 _pubKeyX,
        uint256 _pubKeyY,
        address _buyer
    ) public  {
        require(!orderBook[msg.sender][_buyer].ongoingPurchase, "There can only be one purchase per buyer-seller pair!");
        orderBook[msg.sender][_buyer].timeOut = _timeOut;
        orderBook[msg.sender][_buyer].agreedPrice = _agreedPrice;
        Types.G1Point memory _sellerPubKey = Types.G1Point(_pubKeyX, _pubKeyY);
        orderBook[msg.sender][_buyer].sellerPubKey = _sellerPubKey;
        orderBook[msg.sender][_buyer].ongoingPurchase = true;

        emit BroadcastPubKey(msg.sender, _buyer, _pubKeyX, _timeOut, _agreedPrice);
    }

    // If buyer agrees to the details of the purchase, then it locks the corresponding amount of money.
    function buyerLockPayment(
         address _seller
    ) public payable {
        require(!orderBook[_seller][msg.sender].secretKeySent, "Secret keys have been already revealed!");
        require(msg.value == orderBook[_seller][msg.sender].agreedPrice, "The transferred money does not match the agreed price!");
    }

    function sellerSendsSecKey(
        uint256 _secKey,
        address _buyer
    ) public {
        require(!orderBook[msg.sender][_buyer].secretKeySent, "Secret key has been already revealed.");
        require(mul(P1(),_secKey).x == orderBook[msg.sender][_buyer].sellerPubKey.x, "Invalid secret key has been provided by the seller!");
        orderBook[msg.sender][_buyer].secretKeySent = true;
```

```
        balances[msg.sender]+=orderBook[msg.sender][
            _buyer].agreedPrice;
        orderBook[msg.sender][_buyer].ongoingPurchase =
            false;
        // There is no need to store the secret key in
            storage
        emit BroadcastSecKey(msg.sender, _buyer, _secKey)
            ;
    }

    // This function allocates funds to the server from
        previous accrued purchase incomes
    function withdrawPayment(

    ) public {
        payable(msg.sender).transfer(balances[msg.sender
            ]);

        balances[msg.sender]=0;
    }

    // Buyer can withdraw its money if seller does not
        reveal the correct secret key.
    function withdrawPaymentAfterTimout(
        address _seller
    ) public {
        require(!orderBook[_seller][msg.sender].
            secretKeySent, "The encryption secret key
            has already been sent by the seller!");
        require(block.timestamp >= orderBook[_seller][msg
            .sender].timeOut, "The seller has still time
             to provide the encryption secret key!");
        orderBook[_seller][msg.sender].ongoingPurchase =
            false;
        payable(msg.sender).transfer(orderBook[_seller][
            msg.sender].agreedPrice);
    }
}
```

C PROOFS

C.1 Proof of security of BDE based on VECK's security, Theorem 4.1

Proof of Theorem 4.1. Throughout the proof, we assume that $\text{FDE.S}\textsc{etup}(1^\lambda) \rightarrow \mathsf{pp}$ and

$$\text{FDE.C}\textsc{om}\langle \mathcal{C}(\mathsf{data}), \mathcal{S}()\rangle \rightarrow \langle \mathcal{C}(C_\phi), \mathcal{S}(\mathsf{data})\rangle.$$

We first show **correctness** (Definition 4.2). When both client $\mathcal{C}$ and server $\mathcal{S}$ are honest, $\mathcal{S}$ runs $\textsc{Enc}(C_\phi, \mathsf{data})$ to generate $(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$. By the correctness of the VECK protocol (Definition 3.1), with probability 1, it holds that $\textsc{Ver}_{\mathsf{ct}}(C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) = 1$ and $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$. Then, before the timelock expires, $\mathcal{C}$ locks tk tokens in the bonding contract on Ethereum, and in the case of Bitcoin, creates a bonding contract and sends a pre-signature to $\mathcal{S}$. By the pre-signature correctness of the adaptor signature scheme, $\mathcal{C}$'s signature is verified by $\mathcal{S}$. Afterwards, $\mathcal{S}$ posts the decryption key sk to the bonding contract on Ethereum, upon which the contract sends tk tokens to $\mathcal{S}$ within finite time, as Ethereum is safe and live, $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$, and the timelock has not expired yet. Similarly, $\mathcal{S}$ adapts the pre-signature with sk and posts the signed spending transaction tx to Bitcoin, after which it receives tk tokens within finite time, as Bitcoin is safe and live, the adaptor signature scheme satisfies pre-signature adaptability, $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$, and the timelock has not expired yet. Finally, $\mathcal{C}$ reads sk either from the bonding contract, or from the adaptor signature (by witness extractibility), decrypts ct using sk and obtains the data. Thus, with probability 1, $\mathcal{C}$ receives the data, and $\mathcal{S}$ receives tk tokens.

We next show **client-fairness** (Def. 4.3). Consider an FDE protocol instance between an honest client $\mathcal{C}$ and a PPT server $\mathcal{S}^*$, $\langle \mathcal{C}(\mathsf{data}'), \mathcal{S}(\mathsf{tk}')\rangle \leftarrow \text{FDE.Exc}\langle \mathcal{C}(C_\phi, p), \mathcal{S}^*(C_\phi, \mathsf{data})\rangle$ such that tk' ¡ 0. Since $\mathcal{S}^*$ receives a positive payment tk' ¡ 0, in the case of Ethereum, $\mathcal{C}$ must have posted tk tokens to the bonding contract. Thus, by the existential unforgeability of $\mathcal{C}$'s signatures, $\mathcal{C}$ must have verified the proof π returned by $\mathcal{S}^*$, *i.e.*, $\textsc{Ver}_{\mathsf{ct}}(C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) = 1$, except with negligible probability. Similarly, in the case of Bitcoin, by the existential unforgeability of the adaptor signatures, $\mathcal{C}$ must have sent a presignature to $\mathcal{S}^*$, *i.e.*, $\textsc{Ver}_{\mathsf{ct}}(C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) = 1$, except with negligible probability. Since the bonding contract allows $\mathcal{S}^*$ to receive positive amount of tokens, Bitcoin and Ethereum are safe and live, and the adaptor signatures satisfy existential unforgeability, $\mathcal{S}^*$ must have posted a decryption key sk or a signature adapted with sk, to the bonding contract such that $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$.

Now, for contradiction, assume that given an honest client $\mathcal{C}$, there exists a PPT $\mathcal{S}^*$ such that the following probability is *not* negligible in λ:

$$\Pr\left[\begin{array}{l|r} \text{FDE.V}\textsc{rfy}(\mathsf{data}', \mathsf{com}) = 0 & \mathsf{pp} \leftarrow \text{FDE.S}\textsc{etup}(1^\lambda) \\ \quad\wedge\ \mathsf{tk}' \text{ ¡ } 0 & \langle \mathcal{C}(\mathsf{com}), \mathcal{S}(\mathsf{data})\rangle \leftarrow \text{FDE.C}\textsc{om}\langle \mathcal{C}(\mathsf{data}), \mathcal{S}()\rangle \\ & \text{Inputs: } \mathcal{C}(\mathsf{com}, \mathsf{tk}), \mathcal{S}^*(\mathsf{com}, \mathsf{data}) \\ & \langle \mathcal{C}(\mathsf{data}'), \mathcal{S}^*(\mathsf{tk}')\rangle \leftarrow \text{FDE.Exc}\langle \mathcal{C}, \mathcal{S}^*\rangle \end{array}\right]$$

However, $\text{FDE.V}\textsc{rfy}(\mathsf{data}', \mathsf{com}) = 0$ implies that $\textsc{Dec}(\mathsf{sk}, \mathsf{ct}) = y < \mathsf{data} = F(\phi(\cdot)) = F(w)$, where the witness w is the polynomial $\phi(\cdot)$ and $F(.)$ outputs the sequence of its evaluations. Consequently, the following probability is not negligible in λ either:

$$\Pr\left[\begin{array}{c|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ \textsc{Ver}_{\mathsf{ct}}(C_w, \mathsf{vk}, \mathsf{ct}, \pi) = 1 \wedge & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ \textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1 \wedge & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ y < F(w) & (\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi) \leftarrow \mathcal{A}(\mathsf{pp}, C_w) \\ & y \leftarrow \textsc{Dec}(\mathsf{sk}, \mathsf{ct}) \end{array}\right]$$

However, this contradicts the security of the VECK protocol (Definition 3.1).

We last show **server-fairness** (Def. 4.4). Consider an FDE protocol instance between an honest server $\mathcal{S}$ and a PPT client $\mathcal{C}^*$, $\langle \mathcal{C}(\mathsf{data}'), \mathcal{S}(\mathsf{tk}')\rangle \leftarrow \text{FDE.Exc}\langle \mathcal{C}(C_\phi, \mathsf{tk}), \mathcal{S}(C_\phi, \mathsf{data})\rangle$ such that tk' Ÿ tk. Note that the server $\mathcal{S}$ posts sk to the bonding contract only if $\mathcal{C}^*$ locks tk tokens, in which case it would have received tk tokens. As $\mathcal{S}$ receives only tk' Ÿ tk, it must be that $\mathcal{S}$ has not posted sk. Similarly, in the case of Bitcoin, it would have posted tx with an adapted signature and its signature, only if $\mathcal{C}^*$ has created a bonding contract that allows the spending of tk tokens by a transaction with the adaptor and the server's signature. In this case, $\mathcal{S}$ would have received tk tokens by the existential unforgeability of its signature scheme. Consequently, as $\mathcal{S}$ receives only tk' Ÿ tk, it must be that it has not sent an adapted signature. This implies that the only information sent by $\mathcal{S}$ consists of a verification key (vk), the ciphertexts (ct) and a proof (π) that verifies.

Now, by the computational zero-knowledge property of the VECK scheme (Definition 3.1), for any PPT $\mathcal{C}^*$, there exists a PPT simulator Sim^{C^*} with oracle access to C^* such that there is a negligible function $\mu(\cdot)$ for which the following probability is less than $\frac{1}{2} + \mu(\lambda)$:

$$\Pr\left[\mathcal{A}(\mathsf{pp}, C_w, \mathsf{vk}_b, \mathsf{ct}_b, \pi_b) = b \quad \begin{array}{l} \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ (\mathsf{vk}_0, \mathsf{sk}_0, \mathsf{ct}_0, \pi_0) \\ \qquad\quad \textsc{Enc}(\mathsf{pp}, C_w, w) \\ (\mathsf{vk}_1, \mathsf{ct}_1, \pi_1) \leftarrow \mathsf{Sim}^{C^*}(\mathsf{pp}, C_w) \\ b \leftarrow_R \{0,1\} \end{array}\right]$$

Finally, consider the PPT simulator Sim with oracle access to C^* that simulates vk, ct, and π using the simulator Sim^{C^*} above and equipped with them, simulates a trace α_1 for the interaction of C^* and $\mathcal{S}$ with $\mathsf{tk}' \ddot{Y} \mathsf{tk}$. Then, using the bound on the probability above and the fact that $\mathcal{S}$ only sends a verification key vk, ciphertexts ct and a verifying proof π when $\mathsf{tk}' \ddot{Y} \mathsf{tk}$, we can conclude that the following probability is less than $\frac{1}{2} + \mu(\lambda)$ (otherwise, the probability above would not be less than $\frac{1}{2} + \mu(\lambda)$, implying a contradiction):

$$\Pr\left[C^*(\mathsf{com}, \alpha_b) = b \quad \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{FDE}.\textsc{Setup}(1^\lambda) \\ \langle C(\mathsf{com}), \mathcal{S}(\mathsf{data})\rangle \leftarrow \mathsf{FDE}.\textsc{Com}\langle C(\mathsf{data}), \mathcal{S}()\rangle \\ \text{Inputs: } C^*(\mathsf{com}, \mathsf{tk}), \mathcal{S}(\mathsf{com}, \mathsf{data}) \\ \alpha_0 \leftarrow \mathsf{tr}(\langle C^*, \mathcal{S}\rangle \leftarrow \mathsf{FDE}.\textsc{Exc}\langle C^*, \mathcal{S}\rangle) \\ \text{Outputs: } C^*(\mathsf{data}'), \mathcal{S}(\mathsf{tk}') \text{ s.t. } \mathsf{tk}' \ddot{Y} \mathsf{tk} \\ \alpha_1 \leftarrow \mathsf{Sim}^{C^*}(\mathsf{pp}, \mathsf{com}, \mathsf{tk}) \\ b \xleftarrow{R} \{0,1\} \end{array}\right]$$

Since this holds for any honest server $\mathcal{S}$ and all PPT C^*, this concludes the proof of server-fairness.

C.2 Proof of security for ElGamal-based VECK, Theorem 5.1

In Section 5.1, we gave the intuition behind the proofs; here, we elaborate and give more details.

Proof. **Correctness.** The proof π_α verifies correctly due to the correctness of the KZG commitment scheme. The Chaum-Pedersen discrete logarithm equality proof [22] π_{DLEq} for the quadruple $(Q, Q^*, g_1, \mathsf{vk})$ verifies correctly since $\mathsf{vk} = g^s$, and the following holds

$$Q^* = \frac{\prod_{i=0} \mathsf{ct}_i^{L_{i,S}(\alpha)}}{C_\alpha} = \frac{\prod_{i=0} (h_i^s \cdot g^{\phi(i)})^{L_{i,\ell}(\alpha)}}{C_\alpha} = \frac{Q^s \prod_{i=0} g^{\phi(i) L_{i,\ell}(\alpha)}}{g^{\phi(\alpha)}} = Q^s.$$

The π_{LIN} proof for the linear relation verifies correctly since $C_\alpha = (g_1)^{\phi(\alpha)} \left(g_1^{(\tau-\alpha)}\right)^s$ and $\mathsf{vk} = (1)^{\phi(\alpha)} \cdot (h)^s$. It is easy to see that $\textsc{Ver}_{\mathsf{key}}$ would trivially accept and $\textsc{Dec}$ would output $\phi(i)$ as expected.

Soundness. Let C_ϕ be a commitment to polynomial $\phi(X) \in \mathsf{F}_p[X]$ of degree ℓ. Suppose there is a PPT adversary $\mathcal{A}$ that breaks the security, i.e., with non-negligible probability, the adversary generates $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct} = [\mathsf{ct}_0, \dots, \mathsf{ct}_\ell], \pi = (C_\alpha, \pi_\alpha, \pi_{\mathsf{LIN}}, \pi_{\mathsf{DLEQ}}))$, s.t. the key and ciphertext verifications are successful, yet the decryption of ct with sk outputs $(y_0, \dots y_\ell) < (\phi(0), \dots, \phi(\ell))$. Successful key verification $\textsc{Ver}_{\mathsf{key}}$ implies that for $\mathsf{sk} = s$, $\mathsf{vk} = h^s$. Furthermore, since the ciphertext verification $\textsc{Ver}_{\mathsf{ct}}$ also holds, we have:

$$e(C_\phi / C_\alpha, g_2) = e(\pi_\alpha, g_2^{\tau - \alpha}) \tag{C.1}$$

$$\prod_{i=0} \left(h_i^{L_{i,\ell}(\alpha)}\right)^s \cdot g_1^{-s(\tau-\alpha)} = \prod_{i=0} \mathsf{ct}_i^{L_{i,\ell}(\alpha)} / C_\alpha \tag{C.2}$$

Since π_{LIN} is valid, the adversary must know some value w such that $C_\alpha = g_1^{w + (\tau-\alpha)s}$ (this helps guarantee that w does not depend on τ). Furthermore, in the algebraic group model (AGM), without loss of generality, we can assume that $\pi_\alpha = g_1^{u(\tau)}$, where $u(\tau)$ is a polynomial of degree at most ℓ over the τ. Therefore, equation C.1 becomes:

$$e(g_1^{\phi(\tau) - w - s(\tau-\alpha)}, g_2) = e(g_1^{u(\tau)}, g_2^{\tau-\alpha}) \tag{C.3}$$

Assuming AGM, we must then have the following identity:

$$\phi(x) - w - s(x - \alpha) = u(x) \cdot (x - \alpha) \tag{C.4}$$

We thus deduce that $w = \phi(\alpha)$, and $C_\alpha = g_1^{\phi(\alpha) + (\tau-\alpha)s}$. Therefore, if $\phi(X)$ is of degree ℓ, equation C.2 becomes:

$$\prod_{i=0} \left(h_i^{L_{i,\ell}(\alpha)}\right)^s \cdot g_1^{-s(\tau-\alpha)} = \prod_{i=0} \mathsf{ct}_i^{L_{i,\ell}(\alpha)} / \left(g_1^{(\tau-\alpha)s} \cdot g_1^{\phi(\alpha)}\right) \Leftrightarrow$$

$$g_1^{\phi(\alpha)} = \prod_{i=0} (\mathsf{ct}_i / h_i^s)^{L_{i,\ell}(\alpha)} \tag{C.5}$$

Finally, we show that $\phi(X)$ is of degree ℓ. Let's denote by $\Phi_i = \mathsf{ct}_i / h_i^s \in G_1$, and define the degree ℓ polynomial $\Phi(x) \in G_1[x]$, where $\Phi(i) = \Phi_i$ for $i \in [0, \ell]$. Rewriting Equation (C.5), we have $g_1^{\phi(\alpha)} = \Phi(\alpha)$. Since the polynomials $g_1^{\phi(x)}$ and $\Phi(x)$ agree on a random point α, and the degree of $g_1^{\phi(x)}, \Phi(x)$ is at most $\max(n, \ell) = n$, by Schwartz-Zippel lemma, if the two polynomials are not the same, then they agree at a random point with all but n/p probability. Therefore if the verifier accepts with probability greater than $1 - n/p$, we must have that $g_1^{\phi(x)}, \Phi(x)$ are the same polynomial. As a result, we have $\Phi(i) = g_1^{\phi(i)}$ for all $i \in [0, \ell]$, and both polynomials have degree exactly ℓ. This implies $\mathsf{ct}_i = g_1^{\phi(i)} \cdot h_i^s$ is indeed the correct encryption for the full evaluations of $\phi(x)$ over full evaluation domain $i \in [0, \ell]$.

Computational Zero-Knowledge We build an efficient simulator that takes as input: $(h, \{h_i\}, C_\phi, \alpha, \mathsf{crs})$ and outputs indistinguishably distributed tuple $(\mathsf{vk}, \{\mathsf{ct}_i\}, \pi = (C_\alpha, \pi_\alpha, \pi_{\mathsf{DLeq}}, \pi_{\mathsf{LIN}}))$ without the knowledge of ϕ. We start by sampling ℓ random coefficients $\beta_1, \dots, \beta_\ell \leftarrow_R \mathsf{F}_p$, and $\forall i \in [\ell]$, we set $B_i := g_1^{\beta_i}$. We sample a random $s \leftarrow_R \mathsf{F}_p$ and set $\mathsf{vk} := h^s$. We generate the proofs in a way similar to honest execution, but for a polynomial $\tilde{\phi}(x) = \beta_\ell x^\ell + \dots \beta_1 x + \beta_0$, where we only know $B_0 = g_1^{\beta_0}$ and not the value in the exponent (β_0); B_0 is generated so that $\tilde{\phi}$ commits to the same value as given to the simulator: $g_1^{\tilde{\phi}(\tau)} = C_\phi$, for which we set $B_0 := C_\phi / g_1^{\beta_\ell \tau^\ell + \ldots + \beta_1 \tau}$. For all $i \in [\ell]$, we compute

$$\begin{aligned}
\mathsf{ct}_i :=& h_i^s \cdot C_\phi \cdot g_1^{\beta_\ell (i^\ell - \tau^\ell) + \ldots + \beta_1 (i - \tau)} = h_i^s \cdot g_1^{\hat{\phi}(i)} \\
C_\alpha :=& \left(g_1^{\tau - \alpha}\right)^s \cdot C_\phi \cdot g_1^{\beta_\ell (\alpha^\ell - \tau^\ell) + \ldots + \beta_1 (\alpha - \tau)} = g_1^{\hat{\phi}(\alpha) + s(\tau - \alpha)} \\
\pi_\alpha :=& g_1^{-s} \cdot g_1^{(\beta_\ell (\tau^\ell - \alpha^\ell) + \ldots + \beta_1 (\tau - \alpha))/(\tau - \alpha)} = g_1^{-s} \cdot g_1^{(\hat{\phi}(\tau) - \hat{\phi}(\alpha))/(\tau - \alpha)}
\end{aligned}$$

The verification equations are satisfied. Note that the values that involve τ are computed using the group elements in the crs, since the simulator does not know the trapdoor τ. The π_{DLeq} proof is computed the same way as in the honest execution. The proof π_{LIN} is simulated since we do not know the witness, yet the respective statement is in the language (has the witness). Therefore, its distribution is indistinguishable from the real one. We note that $C_\alpha = C_\phi \cdot \pi_\alpha^{-(\tau - \alpha)}$ is uniquely determined by π_α in both the real and our simulated executions. Hence, the correct distribution of π_α implies the correct distribution for C_α. Next, the DDH assumption in G_1 guarantees that given the generators $(h, \{h_i\}, g_1)$, the distribution of $(h^s, \{(h_i)^s\}, g_1^s)$ is indistinguishable from random both in the simulated and in the real executions, even given π_{DLeq} which could be simulated and would verify correctly even when there is no witness s (which is due to its special-HVZK property).

C.3 Proof of security for Paillier-based VECK, Theorem 5.2

Proof of Theorem 5.2. We show that the protocol described in Figure 4 satisfies correctness, security and zero-knowledge as defined in Definition 3.1.

Correctness. Correctness of decryption and key verification is due to the correctness of the Paillier cryptosystem. The ciphertext verification could fail when either of the z values fall out of the range $[0, A)$. For our protocol, the probability that a single z_i falls out of the range $[0, A)$ is smaller than $(p \cdot B/A)$. Then, the probability of failure due to at least one $z_i \geq A$ being out of range is smaller than $1 - (1 - (p \cdot B/A))^\ell \leq \frac{\ell \cdot p \cdot B}{A} \leq \frac{1}{2^\lambda}$, which is negligible in λ.

Soundness. Suppose there is a PPT adversary $\mathcal{A}$ that breaks the security, *i.e.*, with non-negligible probability, the adversary generates $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct} = (\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell), \pi = (c, W_0, \ldots, W_\ell, z_0, \ldots, z_\ell))$, s.t. the key and ciphertext verifications are successful, yet the decryption outputs an invalid witness. Successful key verification $\textsc{Ver}_{\mathsf{key}}$ implies that for $\mathsf{sk} = (p', q', \mu)$, $\mathsf{vk} = N = p'q'$.

For any value of $g \in \mathsf{Z}_{N^2}^*$ whose order is a non-zero multiple of N the following function is bijective (see Lemma 1, [55]) $\mathcal{E}_g : \mathsf{Z}_N \times \mathsf{Z}_N^* \to \mathsf{Z}_{N^2}^*$, defined as $\mathcal{E}_g : (x, y) \to g^x y^N \mod N^2$. We use $g = (N+1)$, since $(N+1)^N = 1 \mod N^2$. Given a factorization of $N = p' \cdot q'$ or given a Carmichael's lambda function of N: $\mu = \mathsf{lcm}(p', q')$, it is easy to invert $\mathcal{E}_g$ to recover x.

We will now show that if the ciphertext verification $\textsc{Ver}_{\mathsf{ct}}$ holds, the decryption procedure will recover correct values of $\phi(i)$ of the committed polynomial C_ϕ. Without loss of generality we can assume that there exists $x_i \in \mathsf{Z}_N, U_i \in \mathsf{Z}_N^*$, s.t. $\mathsf{ct}_i = (N+1)^{x_i} U_i^N$. We run the adversary twice in order to obtain two proofs $(c, W_0, \ldots, W_\ell, z_0, \ldots, z_\ell)$ and $(c', W_0', \ldots, W_\ell', z_0', \ldots, z_\ell')$ that pass the verification with non-negligible probability. We then obtain the following equations:

$$\begin{aligned}
1 &= (N+1)^{z_i - z_i'} (W_i / W_i')^N \mathsf{ct}_i^{-c+c'} = \\
&= (N+1)^{(z_i - z_i') - (c - c')x_i} (U_i^{-c+c'} W_i / W_i')^N \mod N^2 \\
1 &= \prod_{i \in [\ell]} \mathsf{crs}_i^{z_i - z_i'} / C_\phi^{c - c'} = \prod_{i \in [\ell]} \mathsf{crs}_i^{(z_i - z_i') - \phi(i)(c - c')} \in \mathsf{G}_1
\end{aligned}$$

In the AGM, it follows that the equations below must hold for some $\Delta z_i = z_i - z_i'$, $\Delta z_i \in [-A, A)$ and $\Delta c_i = c - c'$, $\Delta c_i \in [-B, B)$:

$$\Delta z_i = \Delta c_i \cdot x_i \mod N \tag{C.6}$$

$$\Delta z_i = \Delta c_i \cdot \phi(i) \mod p \tag{C.7}$$

By construction, if $x_i < \phi(i)$ for some i (which is checked by recommitting to the decrypted vector), the lattice reduction algorithm is run for each $i \in [\ell]$ to find (σ_i, ν_i) – the shortest vector in the integer lattice with basis $((N, 0), (x_i, 1))$, and the decryption outputs $x_i' = \sigma_i / \nu_i \mod p$. We now argue why $\sigma_i / \nu_i = \Delta z_i / \Delta c_i \mod p$, and hence why the decryption procedure outputs the correct evaluations: $x_i' = \phi(i)$. We follow Poupard and Stern [59] approach (see proof of Theorem 1), where the inner product for the Gauss algorithm is defined to be $(x, y) \cdot (x', y') \quad xx' + A^2/B^2 \times yy'$, the norm is defined in the standard manner: $||(x, y)|| = \sqrt{(x, y) \cdot (x, y)}$. The unknown vector $(\Delta z_i, \Delta c_i)$ is also in the lattice according to equation Equation (C.6), therefore, since it might not be the shortest, $||(\sigma_i, \nu_i)|| \leq ||(\Delta z_i, \Delta c_i)|| \leq \sqrt{A^2 + A^2/B^2 \times B^2} = \sqrt{2}A$. From the definition of inner product, $|\sigma_i| \leq ||(\sigma_i, \nu_i)||$, hence $|\sigma_i| \leq \sqrt{2}A$. Also from the definition $|\nu_i| \leq \frac{B}{A} ||(\sigma_i, \nu_i)||$, therefore $|\nu_i| \leq \sqrt{2}B$.

$\Delta c_i \cdot x_i - \Delta z_i = 0 \mod N$ and $\nu_i \cdot x_i - \sigma_i = 0 \mod N$, therefore $\nu_i \cdot \Delta z_i = \Delta c_i \cdot \sigma_i \mod N$. Since $|\nu_i \cdot \Delta z_i - \Delta c_i \cdot \sigma_i| \leq |\nu_i||\Delta z_i| + |\Delta c_i||\sigma_i| \leq 2\sqrt{2}AB$ and $N \geq 2\sqrt{2}AB$, then the equation must hold over integers: $\nu_i \cdot \Delta z_i = \Delta c_i \cdot \sigma_i$ in Z. The since p is prime, the equation must also hold modulo p: $\nu_i \cdot \Delta z_i = \Delta c_i \cdot \sigma_i \mod p$. Therefore $\sigma_i / \nu_i = \Delta z_i / \Delta c_i = \phi(i) \mod p$. Hence the decryption would correctly output committed values.

Statistical Zero-knowledge. We build a simulator that takes as input the commitment C_ϕ and the public parameter crs. The simulator produces the transcript of the interaction by first generating a correct Paillier public key vk, then choosing random $c \leftarrow_R [0, B)$, $\mathsf{ct}_i \leftarrow_R N^2$, $z_i \leftarrow_R [0, A)$ and $W_i \leftarrow_R \mathsf{Z}_N^*$ for all $i \in [\ell]$, and finally programming the random oracle as $\mathsf{H}(\mathsf{vk}, \{\mathsf{ct}_i\}_{i=1}^\ell, \{(N+1)^{z_i} W_i^N \mathsf{ct}_i^{-c}\}_{i=1}^\ell, C^c / \prod_{i=1}^\ell \mathsf{crs}[i]^{z_i}) := c$. The simulator outputs vk as the verification key, ct_i, $i \in [n]$, as the ciphertexts and $\pi = (c, W_1, \ldots, W_\ell, z_1, \ldots, z_\ell)$ as the simulated proof. For $\frac{pB\ell}{A} \leq \mathsf{negl}(\lambda)$, the distribution of this transcript is statistically indistinguishable from the real transcript (since the probability that any z_i is not in $[0, A)$ in the real execution is negligible in λ).

C.4 Proof of security of VECK for subsets, Theorem 5.3

Correctness holds by construction.
Soundness. Let C_ϕ be the commitment to polynomial $\phi(X) \in \mathsf{F}_p[X]$ of degree ℓ and let $S \subseteq \mathsf{F}_p$ and $|S| \leq \ell + 1$. Suppose there is a PPT adversary $\mathcal{A}$ that breaks the security, *i.e.*, with non-negligible probability, the adversary generates $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi = (C_S, \pi_S, \pi'))$, s.t.

the key and ciphertext verifications are successful, yet the decryption outputs an invalid value. Here, $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi')$ correspond to the output of $\textsc{Enc}(F_S^{\text{full-eval}}, C_S, \phi_S(X))$. Since the ciphertext verification $\textsc{Ver}_{\mathsf{ct}}$ holds, we have:

$$e(C_\phi/C_S, g_2) = e(\pi_S, g_2^{V_S(\tau)}) \tag{C.8}$$

In the algebraic group model, without loss of generality, we can assume that $\pi_S = g_1^{u(\tau)}$, $C_\phi = g_1^{\phi(\tau)}$ and $C_S = g_1^{v(\tau)}$, where $u(x), \phi(x), v(x)$ are polynomials of degree at most n. Therefore, Equation (C.8) becomes:

$$e(g_1^{\phi(\tau)-v(\tau)}, g_2) = e(g_1^{u(\tau)}, g_2^{V_S(\tau)}) \tag{C.9}$$

Assuming AGM, and the fact that the adversary has to satisfy Equation (C.9) without knowing τ, we must have the following identity:

$$\phi(x) - v(x) = u(x) \cdot \prod_{i \in S} (x - i) \tag{C.10}$$

Therefore, $\forall i \in S : v(i) = \phi(i)$. Finally, since we apply a full-opening VECK on the commitment $C_S = g_1^{v(\tau)}$, we must have the encryptions correctly encrypt the values $\phi(i)$ over the subset S.

Computational Zero-Knowledge. We build an efficient simulator that takes as input pp and outputs indistinguishably distributed tuple $(\mathsf{vk}, \mathsf{ct}, \pi)$ without the knowledge of $\phi(X)$. We simulate by sampling a random $y \leftarrow_R \mathbb{F}_p$ and setting $C_{\phi_S} := C_\phi / g_1^{yV_S(\tau)}$ and $\pi_S := g_1^y$. Note that there is a one-to-one relationship between C_{ϕ_S} and π_S that guarantees acceptance by the verifier: $C_{\phi_S} = C_\phi \pi_S^{-V_S(\tau)}$, so the correct distribution on π_S implies correct distribution on C_{ϕ_S}. We note that in the honest execution $\pi_S = g_1^t \cdot g_1^{(\phi(\tau) - \sum_{i \in S} \phi(i) L_{i,S}(\tau))/V_S(\tau)}$, and in here in the simulated execution $\pi_S = g_1^y$. Since both t and y are sampled uniformly at random, the distributions are equivalent.

C.5 Proof of security of Multi-Client VECK, Theorem 7.1

Proof of Theorem 7.1. We prove correctness, soundness and computational $\mathcal{L}$-bits zero-knowledge for the MC-VECK protocol in Section 7.3, Figure 7.

Correctness: When $\textsc{Prep}(C_\phi, \phi) = \textsc{Enc}(C_\phi, \phi) \to (\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi)$ and $D_C = h^{\delta_C}$, $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C/D_C, \mathsf{ct}, \pi)$ returns 1 by the correctness of the VECK protocol of Section 5.1, and $\mathsf{vk}_C = h^{\mathsf{sk}+\delta_C} = h^{\mathsf{sk}_C}$. Moreover, given $h_{C,i} = (h_i^{\delta_C})_{i \in [n]}$, the discrete logarithm equality proof π_{DLEq} verifies for $(Q := \prod_{i \in [n]} h_i^{e_i}, Q^* := \prod_{i \in [n]} h_{C,i}^{e_i}, h, D_C)$ for any $(e_i)_{i \in [n]} \in \mathbb{Z}_p^n$ by the correctness of [22]. Therefore, verification for honestly generated keys and ciphertexts will always succeed.

Soundness: Consider an adversary $\mathcal{A}(C_\phi) \to (\mathsf{sk}_C, \mathsf{vk}_C, \mathsf{ct}, \pi_C)$ for which $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C, \mathsf{ct}, \pi_C) = 1$ and $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}_C, \mathsf{sk}_C) = 1$, yet $F(\phi) < \textsc{Dec}(\mathsf{sk}_C, \mathsf{ct})$ for $F := F_{[n]}^{\text{full-eval}}$. Let $\pi_C = (\pi, D_C, \pi_{\mathsf{DLEq}}, (h_{C,i})_{i \in [n]})$. Let sk denote the discrete logarithm between vk_C/D_C and h (thus, by definition, $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}_C/D_C, \mathsf{sk}) = 1$), and δ_C the discrete logarithm between D_C and h. Note that $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C, \mathsf{ct}, \pi_C) = 1$ implies (i) π_{DLEq} verifies against $(Q := \prod_{i \in [n]} h_i^{e_i}, Q^* := \prod_{i \in [n]} h_{C,i}^{e_i}, h, D_C)$ for $(e_i)_{i \in [n]} = (H(D_C, i))_{i \in [n]}$, and (ii) $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C/D_C, \mathsf{ct}, \pi) = 1$. Then, by (i) and the security of the scheme in [22],

$$\prod_{i \in [n]} h_{C,i}^{e_i} = \prod_{i \in n} h_i^{\delta_C e_i}$$

for $(e_i)_{i \in [n]} = (H(D_C, i))_{i \in [n]}$, where $H(.)$ is modelled as a random oracle. In the Random Oracle model, this equality must hold for a random $(e_i)_{i \in [n]} \leftarrow_R \mathbb{Z}_p^n$, with all but $1/p$ probability, and we must have $h_{C,i} = h_i^{\delta_C}$ for all $i \in [n]$. Since $\mathsf{vk}_C = h^{\mathsf{sk}+\delta_C}$, $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}_C, \mathsf{sk}_C) = 1$ implies $\mathsf{sk}_C = \mathsf{sk} + \delta_C \mod p$. Therefore, if $\textsc{Dec}(\mathsf{sk}_C, (\mathsf{ct}_i, h_{C,i})_{i \in [n]})$ succeeds, it outputs $(m_i)_{i \in [n]}$ such that

$$g_1^{m_i} = \mathsf{ct}_i \cdot h_{C,i}/h_i^{\mathsf{sk}+\delta_C} = \mathsf{ct}_i \cdot h_i^{\delta_C}/h_i^{\mathsf{sk}+\delta_C} = \mathsf{ct}_i/h_i^{\mathsf{sk}},$$

which implies $\textsc{Dec}(\mathsf{sk}_C, \mathsf{ct}_C) = \textsc{Dec}(\mathsf{sk}, \mathsf{ct})$. As $F(\phi) < \textsc{Dec}(\mathsf{sk}_C, \mathsf{ct})$, it holds that $F(\phi) < \textsc{Dec}(\mathsf{sk}, \mathsf{ct})$.

Finally, for contradiction, suppose the following probability is not negligible for the MC-VECK protocol:

$$\Pr\left[\begin{array}{l|l} \textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}_C, \mathsf{ct}, \pi_C) = 1 \land & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ \textsc{Ver}_{\mathsf{key}}(\mathsf{vk}_C, \mathsf{sk}_C) = 1 \land & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, \phi) \\ y < F(\phi) & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ & (\mathsf{sk}_C, \mathsf{vk}_C, \mathsf{ct}, \pi_C) \\ & \qquad \mathcal{A}(\mathsf{pp}, F, C_\phi) \\ & y \leftarrow \textsc{Dec}(\mathsf{sk}, \mathsf{ct}) \end{array}\right]$$

Then, for $(\mathsf{sk}, \mathsf{vk}, \mathsf{ct}, \pi) \leftarrow \mathcal{A}'(C_\phi)$ that outputs a subset of $\mathcal{A}$'s output, it holds that $\textsc{Ver}_{\mathsf{ct}}(F, C_\phi, \mathsf{vk}, \mathsf{ct}, \pi) = 1$, $\textsc{Ver}_{\mathsf{key}}(\mathsf{vk}, \mathsf{sk}) = 1$ and $F(\phi) < \textsc{Dec}(\mathsf{sk}, \mathsf{ct})$ with non-negligible probability in λ. However, this conflicts with the security of the VECK protocol of Section 5.1.

Computational $\mathcal{L}$-bits zero-knowledge: To prove computational $\mathcal{L}$-bits zero-knowledge for the MC-VECK protocol, we first observe that the distributions

$$\{(\mathsf{aux}, \mathsf{msk}) | (\mathsf{aux}, \mathsf{msk}) \leftarrow \textsc{Prep}(F, C_w, w)\}$$

$$\{(\mathsf{vk}, \mathsf{ct}, \pi, \mathsf{sk}) | (\mathsf{vk}_*, \mathsf{sk}_*, \mathsf{ct}_*, \pi_*) \leftarrow \textsc{Enc}(F, \mathsf{aux}, \mathsf{msk})\}$$

are statistically indistinguishable. Thus, computational $\mathcal{L}$-bits zero-knowledge for the original VECK protocol (defined below) implies computational $\mathcal{L}$-bits zero-knowledge for the MC-VECK protocols: *Computational $\mathcal{L}$-bits zero-knowledge for the VECK protocols:* Consider a normal VECK protocol $\Pi' = (\textsc{Gen}, \textsc{Enc}', \textsc{Ver}_{\mathsf{ct}}, \textsc{Ver}_{\mathsf{key}}, \textsc{Dec})$. We say that it satisfies computational $\mathcal{L}$-bits zero-knowledge, if for any PPT algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a PPT simulator $\Pi \cdot \mathsf{Sim}$ such that there is a negligible function $\mu(\cdot)$, s.t. for all $w \in \mathcal{W}, \forall F \in \mathcal{F}$ the following probability is less than $1/2 + \mu(\lambda)$:

$$\Pr\left[\begin{array}{l|l} & \mathsf{crs} \leftarrow \textsc{Setup}(1^\lambda) \\ & C_w \leftarrow \textsc{Commit}(\mathsf{crs}, w) \\ & \mathsf{pp} \leftarrow \textsc{Gen}(\mathsf{crs}) \\ & (\mathsf{vk}_*, \mathsf{sk}_*, \mathsf{ct}_*, \pi_*) \\ \mathcal{A}_2(\mathsf{pp}, F, C_w, & \qquad \Pi' \cdot \textsc{Enc}'(\mathsf{pp}, F, C_w, w) \\ \mathsf{hint}, \mathsf{aux}_b) = b & \mathsf{hint} \leftarrow \mathcal{A}_1(\mathsf{vk}_*, \mathsf{sk}_*, \mathsf{ct}_*, \pi_*) \\ & (\mathsf{aux}_0 = (\mathsf{vk}_0, \mathsf{ct}_0, \pi_0), \mathsf{msk}_0 = \mathsf{sk}_0) \\ & \qquad \Pi' \cdot \textsc{Enc}'(\mathsf{pp}, F, C_w, w) \\ & \mathsf{aux}_1 = (\mathsf{vk}_1, \mathsf{ct}_1, \pi_1) \\ & \qquad \Pi' \cdot \mathsf{Sim}(\mathsf{pp}, F, C_w, \mathsf{hint}) \\ & b \leftarrow_R \{0, 1\} \end{array}\right]$$

Note that no PPT adversary would be able to distinguish the encryptions of any two values $w_0 < w_1$ with the same commitment C_w by the computational zero-knowledge property of the original VECK protocol Π'. Although many such values exist, given an honestly generated crs, no PPT algorithm can break binding and actually find them. Then, even if the adversary knows the value w_0, it would not be able to distinguish the encryption of $F(w_0)$ from the encryption of $F(w_1)$ until the decryption keys for the ciphertexts of $F(w_0)$ and $F(w_1)$ are revealed. Since this is true for the adversary that might know w_0, it is also true for the adversary that holds partial information about $F(w_0)$ or w_0 itself, *e.g.*, a hint based on w_0. Now, let $\Pi \cdot \mathsf{Sim}$ be simply the PPT simulator of the original VECK protocol Π' (*cf.* computational zero-knowledge definition, Definition 3.1). By the argument above, given any hint generated by the PPT adversary ($\mathcal{A}_1$) based on w, the output $(\mathsf{vk}_1, \mathsf{ct}_1, \pi_1)$ generated by the simulator cannot be distinguished from a correctly generated output by any PPT distinguisher ($\mathcal{A}_2$) except with negligible probability. This implies L-bits zero-knowledge for the VECK protocol Π' as well as the MC-VECK protocols above.