COM-506 Report : Atomic and Fair Data Exchange via Blockchain

Brandy Nogales - 275195
Engjëll Ismaili - 344534
Louis Giraud - 334749

March 14, 2025

1 Introduction

In this report, we begin by introducing the concept of Fair Data Exchange (FDE) and detailing its operational mechanisms. We then explore Smart Contracts and discuss how they can replace a trusted third party to facilitate a fair exchange. Next, we introduce the cryptographic primitive known as Verifiable Encryption under Committed Key (VECK), which, in our context, enables a client to store data on a server and later retrieve exactly the same data. The client commits to the data upfront, ensuring that the received data can be verified against the original commitment. We also present the mathematical foundations of the underlying commitment scheme, which is based on KZG Polynomial Commitments. Furthermore, we describe various VECK constructions, including those based on ElGamal and Paillier encryption schemes, and explain how the protocol operates with each variant. Performance evaluation of the FDE protocol under both the ElGamal and Paillier constructions is provided, followed by an analysis of how the protocol can be extended to a multi-client setting.

2 Fair Data Exchange

In general, the Fair Data Exchange (FDE) protocol involves two parties—typically a client and a server—each possessing something that the other requires. For the exchange to be considered fair, the following conditions must be met:

- A fair exchange occurs when both parties receive exactly what the other offers; if the process is interrupted, neither party should obtain any information about the other's data.
- An exchange is deemed unfair if one party obtains information about the other's data without reciprocating with their own.

In our example, a client intends to store potentially large data on a cloud provider's server. Because the server is responsible for data storage, the client must pay for the service when retrieving the data. One common approach is a subscription-based model, where the client pays in advance and relies on the server's reputation to deliver the data. However, a truly fair exchange would ensure that the client receives the data only if the server obtains the payment. To achieve this fairness without depending on a trusted third party, blockchain technology and smart contracts can be employed to securely mediate the exchange.

3 Example protocol overview

We will see an example of how we could use VECK in an Ethereum Based protocol as illustrated in Figure 1, which unfolds in six steps: **(Step 1)** The server sends a public key that serves as a commitment to its secret key. **(Step 2)** The server sends the encrypted data along with a proof that it correctly possesses the secret key needed to decrypt the data. **(Step 3)** The client locks funds in the smart contract. **(Step 4)** The server submits its secret key to the smart contract. **(Step 5)** The smart contract verifies

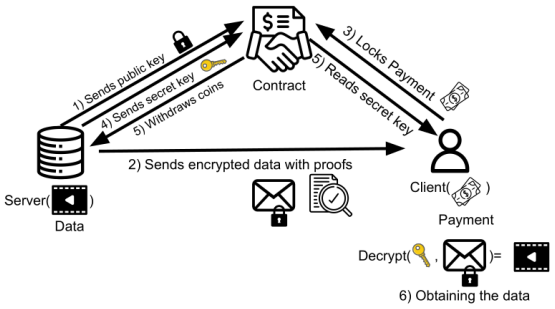

Figure 1: The blueprint of FDE protocols on Ethereum that relies on a smart contract for achieving fairness

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

Figure 2: Algorithms from the paper

that the server's public key is a valid commitment to the secret key. If the verification succeeds, it transfers the funds to the server and the secret key to the client. **(Step 6)** The client uses the secret key to decrypt the data.
With this overview of Fair Data Exchange, smart contracts, and the protocol in place, we now proceed to introduce the concept of Verifiable Encryption under Committed Key (VECK).

4 Verifiable encryption under committed key (VECK)

Before presenting the formal definition for VECK, let's recall what polynomial commitments are and their properties. As said in the name, these schemes allow us to commit to univariate polynomials of degree at most l over $\mathbb{F}_p$ and are comprised of the algorithms in Figure 2. A secure polynomial commitment scheme should also satisfy the following properties : **Correctness**, **Polynomial Binding** and **Evaluation Binding**.

Veck allows the encryptor to encrypt some data that will output : a verification key, the ciphertext and a zero-knowledge proof of correct encryption. It satisfies correctness, soundness and zero-knowledge. For each of them, here are the guarantees they provide :

- **Correctness** guarantees that decryption key corresponding to the verification key correctly decrypts the original data.
- **Soudness** guarantees that a polynomial time adversary can't generate a valid proof about an incorrect encryption (the server can't cheat)
- **Zero-Knowledge** guarantees that the verification key, the proof and the ciphertext don't leak information that could recover the data (the client can't cheat).

In the paper, they defined VECK as shown in Figure 3, which satisfies the above properties. Finally, note that VECK can also use symmetric key encryption, opening a prospect for using more efficient schemes. For the FDE application, we will introduce a commitment scheme where the VECK function F will be the evaluation of a given degree-l polynomial : $F(\phi) = \{\phi(i)\}_{i \in [l]}$, where $\phi(X)$ is a polynomial of degree l.

5 VECK Constructions

The paper presents two instantiations of the VECK primitive. The first one based on the Decisional Diffie-Hellman problem, while the second one uses the Decisional Composite Residuosity problem, based on the Paillier encryption scheme. Both use KZG Polynomial Commitments as a Commitment primitive, which we will explain in Section 5.1.

5.1 KZG Polynomial Commitments

It is one of the most widely used polynomial commitment schemes. The goal here is to commit to a polynomial $P(x)$ of degree $\leq l$ with coefficients

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

- $\text{Gen}(\text{crs}) \to \text{pp}$*: 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.*
- $\text{Enc}(\text{pp}, F, C_w, w) \to (\text{vk}, \text{sk}, \text{ct}, \pi)$*: Probabilistic polynomial-time algorithm, run by the server, It takes in the function* F*, the commitment* C_w *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*.*
- $\text{Ver}_{\text{ct}}(\text{pp}, F, C_w, \text{vk}, \text{ct}, \pi) \to 1/0$*: A deterministic polynomial-time algorithm run by the client that outputs accept or reject.*
- $\text{Ver}_{\text{key}}(\text{pp}, \text{vk}, \text{sk}) \to 1/0$[5]*: A deterministic polynomial-time algorithm run by the blockchain or a trusted third party that checks the validity of the secret key.*
- $\text{Dec}(\text{pp}, \text{sk}, \text{ct}) \to v/\perp$*: A deterministic polynomial-time algorithm run by the client, it outputs a value (such as an evaluation of* F *on* w*) or* $\perp$*.*

Figure 3: Definition of VECK from the original paper

in $\mathbb{F}_p$. The scheme is based on the pairing of elliptic curves, using a nontrivial bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A prover can generate a proof that $P(a) = b$ without revealing $P(x)$. The verifier can verify the proof without having the coefficients of $P(x)$ using a pairing-based equation. The prover can then reveal $P(x)$, which he had committed to beforehand.

5.2 ElGamal VECK and Paillier VECK

The first instantiation of the protocol bases its security on the hardness of the DDH problem and is proven secure in the Algebraic Group Model (AGM). The AGM is an abstraction of an adversary that can only use algebraic algorithms in its attack, i.e., generate an output only algebraically based on the inputs provided in the game definition of the problem. We will not go into the mathematical details of the instantiation in this summary, but all details can be found in the original paper.

The main issue with ElGamal VECK is the size of the ciphertext. With the described protocol, there is a non-constant bloat up when encrypting that can be reduced when using this second solution. This solution is based on the Paillier encryption scheme, and has a constant bloat-up factor. This second protocol also comes with its downsides, which will be explained in more details in the performance part. The mathematical details of these instantiations will not be mentioned here, but they can be found in the original paper.

6 Performance Evaluation

The protocols introduced in this paper, including FDE-ElGamal and FDE-Paillier, offer significant improvements over previous methods like FairSwap, FileBounty, and FairDownload by employing KZG polynomial commitments instead of traditional Merkle trees. This change reduces the complexity to three rounds and minimizes on-chain communication. Additionally, the server amortizes costs by consolidating funds from multiple exchanges, enabling single withdrawals to cover various payments. Performance tests were conducted on an AMD Ryzen 5 3600 (6-core) CPU with 8GB RAM.

6.0.1 Off-Chain Costs

Prover Time: In the Exponential ElGamal scheme, we set $k = 8$, which involves dividing each BLS12-381 scalar into k smaller plaintexts. This approach improves decryption efficiency but results in an increase in ciphertext size by a factor of k. Encrypting and generating consistency proofs for 4,096 BLS12-381 points ($\approx$ 128 KiB) requires 89 seconds. In contrast, the Paillier-based protocol scales linearly with the number of elements and takes only about 5.09 seconds.

Verifier Time: Verifying 4,096 ElGamal ciphertexts against a KZG commitment takes 34.15 seconds, primarily due to computationally expensive multi-scalar multiplications (MSMs). On the other hand, verification for the Paillier protocol takes 19.45 seconds, with decryption requiring only 9.54 seconds.

Proof Size: The Exponential ElGamal protocol requires 1.56 MB of bandwidth to transmit ciphertexts and proofs for 0.13 MB of raw data to the client, representing an 11.95 times bandwidth overhead. In contrast, the Paillier protocol has even higher bandwidth demands : 6.55 MB

for the same volume of raw data (50.18 times bandwidth overhead).

Despite its computational advantages, the Paillier protocol does not effectively reduce bandwidth costs compared to the Exponential ElGamal protocol.

Metric	Data Size	**ElGamal**	**Paillier**
Prov. Time	0.13 MB	89 sec	5 sec
	1 GB	**194 hrs**	**11 hrs**
Verif. Time	0.13 MB	34.15 sec	19.45 sec
	1 GB	**74 hrs**	**42 hrs**
‖ctxt + proofs‖	0.13 MB	1.56 MB	6.55 MB
	1 GB	**12 GB**	**50 GB**

Table 1: Performance metrics for ElGamal and Paillier schemes for 0.13 MB and 1 GB of data, chosen to illustrate a realistic application scenario.

6.0.2 On-Chain Costs

On-chain operations are managed by smart contract logic on either Bitcoin or Ethereum to support the transaction processes.

Bitcoin: The fee for the bonding contract is expected to be below $10 for a 1-hour confirmation time, regardless of the data size. This contract includes two conditional executions: one utilizes a timelock allowing client access post-timeout, and the other allows the release of funds with the appropriate signatures.

Ethereum: Smart contracts on the Ethereum Virtual Machine (EVM) use four critical functions, as detailed in Figure 4. The server is allowed to withdraw funds once it provides the decryption key, whereas clients can reclaim their payments if the decryption key is retained. The costs listed in the table are based on the ETH exchange rate as of February 3, 2024. Since that time, the Ethereum prices have dropped to approximately 1'678.42 USD/ETH as of March 14, 2025, lowering transaction costs.

7 Multi-Client Model

In situations where a server needs to send the same data to many clients, running a separate exchange for each one can be inefficient and expensive. The Multi-Client-VECK protocol helps by letting the server preprocess the encryption and proofs once and use it for all clients, reducing costs significantly.

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 $

Figure 4: EVM gas costs for smart contract functions in Exponential ElGamal and Paillier-based FDE schemes. Exchange rate on February 3, 2024. 2,302.35 USD/ETH

However, a problem arises if clients start sharing the decryption keys among themselves without paying. To prevent this, the MC-VECK protocol makes sure each client must get their decryption key directly from the server.

8 Conclusion

This paper demonstrates that while FDE protocol implementations can reduce on-chain transaction sizes, they pose significant practical challenges. Notably, off-chain costs are high due to intensive computation and data transmission overheads. For example, processing a 1GB file with these protocols significantly increases bandwidth usage and processing time, limiting their practical application in real-world scenarios. The MC-VECK setting attempts to amortize computation costs across multiple clients by preprocessing encryption and proofs. However, this introduces security challenges, as clients might share decryption keys to avoid payments. Future developments need to focus on minimizing the size of encrypted data and proofs and reducing computation and verification times to enhance the feasibility of FDE protocols for real-world adoption.