# Fair Data Exchange with Constant-Time Proofs

Majid Khabbazian

University of Alberta, Canada

**Abstract.** The Fair Data Exchange (FDE) protocol introduced at CCS 2024 offers atomic pay-per-file transfers with constant-size proofs, but its prover and verifier runtimes still scale linearly with the file length $n$. We collapse these costs to $O(\lambda)$—essentially constant—by viewing the file as a rate-1 Reed–Solomon (RS) codeword, extending it to a lower-rate RS code with constant redundancy, encrypting this extended vector, and then proving correctness for only a pseudorandom $\Theta(\lambda)$ subset of the resulting ciphertexts; RS decoding repairs any corrupted symbols with negligible failure probability. Our protocol preserves full client- and server-fairness, and adds only a tunable communication redundancy overhead.

Finally, we patch the elliptic-curve mismatch in the Bitcoin instantiation of FDE with a compact zk-SNARK, enabling the entire exchange to run off-chain and falling back to just two on-chain transactions when channels are unavailable.

## 1 Introduction

Digital commerce increasingly revolves around the one-shot sale of large digital assets—scientific data sets, proprietary machine-learning weights, high-resolution media, even genomic archives—between parties with no prior trust relationship. Ensuring that the buyer actually receives the promised file while the seller simultaneously receives payment is therefore a foundation stone of the data-driven economy. Traditional escrow or licensing services resolve this tension by inserting costly legal or institutional intermediaries; blockchain-based atomic fair-exchange protocols aim to provide the same fairness guarantee with only a smart contract as referee. Achieving that goal without inflating on-chain fees or forcing the parties through many interactive rounds, however, remains challenging once file sizes grow into the multi-megabyte range.

The prominent blockchain-based fair-exchange protocols—FairSwap [1], FileBounty [3], and FairDownload [2]—all follow a dispute-driven model, resolving misbehavior through an explicit on-chain arbitration phase. FairSwap requires both parties to remain responsive throughout a complaint period, during which buyers must provide Merkle proofs of misbehavior. FileBounty relaxes continuous responsiveness by allowing zk-SNARK-based disputes at any point, limiting potential loss to individual file chunks but implicitly assuming partial data has proportional utility. FairDownload resolves disputes via off-chain exchange of signed data chunks, with disputes settled on-chain using $O(\log k)$-sized Merkle inclusion proofs.

The FDE protocol [5] entirely eliminates dispute timers and achieves constant-sized on-chain communication. In FDE, the seller publishes a single KZG polynomial commitment and provides an upfront, constant-sized *Verifiable Encryption of Committed Knowledge* (VECK) proof that every ciphertext sent to the buyer encrypts the correct polynomial evaluation. The protocol requires only three on-chain messages—commitment $\rightarrow$ payment $\rightarrow$ decryption key. The buyer can verify the VECK proof immediately, pay, and decrypt the file without waiting or worrying about subsequent disputes.

A significant drawback of the VECK protocols introduced in [5] for FDE is their high computational cost for both proof generation and verification. For a relatively small file (128 KiB, represented by $n = 4096$ encrypted evaluations), the original authors report approximately 34 seconds of verifier runtime and more than 40 seconds of prover runtime on commodity hardware.[1] Since these runtimes scale roughly linearly with file size, exchanging a 128 MiB file would extrapolate to *over nine hours* of verification and *approximately seventeen hours* of proof generation—clearly impractical for most real-world data-exchange scenarios.

An approach to overcoming the computational bottleneck described above is to incorporate coding. Consider the scenario where the original data is first encoded using a fixed-rate code with high minimum distance and an efficient decoding algorithm, and the commitment is subsequently applied to this encoded data. In this setup, the verifier only requires proof that a randomly selected subset of ciphertexts—of size proportional to the security parameter—decrypts correctly. This design effectively guarantees data recovery for the buyer, as any corrupted segments can be corrected through the decoding process.

However, embedding redundancy directly into the original committed data increases its size, an undesirable property for blockchain applications where storage efficiency is critical. For instance, Ethereum blobs are limited to 128 KiB; thus, coding inherently reduces the useful payload size. Moreover, and perhaps more critically, including coding at the commitment stage obliges proposers and validators on the blockchain to verify correctness of encoding, imposing significant additional computational burdens.

An alternative and more practical strategy is for the seller to separately provide additional encrypted redundancy alongside the original encrypted data. The seller must then demonstrate two critical facts to the buyer: (1) a randomly selected subset of ciphertexts correctly decrypts to the committed data values, and (2) the provided redundancy indeed results from applying an appropriate error-correcting code directly to the original data.

Our proposed solution follows this second approach. A key step is identifying an appropriate error-correcting code that not only has high minimum distance and efficient decoding but also permits concise proofs verifying that the coding was correctly applied to the original data committed by the seller. Upon investigating various coding strategies, we discovered a naturally fitting code already inherent within the KZG polynomial commitment framework used by FDE: Reed–Solomon codes. Reed–Solomon codes are maximum-distance sepa-

[1] Figure 6 in [5]; measurements for the El Gamal-based VECK protocol.

rable (MDS), enable efficient decoding, and the original KZG commitment inherently includes commitments to the redundancy portion. As a result, we can directly utilize the existing FDE protocol—with moderate modification—to realize our coding-based enhancement.

We demonstrate that our coding-based VECK protocol significantly reduces both proof generation and verification costs, achieving essentially constant complexity with respect to the file size, while fully preserving the atomicity and fairness guarantees of the original FDE protocol. This substantial improvement comes with two minor trade-offs: (1) the seller encrypts and transmits a modest and adjustable amount of redundancy (for example, a 10% overhead), and (2) the buyer executes an efficient decoding routine only in the unlikely event of seller misbehavior; even in such scenarios, fairness and atomicity remain strictly enforced. Notably, although our coding approach introduces redundancy alongside the original data, it does not require a larger Common Reference String (CRS) than the one used by the original FDE protocol.

In addition to the contributions described above, we identify a compatibility issue with the Bitcoin-based implementation of FDE proposed by the original authors. Specifically, the elliptic curve groups currently used by Bitcoin adaptor signatures (secp256k1) do not match the pairing-friendly elliptic curve groups (e.g., BLS12-381) required by the VECK protocols proposed in [5]. Moreover, Bitcoin currently has no proposals to adopt pairing-friendly curves. To resolve this incompatibility, we introduce a generic constant-time solution that leverages standard zk-SNARKs. Our approach not only addresses the elliptic-curve mismatch but also enables the entire protocol to be executed exclusively off-chain (within Bitcoin's Lightning Network), hence eliminating costly blockchain transactions that would otherwise increase the expense of file exchanges.

## 2 Background

### 2.1 Notation and Definitions

Let $\lambda \in \mathbb{N}$ denote the security parameter. A non-negative function $\sigma(\lambda)$ is said to be *negligible* if, for every polynomial $p(\lambda)$, there exists a sufficiently large $\lambda_0$ such that for all $\lambda \geq \lambda_0$:

$$\sigma(\lambda) \leq \frac{1}{p(\lambda)}.$$

For a random variable $x$, we write $x \leftarrow_R X$ to indicate that $x$ is drawn uniformly at random from the set $X$.

Throughout this paper, we denote by $\mathbb{F}_p$ the finite field of prime order $p$. We let $\mathbb{F}_p[X]$ represent the set of all univariate polynomials with coefficients in $\mathbb{F}_p$. Specifically, we denote by $\mathbb{F}_p^k[X]$ the set of polynomials of degree exactly $k$, and by $\mathbb{F}_p^{\leq k}[X]$ the set of polynomials of degree at most $k$.

We work with elliptic-curve groups $G_1$, $G_2$, and a target group $G_T$, each of prime order $p$. We utilize a cryptographic bilinear pairing map $e : G_1 \times G_2 \to G_T$, satisfying standard properties of bilinearity, non-degeneracy, and efficient computability. Our constructions specifically utilize the pairing-friendly elliptic curve BLS12-381, which provides approximately 128 bits of security. The prime order $p$ in the BLS12-381 curve is roughly 255 bits in length.

For a set $S$ and a function $\phi(X) \in \mathbb{F}_p[X]$, $\phi_S(X)$ denotes the minimal-degree polynomial in $\mathbb{F}_p[X]$ satisfying $\phi_S(i) = \phi(i)$ for all $i \in S$. Additionally, define

$$V_S(X) \coloneqq \prod_{i \in S}(X - i).$$

## 2.2 KZG Polynomial Commitments

The KZG polynomial commitment scheme [4] provides a cryptographic method for succinctly committing to polynomials and efficiently generating evaluation proofs. It leverages pairing-based cryptography to achieve constant-size commitments and evaluation proofs, independently of the polynomial degree.

Formally, the KZG polynomial commitment scheme consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda, n) \rightarrow \mathsf{crs}$: generates $\mathcal{G}$: elliptic-curve groups $G_1, G_2, G_T$ of prime order $p \geq 2^{2\lambda}$, with generators $g_1 \in G_1$, $g_2 \in G_2$, $g_T \in G_T$, and a bilinear pairing map $e : G_1 \times G_2 \rightarrow G_T$. Samples a uniformly random secret $\tau \leftarrow_R \mathbb{F}_p$ and publish
$$\mathsf{crs} = \left(\mathcal{G}, \{g_1^{\tau^i}\}_{i=1}^n, \{g_2^{\tau^i}\}_{i=1}^n\right).$$
- $\mathsf{Commit}(\mathsf{crs}, \phi) \rightarrow C$: Given the $\mathsf{crs}$ and the coefficients of $\phi(X) \in \mathbb{F}_p[X]$, output the commitment
$$C = g_1^{\phi(\tau)}.$$
- $\mathsf{Open}(\mathsf{crs}, \phi, i) \rightarrow \pi$: Given public parameters $\mathsf{crs}$, polynomial $\phi(X) \in \mathbb{F}_p[X]$, and an evaluation point $i$, the prover outputs the opening proof:
$$\pi = g_1^{\frac{\phi(\tau) - \phi(i)}{\tau - i}} \in G_1.$$
- $\mathsf{Verify}(\mathsf{crs}, C_\phi, i, \phi(i), \pi) \rightarrow 0/1$: To verify an opening, the verifier checks the pairing equation:
$$e(C_\phi / g_1^{\phi(i)}, g_2) \stackrel{?}{=} e(\pi_i, g_2^{\tau - i}),$$
and outputs 1 if it holds, otherwise 0.
- $\mathsf{batchOpen}(\mathsf{crs}, \phi, S) \rightarrow \pi$: Given public parameters $\mathsf{crs}$, polynomial $\phi(X)$, and multiple distinct evaluation points $S$, the prover computes the evaluations $\{\phi(i)\}_{i \in S}$ and produces a single aggregated proof:
$$\pi = g_1^{q(\tau)},$$
where
$$q(X) = \frac{\phi(X) - \phi_S(X)}{V_S(X)}.$$

– $\mathsf{batchVerify}(\mathsf{crs}, C_\phi, S, \{\phi(i)\}_{i\in S}, \pi) \to 0/1$: The verifier efficiently checks the aggregated proof $\pi$ by verifying the following pairing equation:

$$e\left(C_\phi / g_1^{\phi_S(\tau)},\, g_2\right) \overset{?}{=} e\left(\pi,\, g_2^{V_S(\tau)}\right).$$

The verifier outputs 1 if this equation holds and 0 otherwise.

## 2.3 FDE Protocol

FDE is a blockchain-based protocol which enables a client and a storage server to atomically exchange data for payment. Atomicity guarantees fairness: the server receives payment if and only if the client obtains the promised data. The FDE protocol leverages the KZG polynomial commitment scheme, chosen due to its constant-size commitments and efficient batchable opening proofs, making it particularly suitable for scenarios where clients may retrieve subsets of data. Additionally, KZG commitments are already widely adopted in blockchain ecosystems, notably in Ethereum's Danksharding for data availability, making FDE naturally compatible with existing infrastructure.

In the FDE protocol, data is represented as evaluations of a polynomial $\phi(\cdot)$ of degree $\ell \le n$, that is as $\{\phi(i)\}_{i=0}^{\ell}$. The server first publishes a public verification key $\mathsf{vk}$ to a blockchain smart contract, alongside specific transaction details such as the agreed price and the client's blockchain address (step 1). Off-chain, the server then sends the encrypted evaluations $\{\mathsf{ct}_i\}_{i=0}^{\ell}$ of each data point $\{\phi(i)\}_{i=0}^{\ell}$ to the client. These ciphertexts are accompanied by a cryptographic proof showing that each ciphertext $\mathsf{ct}_i$ correctly encrypts the corresponding polynomial evaluation $\phi(i)$ committed to by a KZG polynomial commitment $C_\phi$, under a secret decryption key $\mathsf{sk}$ that matches the previously submitted verification key $\mathsf{vk}$ (step 2).

Upon receiving and verifying these ciphertexts and associated proofs, the client locks the agreed-upon funds in the blockchain smart contract (step 3). The server subsequently can claim these funds only by revealing the correct decryption key $\mathsf{sk}$ that corresponds to the public verification key $\mathsf{vk}$ (step 4). After the server publishes the decryption key, the client retrieves it from the blockchain (step 5), allowing immediate decryption of the ciphertexts and recovery of the original committed data (step 6). If the server fails to reveal the secret key within a specified timeout, the client recovers the funds locked in the contract, thus preserving fairness.

FDE satisfies three critical properties: correctness (honest parties always succeed), client-fairness (the server obtains payment only if the client receives the data), and server-fairness (the client learns nothing about the data unless the server is paid).

## 2.4 Verifiable Encryption under Committed Key (VECK)

At the heart of the FDE protocol lies a novel cryptographic primitive called VECK. At a high level, VECK enables a prover (in our context, the server) to demonstrate that a set of ciphertexts indeed encrypts evaluations of a polynomial at specific points, using an encryption key consistent with a publicly committed verification key. Concretely, VECK allows efficient verification of ciphertext correctness against a polynomial commitment without revealing the underlying plaintext evaluations or the encryption key itself. This construction ensures that the verifier (client) learns no additional information beyond the correctness of ciphertexts until the prover reveals the corresponding decryption key.

**Formal description.** Let $(\mathsf{Setup}, \mathsf{Commit})$ be a non-interactive binding commitment scheme, where $\mathsf{Setup}(1^\lambda, n) \to \mathsf{crs}$ generates a public common reference string, and $\mathsf{Commit}(\mathsf{crs}, w \in W) \to C_w$ generates a commitment to $w$. A non-interactive VECK scheme for a class of functions $\mathcal{F} = \{F : W \to V\}$ is defined as follows [5]:

- $\mathsf{VECK.Gen}(\mathsf{crs}) \to \mathsf{pp}$: A PPT algorithm that, given the $\mathsf{crs}$, outputs parameters $\mathsf{pp}$ and defines relevant spaces. The parameters $\mathsf{pp}$ are implicitly provided to subsequent algorithms.
- $\mathsf{VECK.Enc}(\mathsf{F}, \mathsf{C_w}, \mathsf{w}) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$: A PPT algorithm run by the server that takes $(\mathsf{F}, \mathsf{C_w}, \mathsf{w})$ and outputs a verification key $\mathsf{vk}$, a decryption key $\mathsf{sk}$ and the encryptions $\mathsf{ct}$ of $F(w)$, as well as a proof $\pi$.
- $\mathsf{VECK.Ver_{ct}}(\mathsf{F}, \mathsf{C_w}, \mathsf{vk}, \mathsf{ct}, \pi) \to \{0,1\}$: A deterministic polynomial-time algorithm executed by the client that returns either *accept* or *reject*.
- $\mathsf{VECK.Ver_{key}}(\mathsf{vk}, \mathsf{sk}) \to \{0,1\}$: A deterministic polynomial-time algorithm executed by the blockchain or a trusted third party to verify the validity of the secret key.
- $\mathsf{VECK.Dec}(\mathsf{sk}, \mathsf{ct}) \to v/\perp$: A deterministic polynomial-time algorithm executed by the client that outputs a value in $V$ or a failure symbol $\perp$.

A secure VECK scheme must satisfy the following key properties:

- *Correctness (informal)*: Honestly generated ciphertexts and proofs always verify correctly, and decryption recovers the original polynomial evaluations.
- *Soundness (informal)*: No computationally bounded adversary can generate ciphertexts and corresponding proofs that pass verification yet encrypt values different from the committed polynomial evaluations, unless cryptographic assumptions are broken.
- *Zero-Knowledge (informal)*: The proof and ciphertexts do not reveal any additional information about the underlying polynomial evaluations or secret encryption key beyond correctness until the decryption key is explicitly revealed.

### 2.5 Reed–Solomon Codes

Let $(a_0, \dots, a_\ell) \in \mathbb{F}_p^{\ell+1}$ be a message vector and define the degree-$\ell$ polynomial

$$\phi(X) = \sum_{i=0}^{\ell} a_i X^i.$$

Fix $n > \ell$ pairwise-distinct evaluation points $\alpha_1, \dots, \alpha_n \in \mathbb{F}_p$. The corresponding Reed–Solomon codeword is

$$\mathbf{c} = \big(\phi(\alpha_1), \dots, \phi(\alpha_n)\big) \in \mathbb{F}_p^n.$$

This construction yields an $(n, \ell + 1)$ linear code with rate $R = (\ell + 1)/n$ and minimum Hamming distance $d_{\min} = n - \ell$, hence being able to correct up to $\lfloor (d_{\min} - 1)/2 \rfloor$ symbol errors. Decoding is efficient: algorithms such as Berlekamp–Massey, the extended Euclidean method, or FFT-based techniques recover the original message in $O(n^2)$ down to $O(n \log^2 n)$ time.

## 3 FDE with Constant Proof-time

**Overview.** To achieve constant proof-time in FDE, we propose a new VECK protocol, denoted as $\mathsf{VECK}^+_{El}$, and integrate it into the existing FDE protocol. Before formally presenting $\mathsf{VECK}^+_{El}$, we summarize the key insights underlying our approach.

Recall that in the original FDE protocol, the data consists of evaluations of a polynomial $\phi(X)$ over a domain $[\ell] \coloneqq \{0, 1, \dots, \ell\}$, with a succinct polynomial commitment $C_\phi$. This structure naturally supports polynomial commitment schemes such as KZG, which conveniently enable efficient batch openings and compact verification proofs.

While data continues to be explicitly represented by polynomial evaluations at the points in $[\ell]$, we observe that knowledge of the polynomial coefficients $(a_0, a_1, \dots, a_\ell)$ is sufficient to completely determine the data, as these coefficients fully specify the polynomial $\phi(X) = a_0 + a_1 x + \cdots + a_\ell x^\ell$. Thus, one may view the coefficient vector as a *complete representative* of the data. Evaluating this polynomial at the points in $[\ell]$ corresponds precisely to encoding the coefficient vector into a Reed–Solomon (RS) codeword.

Crucially, by extending the evaluation domain beyond $[\ell]$, we obtain a longer RS codeword with a lower rate, inherently capable of correcting errors. This insight motivates the first part of our enhanced protocol: the server encrypts an expanded set of polynomial evaluations, extending beyond the original evaluation points. A subtle yet significant observation is that this expanded evaluation set does not alter the underlying polynomial commitment. In other words, the same original commitment $C_\phi$ is valid for the extended set of evaluations. Thus, the prover can seamlessly use the original commitment, eliminating any need to create or prove additional commitments.

The final piece of our approach involves verifying encryption correctness on only a small random subset of size $\Theta(\lambda)$. Assuming that the verifaciton passes, due to the error-correcting capability of the underlying RS code, the probability of decoding failure will be negligible in $\lambda$. Consequently, the soundness of the original protocol—requiring the adversary's success probability to be negligible—is preserved.

In the more general setting, where a client may request evaluations of the polynomial at a subset of points $S \subseteq [\ell]$, the server first constructs a polynomial

$\phi'_S(X)$ that matches the original polynomial $\phi(X)$ over the points in $S$. The server then sends encrypted evaluations of $\phi'_S(X)$ at an extended superset of points $S^+ \supseteq S$. Importantly, the evaluations on the extended set $S^+$ reveal no additional information beyond what is already contained in the evaluations of $\phi'(X)$ over the original set $S$. The server subsequently proves correctness of encryption only on a small random subset $S_R \subseteq S^+$, of size $\Theta(\lambda)$.

We formalize the above idea by proposing $\mathsf{VECK}^+_{El}$, an enhanced ElGamal-based VECK protocol.

**Formal description.** As in [5], we use KZG as our commitment scheme and let the function $F$ in VECK be the evaluation of polynomial at a given set of points. Our $\mathsf{VECK}^+_{El}$ protocol is ElGmal based. This choice is merely because the ElGamal-based VECK protocol proposed in [5] requires much lower bandwidth than the Paillier-based VECK at the cost of higher proof-time complexity. This choice is therefore reasonable as we reduce the proof time to constant (but do not improve the communication overhead).

Let $\mathsf{VECK}_{\mathsf{El}}$ denote the Elgamal-based VECK protocol proposed in [5]. Since $\mathsf{VECK}^+_{\mathsf{El}}$ and $\mathsf{VECK}_{\mathsf{El}}$ have similarities, instead of constructing $\mathsf{VECK}^+_{\mathsf{El}}$ from scratch, we strive to call $\mathsf{VECK}_{\mathsf{El}}$ operation whenever possible to reduce repetition of basic operations already covered in the construction of $\mathsf{VECK}_{\mathsf{El}}$. Towards this end, we split the operations $\mathsf{VECK}_{\mathsf{El}}.\mathsf{Enc} \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$ into two sub-operations:

$$\mathsf{VECK}_{\mathsf{El}}.\mathsf{Enc}_1 \rightarrow (\mathsf{vk}, \mathsf{sk}, \mathsf{ct})$$

and

$$\mathsf{VECK}_{\mathsf{El}}.\mathsf{Enc}_2 \rightarrow (\pi, \mathsf{ct}_-).$$

The first sub-operation returns all outputs except the proof, while the second sub-operation returns only the proof along with $\mathsf{ct}_-$, an encryption at point $-1$ employed in $\mathsf{VECK}_{\mathsf{El}}.\mathsf{Enc}$ to assist with ciphertext verification [5].

Furthermore, we stipulate that if $\mathsf{VECK}_{\mathsf{El}}.\mathsf{Dec}$ fails to recover a value—that is, the plaintext falls outside the brute-force search range—it returns $\perp$, which we treat as an erasure at that position.

Let $\beta > 1$ be a fixed constant that balances bandwidth overhead against computational cost in the protocol. Given a (possibly corrupted) set of polynomial evaluations $D = \{\phi(i)\}_{i \in I}$ of a polynomial $\phi(X)$ at the index set $I$, we denote by

$$\mathsf{RS.Dec}(S, I, D)$$

the Reed–Solomon decoder at positions $S$. If decoding succeeds, it returns the correct values $\{\phi(i)\}_{i \in S}$.

**Case $S = [\ell]$.** This corresponds to the scenario in which the client requests the entire data set. We handle the case $S \subset [\ell]$ separately, as requesting the full data set ($S = [\ell]$) allows fewer steps in both $\mathsf{VECKplus_{El}.Enc}$ and $\mathsf{VECK^{+}_{El}.Ver_{ct}}$.

$\mathsf{VECK^{+}_{El}}$*:*

- $\mathsf{VECK^{+}_{EL}.GEN(crs)} \to \mathsf{pp}$: On input $\mathsf{crs} = \left(\mathcal{G}, \{g_1^{\tau_i}\}_{i=1}^{n}, \{g_2^{\tau_i}\}_{i=1}^{n}\right)$, generate random group elements with unknown discrete logarithms $h \leftarrow_R G_1$ and $h_i \leftarrow_R G_1$ for $i \in [m] \cup \{-1\}$, where $m = \lceil \beta \cdot \ell \rceil$.

- $\mathsf{VECK^{+}_{EL}.Enc}(\mathsf{F}_{[\ell]}, \mathsf{C}_\phi, \phi(\mathsf{X})) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$.
  1. Compute $(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}) \coloneqq \mathsf{VECK_{El}.Enc_1}(\mathsf{F_{[m]}}, \mathsf{C}_\phi, \phi(\mathsf{X}))$
  2. Generate a Fiat-Shamir pseudo-random challenge subset $S_R \subset [m]$ with $|S_R| = \min(\ell + 1, \lceil \frac{\lambda}{\beta - 1} \rceil)$.
  3. Compute $(\pi_R, \mathsf{ct}_-) \coloneqq \mathsf{VECK_{El}.Enc_2}(\mathsf{F_{S_R}}, \mathsf{C}_\phi, \phi(\mathsf{X}))$.
  4. Output $(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi = (\pi_\mathsf{R}, \mathsf{ct}_-))$

- $\mathsf{VECK^{+}_{EL}.Ver_{ct}}(\mathsf{F}_{[\ell]}, \mathsf{C}_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \to 0/1$
  1. Parse $\pi$ as $(\pi_R, \mathsf{ct}_-)$.
  2. Output $\mathsf{VECK_{El}.Ver_{ct}}(\mathsf{F_{S_R}}, \mathsf{C}_\alpha, \mathsf{vk}, \{\mathsf{ct_i}\}_{\mathsf{i \in S_R}} \cup \mathsf{ct}_-, \pi_\mathsf{R})$.

- $\mathsf{VECK^{+}_{El}.Ver_{key}}(\mathsf{vk}, \mathsf{sk}) \to 0/1$ : For $\mathsf{sk} = s \in \mathbb{F}_p$, return 1 iff $\mathsf{vk} = h^s$.

- $\mathsf{VECK^{+}_{El}.Dec}(\mathsf{F}_{[\ell]}, \mathsf{sk}, \mathsf{ct}) \to \{\phi(i)\}_{i \in [\ell]}$.
  1. Output $\mathsf{RS.Dec}([\ell], [\mathsf{m}], \mathsf{VECK.Dec}(\mathsf{F_{[m]}}, \mathsf{sk}, \mathsf{ct}))$.

**Theorem 1.** *For the case $S = [\ell]$, the generation and verification times of $\mathsf{VECK^{+}_{El}}$ are $\mathcal{O}(\lambda)$.*

*Proof.* Each step (specifically, steps 2 and 3) in the proof generation runs in $\mathcal{O}(\lambda)$ time: generating the random subset $S_R$ requires $\mathcal{O}(R) = \mathcal{O}(\lambda)$ operations, and $\mathsf{VECK_{El}.Enc_2}$ (step 3) has complexity linear in $|S_R|$, which is also $\mathcal{O}(\lambda)$. Similarly, the time complexity of $\mathsf{VECK_{El}.Ver_{ct}}$ is linear in $|S_R| \in \mathcal{O}(\lambda)$.

**Case $S \subset [\ell]$.** This corresponds to the scenario in which the client does not request the entire data set.

$\mathsf{VECK}^{+}_{\mathsf{El}}$:

- $\mathsf{VECK}^{+}_{\mathsf{El}}.\mathsf{GEN}(\mathsf{crs}) \to \mathsf{pp}$: On input $\mathsf{crs} = \big(\mathcal{G}, \{g_1^{\tau_i}\}_{i=1}^{n}, \{g_2^{\tau_i}\}_{i=1}^{n}\big)$, generate random group elements with unknown discrete logarithms $h \leftarrow_R G_1$ and $h_i \leftarrow_R G_1$ for $i \in [m] \cup \{-1\}$, where $m = \lceil \beta \cdot |S| \rceil$.

- $\mathsf{VECK}^{+}_{\mathsf{EL}}.\mathsf{Enc}(\mathsf{F_S}, \mathsf{C}_\phi, \phi(\mathsf{X})) \to (\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi)$.
  1. Sample $t \leftarrow_R \mathbb{F}_p$.
  2. Set $\phi'_S(X) := t\, V_S(X) + \phi_S(X)$.
  3. Compute $C_S := \mathsf{Commit}(\mathsf{crs},\ \phi'_S(X))$.
  4. Compute $(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}) := \mathsf{VECK_{EL}}.\mathsf{Enc_1}(\mathsf{F_{[m]}}, \mathsf{C_S}, \phi'_{\mathsf{S}}(\mathsf{X}))$
  5. Compute $\pi_S := \mathsf{batchOpen}\big(\mathsf{crs},\ \phi(X) - \phi'_S(X), S\big)$.
  6. Generate a Fiat-Shamir pseudo-random challenge subset $|S_R| = \min(|S| + 1, \lceil \frac{\lambda}{\beta - 1} \rceil)$.
  7. Compute $(\pi_R, \mathsf{ct}_-) := \mathsf{VECK_{El}}.\mathsf{Enc_2}(\mathsf{F_{S_R}}, \mathsf{C_S}, \phi'_{\mathsf{S}}(\mathsf{X}))$.
  8. Output $\big(\mathsf{vk}, \mathsf{sk}, \mathsf{ct}, \pi = (\pi_{\mathsf{S}}, \pi_{\mathsf{R}}, \mathsf{ct}_-)\big)$

- $\mathsf{VECK}^{+}_{\mathsf{EL}}.\mathsf{Ver_{ct}}(\mathsf{F_S}, \mathsf{C}_\phi, \mathsf{vk}, \mathsf{ct}, \pi) \to 0/1$
  1. Parse $\pi$ as $(\pi_S, \pi_R, \mathsf{ct}_-)$.
  2. Compute $b_1 := \mathsf{batchVefiry}(\mathsf{crs}, \mathsf{C}_\phi / \mathsf{C_S}, \mathsf{S}, \mathbf{0}, \pi_{\mathsf{S}})$.
  3. Compute $b_2 = \mathsf{VECK_{El}}.\mathsf{Ver_{ct}}(\mathsf{F_{S_R}}, \mathsf{C_S}, \mathsf{vk}, \{\mathsf{ct_i}\}_{\mathsf{i} \in \mathsf{S}} \cup \mathsf{ct}_-, \pi_{\mathsf{R}})$.
  4. Output $b_1 \wedge b_2$.

- $\mathsf{VECK}^{+}_{\mathsf{El}}.\mathsf{Ver_{key}}(\mathsf{vk}, \mathsf{sk}) \to 0/1$ : For $\mathsf{sk} = s \in \mathbb{F}_p$, return 1 iff $\mathsf{vk} = h^s$.

- $\mathsf{VECK}^{+}_{\mathsf{El}}.\mathsf{Dec}(\mathsf{F_S}, \mathsf{sk}, \mathsf{ct}) \to \{\phi(i)\}_{i \in S}$.
  1. Output $\mathsf{RS.Dec}(\mathsf{S}, [\mathsf{m}], \mathsf{VECK.Dec}(\mathsf{F_{[m]}}, \mathsf{sk}, \mathsf{ct}))$.

**Theorem 2.** *Let $S \subset [\ell]$.*

*(i) If the verifier has the pre-computed element ${g_2}^{V_S(\tau)}$, the running time of* $\mathsf{VECK}^{+}_{\mathsf{El}}.\mathsf{Ver_{ct}}$ *is $\mathcal{O}(\lambda)$.*
*(ii) Otherwise, the prover can supply a KZG opening proof for $V_S(X)$ at a Fiat–Shamir-chosen point $\kappa$, in which case verification costs $\mathcal{O}(|S| + \lambda)$.*

*Proof.* When ${g_2}^{V_S(\tau)}$ is available, Step 2 requires only a constant number of group operations, i.e. $\mathcal{O}(1)$. Step 3 performs $\lambda$ pairing/exponentiation checks, so the total cost is $\mathcal{O}(\lambda)$.

If the verifier lacks ${g_2}^{V_S(\tau)}$, the prover can provide a KZG opening $\pi$ of $V_S(X)$ at $\kappa$. Verifying $\pi$ costs $\mathcal{O}(1)$ pairings, but the verifier must evaluate $V_S(\kappa)$ locally, which takes $\mathcal{O}(|S|)$ field operations. Together with the $\mathcal{O}(\lambda)$ checks, this yields $\mathcal{O}(|S| + \lambda)$ overall.

**Theorem 3.** *In the random-oracle and algebraic-group models, the protocol* $\mathsf{VECK}^{+}_{\mathsf{El}}$ *satisfies correctness, soundness, and computational zero-knowledge; hence it is a secure* $\mathsf{VECK}$.

## 4 FDA via Payment Channels

Tas *et al.* [5] proposed a Bitcoin-based implementation of FDA that leverages adaptor signatures. However, their scheme utilizes pairing-friendly elliptic curves (e.g., BLS12-381), which are incompatible with the secp256k1 curve currently used by Bitcoin's adaptor signatures. Consequently, direct integration of their proposed FDE protocols into Bitcoin faces significant practical barriers.

To resolve this incompatibility, we propose leveraging Bitcoin's Lightning Network (LN) combined with zk-SNARKs. As an additional advantage, our approach naturally eliminates the need for costly on-chain transactions.

Specifically, the server (prover) generates a zk-SNARK proof $\pi_t$ attesting knowledge of a secret scalar $sk$ that satisfies the following relation:

$$R\big((h, vk, t), sk\big) = [vk = h^{sk} \wedge t = H(sk)],$$

where $h$, $vk$, and $t$ are publicly known parameters. Although generating the zk-SNARK proof $\pi_t$ introduces computational overhead, this cost is constant with respect to the data size and can be precomputed.

Our resulting FDA protocol operates as follows:

**Step 1:** The server sends to the client the ciphertext and its proof by calling $\mathsf{VECK}^{+}_{\mathsf{El}}.\mathsf{Enc}$, the value $t$, and the zk-SNARK proof $\pi_t$.

**Step 2:** The client verifies all provided proofs, including $\pi_t$. Upon successful verification, the client initiates a payment via LN, using $t$ as the hash value in the HTLC.

Due to the atomicity provided by LN, the server obtains the payment *iff* it reveals the secret scalar $\mathsf{sk}$, enabling the client to decrypt the ciphertext. Crucially, this achieves atomic exchange without incurring additional costly blockchain transactions.

*Fallback without LN.* If either party lacks LN connectivity, the same fairness guarantees can be enforced using exactly two Bitcoin transactions[2], structured as follows:

1. **Funding Transaction:** The client locks the payment amount in a P2WSH output with the script:

```
   OP_IF   OP_SHA256 ⟨t⟩ OP_EQUALVERIFY
           ⟨server_pk⟩ OP_CHECKSIG
 OP_ELSE   ⟨T_timeout⟩ OP_CHECKLOCKTIMEVERIFY OP_DROP
           ⟨client_pk⟩ OP_CHECKSIG
  OP_ENDIF
```

[2] The original FDA protocol on Ethereum uses three on-chain transactions in the success path: registering the server's public key, locking payment by the client, and the server revealing the decryption key [5].

Here, $t = H(sk)$ is the hash commitment, and $T_{\text{timeout}}$ is a blockchain timestamp after which the client can reclaim the funds.

2. **Spend or Refund Transaction:** The above output is spent via one of two branches:
   - **Success Path (before $T_{\text{timeout}}$):** The server spends the output with witness data:
   $$\langle sk \rangle,\ \langle \sigma_{\text{server}} \rangle,\ \texttt{1}$$
   The secret scalar $sk$ is thus revealed on-chain, allowing the client to decrypt the data.
   - **Refund Path (after $T_{\text{timeout}}$):** The client reclaims the output by providing:
   $$\langle \sigma_{\text{client}} \rangle,\ \texttt{0}$$
   No secret is revealed, and the client safely recovers their funds.

This fallback ensures fairness with minimal overhead when LN nodes are unavailable.

## 5 Conclusion

We presented a practical implementation of FDE that reduces proof generation and verification complexities from linear in the file size to $O(\lambda)$, achieved by incorporating Reed–Solomon redundancy and sampling only $\Theta(\lambda)$ ciphertexts. The proposed protocol maintains FDE's fairness guarantees while incurring minimal additional bandwidth overhead.

To further reduce computational overhead at the verifier (client) side, an honest server may optionally transmit the plaintext directly to the client after receiving the payment in full.

## References

1. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS. pp. 967–984. ACM (2018). https://doi.org/10.1145/3243734.3243857
2. He, S., Lu, Y., Tang, Q., Wang, G., Wu, C.Q.: Fair peer-to-peer content delivery via blockchain. In: Bertino, E., Schulmann, H., Waidner, M. (eds.) Computer Security - ESORICS. Lecture Notes in Computer Science, vol. 12972, pp. 348–369. Springer (2021). https://doi.org/10.1007/978-3-030-88418-5_17, https://doi.org/10.1007/978-3-030-88418-5_17
3. Janin, S., Qin, K., Mamageishvili, A., Gervais, A.: Filebounty: Fair data exchange. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops. pp. 357–366. IEEE (2020). https://doi.org/10.1109/EUROSPW51379.2020.00056

4. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 177–194 (2010). https://doi.org/10.1007/978-3-642-17373-8_11, https://doi.org/10.1007/978-3-642-17373-8_11
5. Tas, E.N., Seres, I.A., Zhang, Y., Melczer, M., Kelkar, M., Bonneau, J., Nikolaenko, V.: Atomic and fair data exchange via blockchain. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS. pp. 3227–3241. ACM (2024). https://doi.org/10.1145/3658644.3690248