Atomic and Fair Data Exchange via Blockchain

Ertem Nusret Tas

Stanford University

The Science of Blockchain Conference 2024 (SBC'24)

István András Seres
Eötvös Loránd U.

Yinuo Zhang
UC Berkeley

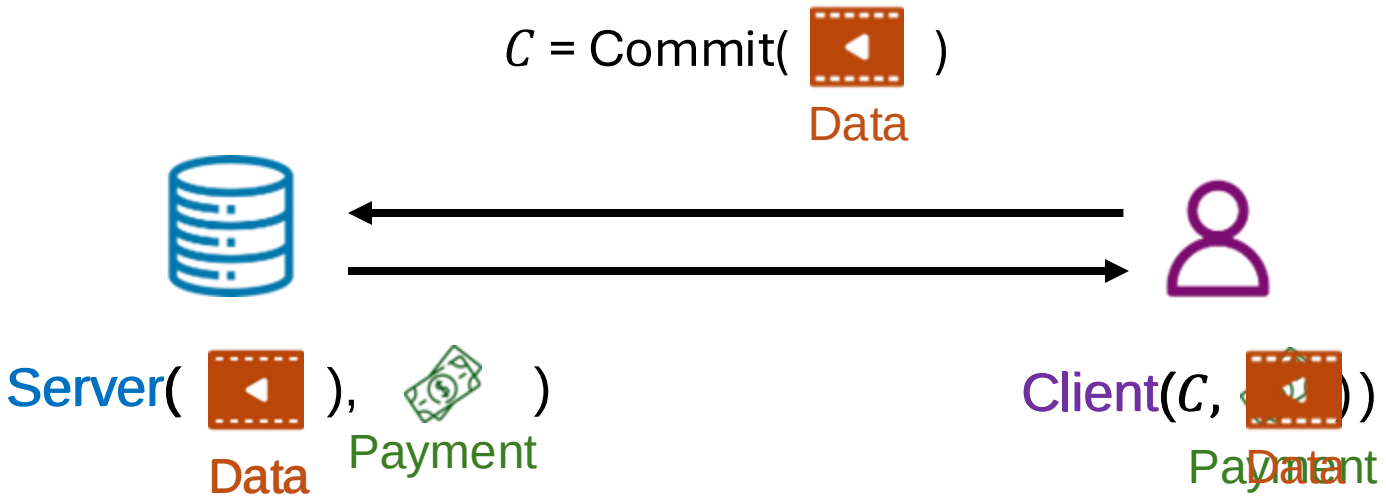
Márk Melczer
Eötvös Loránd U.

Mahimna Kelkar
Cornell Tech

Joseph Bonneau
a16z crypto research & NYU

Valeria Nikolaenko
a16 crypto research

The Fair Data Exchange (FDE) Problem



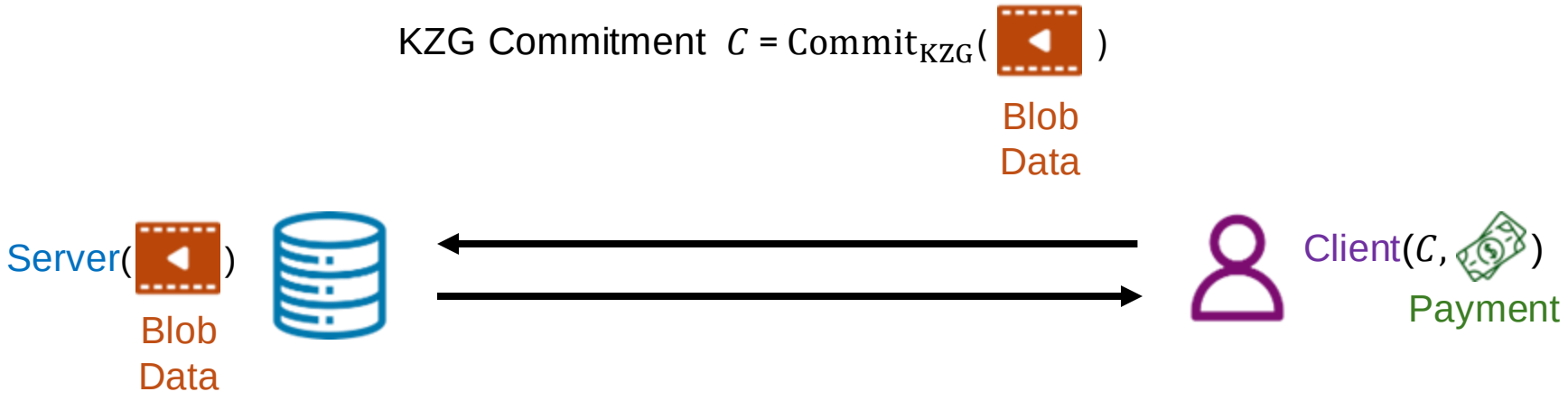
Server Fairness:
An adversarial client cannot learn anything about the data without paying the server.

Correctness:
If the client and the server are honest, the client obtains the data, and the server obtains the payment.

Client Fairness:
An adversarial server cannot receive any payment if the client does not obtain the data.

Applications of FDE over Blockchains

ProtoDanksharding (EIP-4844)

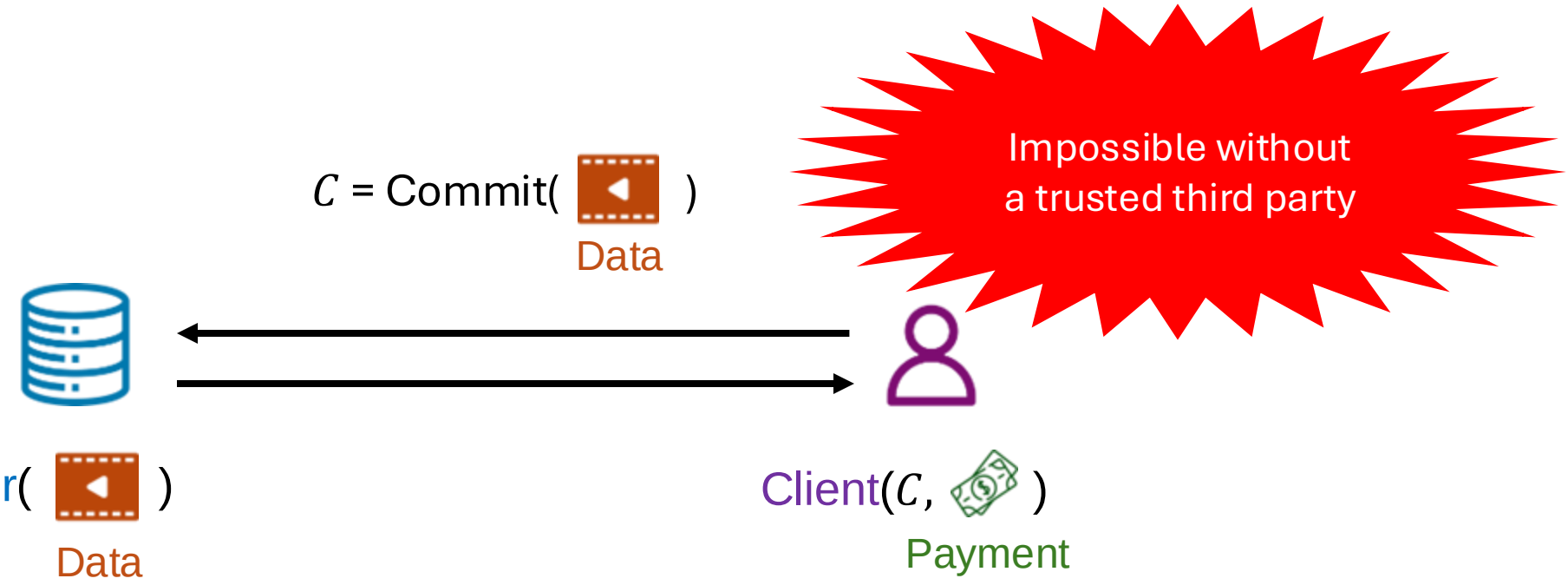


Other Applications:

- Other archival blockchain data
- Data marketplaces (e.g., streaming movies, …)

The Fair Data Exchange (FDE) Problem

TTP?
Blockchains!

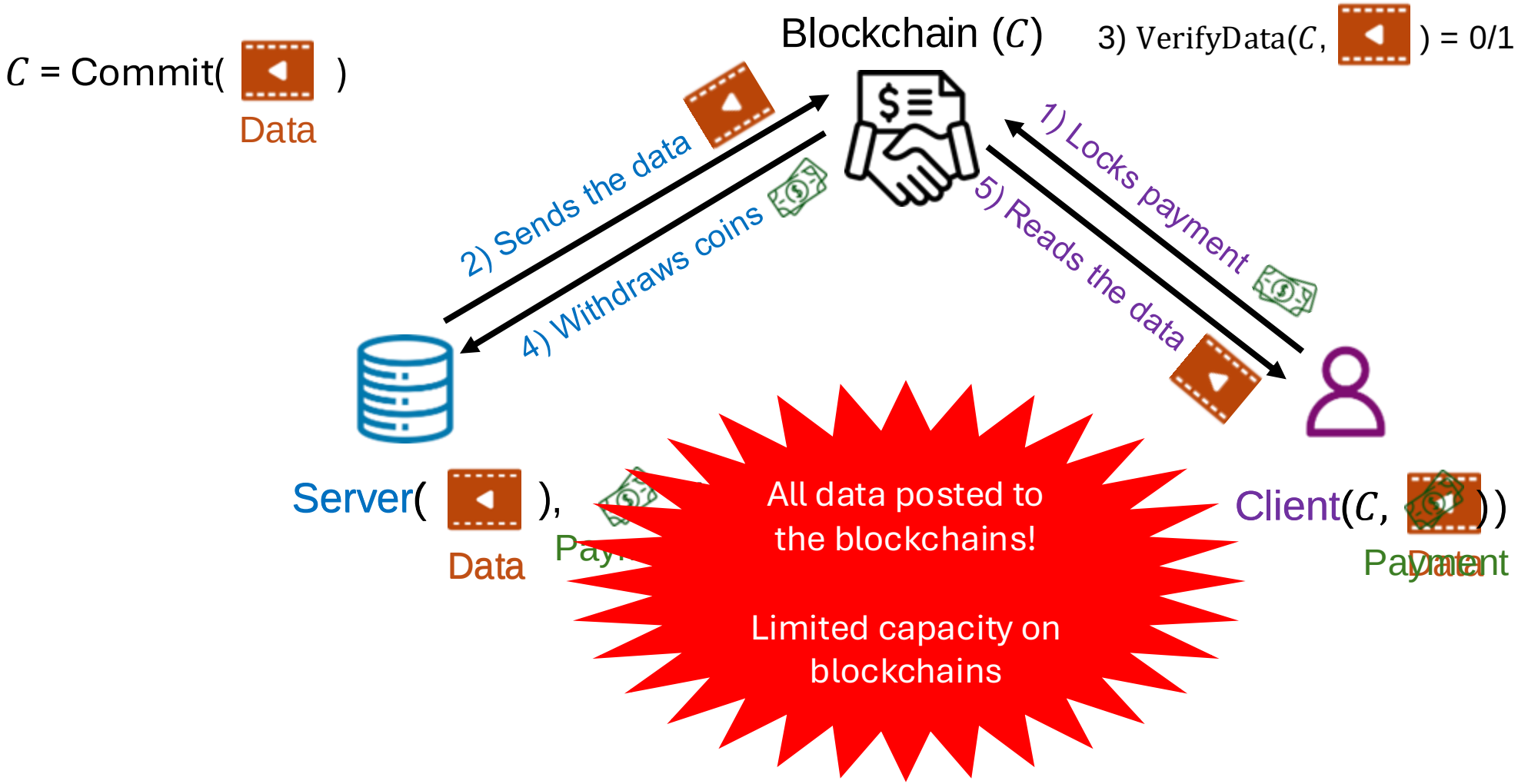


Server Fairness:
An adversarial client cannot learn anything about the data without paying the server.

Correctness:
If the client and the server are honest, the client obtains the data, and the server obtains the payment.

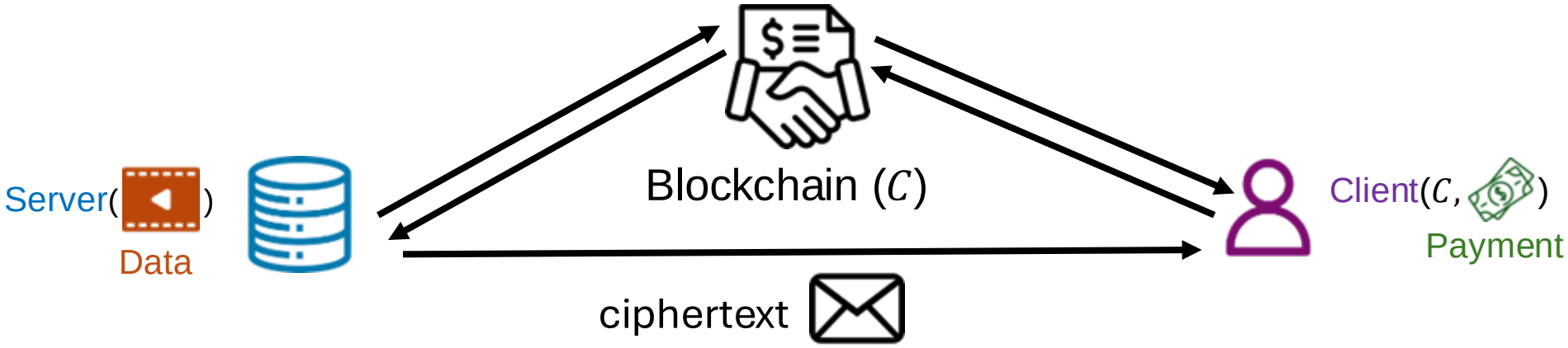
Client Fairness:
An adversarial server cannot receive any payment if the client does not obtain the data.

Strawman Solution



The FDE Protocol

- Server encrypts the data.
- Transfers the encrypted data off-chain **(large off-chain bandwidth)**
- Sells the key on-chain **(small on-chain bandwidth)**



Server must prove that the key sold on-chain decrypts the ciphertext to the data committed by C!

Verifiable Encryption under Committed Key (VECK)

- $\mathrm{Gen}(\mathrm{crs}) \to \mathrm{pp}$

- $\mathrm{Enc}(C, w) \to (vk, sk, ct, \pi)$

- $\mathrm{Verify}_{\mathrm{key}}(vk, sk) \to 1/0$

- $\mathrm{Verify}_{\mathrm{ct}}(C, vk, ct, \pi) \to 1/0$

- $\mathrm{Decrypt}(sk, ct) \to w/\perp$

Proof π: ct is the encryption of the data committed by C under the secret key sk committed by the verification key vk.

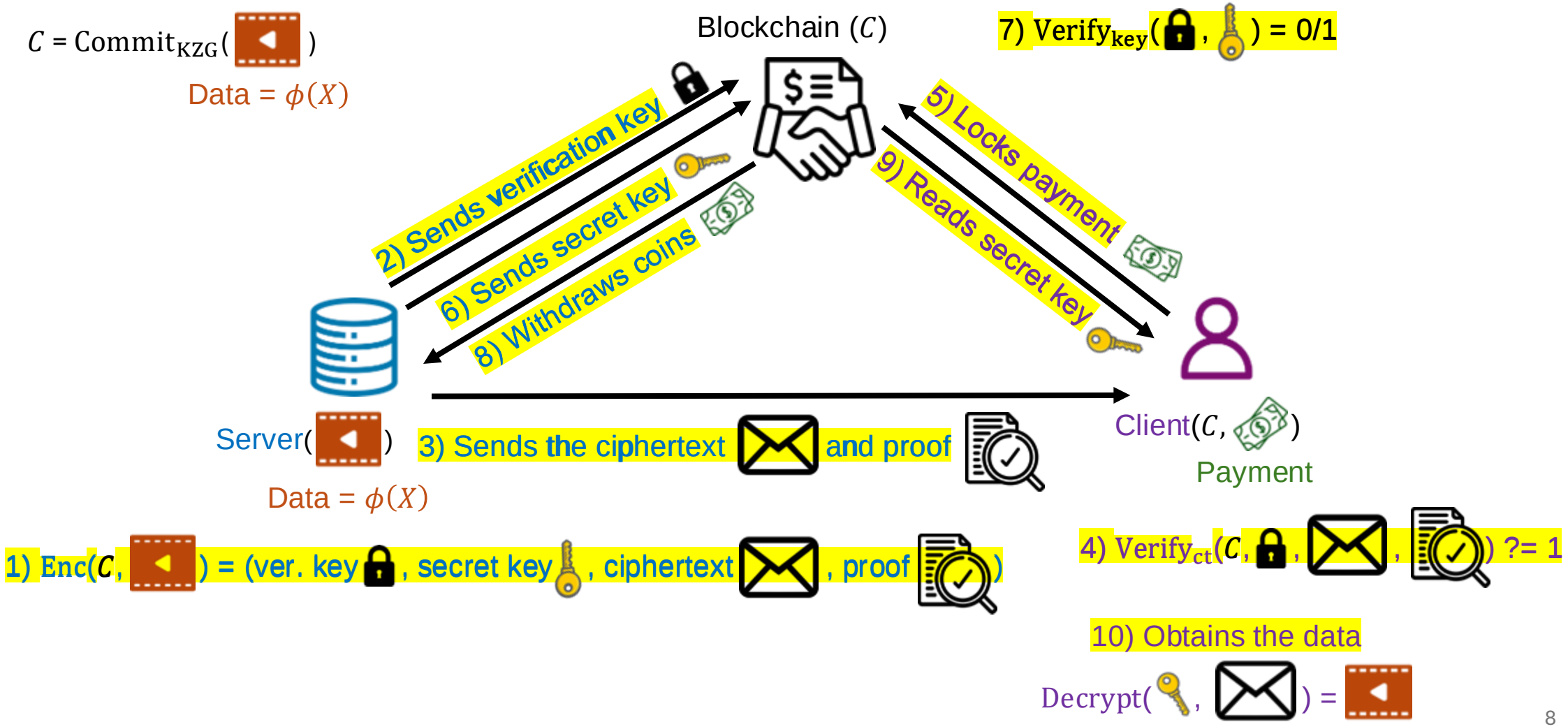
Correctness:
Verifications for honestly generated encryption succeed.
⇒ **correctness of FDE**

Soundness:
No PPT adversary can generate sk, vk, ct, π such that verification succeeds, yet decryption does not output w.
⇒ **client-fairness of FDE**

Computational Zero-Knowledge:
The ciphertext and the proof leak no additional information about the witness.
No client can help the others recover w.
⇒ **server-fairness of FDE**

VECK for KZG Commitments: $w = \phi, \mathrm{data} = \big(\phi(0), \ldots, \phi(l)\big), \quad C = \mathrm{Commit}_{\mathrm{KZG}}(\phi)$

The FDE Protocol for KZG Commitments



ElGamal-based VECK for KZG Commitments

$$\text{pp} = (h, h_0, \ldots, h_l)$$

Prover (Server):

$\mathbf{Enc}(\boldsymbol{C}, \boldsymbol{\phi}(\boldsymbol{X})) \rightarrow (\boldsymbol{sk}, \boldsymbol{vk}, \boldsymbol{ct}, \boldsymbol{\pi})$

Sample $s \leftarrow_R \mathbb{F}_p$

Set $sk := s$ (ElGamal secret key)

$vk := h^s$ (ElGamal verification key)

Data points: assume small

$\boldsymbol{ct} := \left\{h_i^s g_1^{\phi(i)}\right\}_{i=0}^{l}$ (ElGamal ciphertexts)

Generate $\pi :=$ SNARK. Prove$(crs,\ \text{instance} = (C, vk, ct),$ witness $= (s, \phi(X)))$

$vk, \boldsymbol{ct}, \pi$

Verifier (Client):

$\mathbf{Verify}_{\mathbf{ct}}(\boldsymbol{C}, \boldsymbol{vk}, \boldsymbol{ct}, \boldsymbol{\pi}) \rightarrow \mathbf{0/1}$

Output 1 if π verifies against C, vk, ct

$\mathbf{Decrypt}(\boldsymbol{pp}, \boldsymbol{sk}, \boldsymbol{ct}) \rightarrow \boldsymbol{w}/\perp$

Decryption algorithm for ElGamal

Blockchain Contract:

$\mathbf{Verify}_{\mathbf{key}}(\boldsymbol{pp}, \boldsymbol{vk}, \boldsymbol{sk}) \rightarrow \mathbf{1/0}$

Check if $h^{sk} == vk$

More on VECK for KZG Commitments

Requires inclusion of the 'encryption' as part of the SNARK relation → Potential effects on efficiency!

An efficient VECK protocol that exploits the (shared) structure of the ElGamal ciphertexts and KZG commitments! Can support large messages by splitting the messages into $k \in [8,16]$ chunks with range proofs.

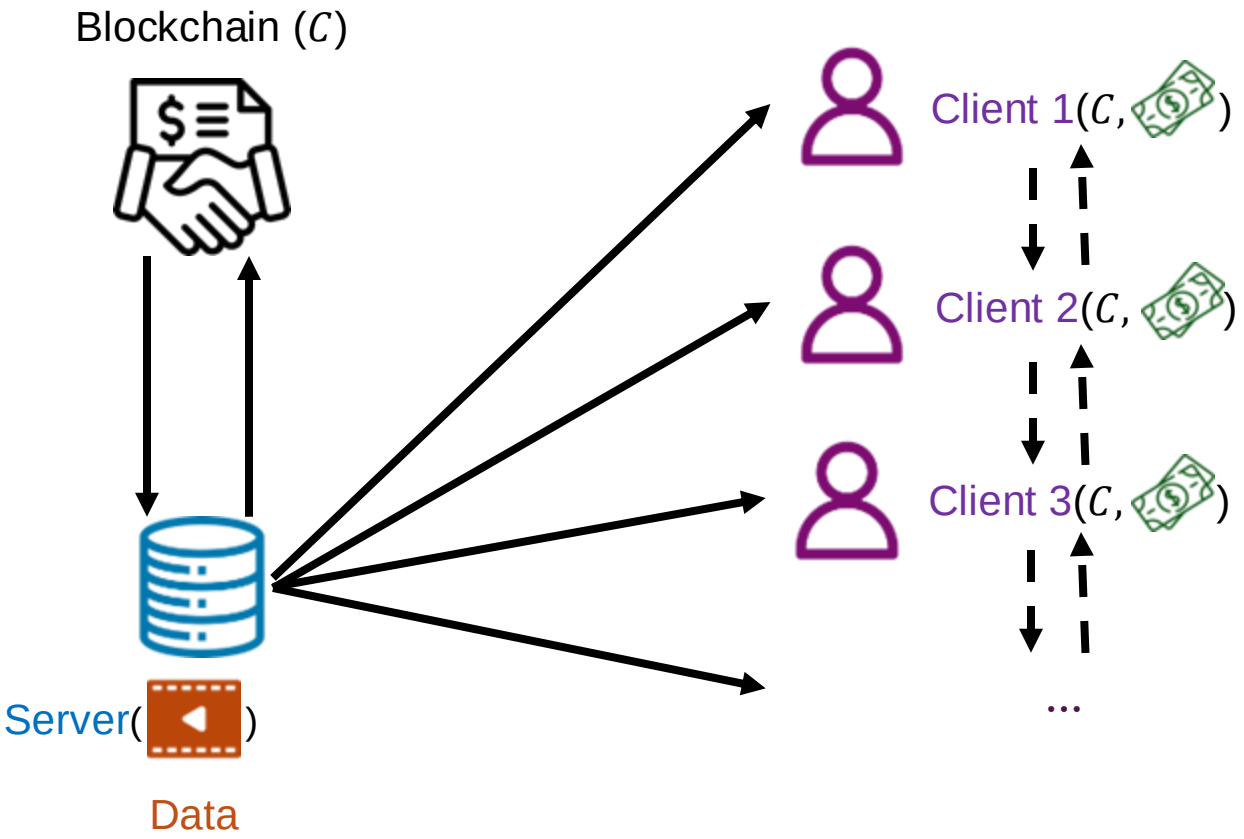
But, blows up ciphertext size by k!

Paillier-based VECK for KZG commitments to avoid the ciphertext blow-up!

Thm: Assuming Decisional Diffie-Hellman (DDH) assumption, ElGamal-based VECK protocol satisfies security (correctness, soundness and computational zero-knowledge) in the random oracle and algebraic group models.

Thm: Assuming Decisional Composite Residuosity (DCR) assumption, Paillier-based VECK protocol satisfies security in the random oracle and algebraic group models.

Multi-client Model



Can the server save work by amortizing proof generation across many clients?

Strawman

Reuse the same sk, vk, ct across all clients.

Can share $o(|\text{Data}|)$ bits with each other $\Rightarrow$ Must rerandomize sk across clients!

Multi-client VECK (MC-VECK) protocol:
Prover saves work by moving parts of the proof generation to a preprocessing step.

No need to generate **new ciphertexts** and **range proofs** per client!

Implementation: Prover Time

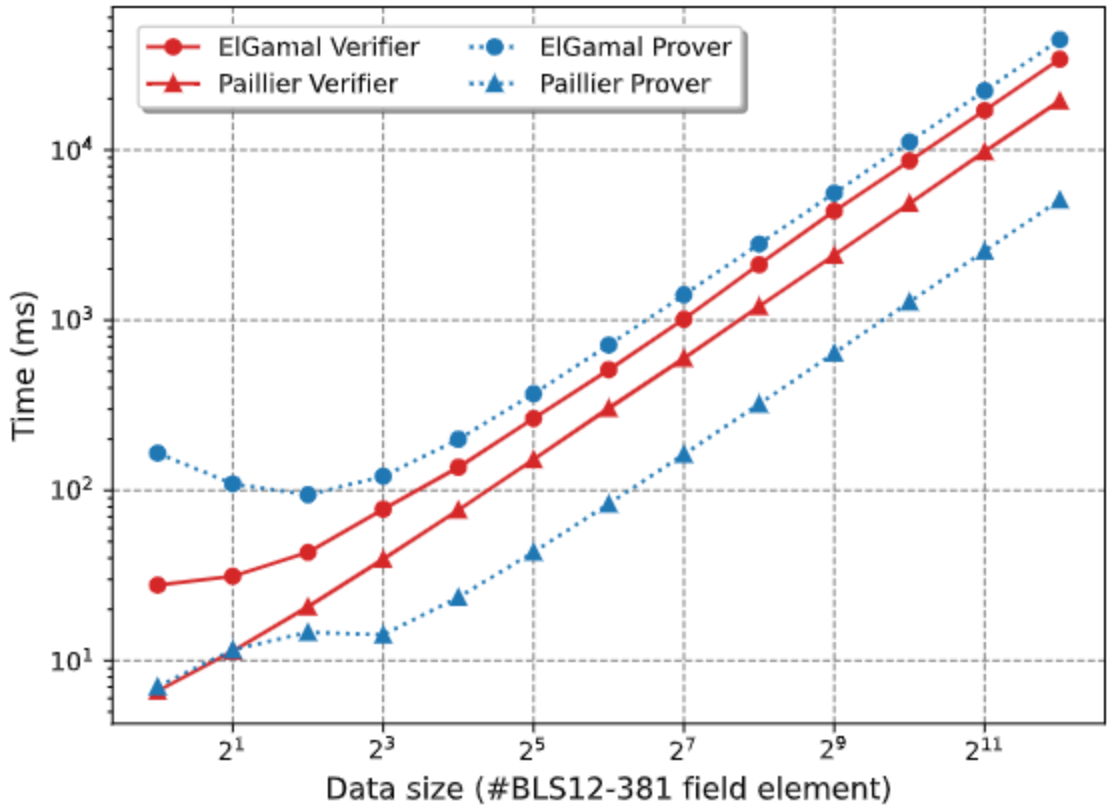
- PoC implementation in **Rust v1.74.0** on **consumer PC** with **AMD Ryzen 5 3600 (6-core) CPU** and **8GB RAM**.
- Criterion benchmarking **crate8**.

Prover time for 4096 BLS12-381 field elements (≈ 128 kB):

Exponential ElGamal encryption ($k = 8$):

- Range proofs + ciphertexts: $\approx$ **89 s**
- Overhead for proving the consistency of ciphertexts w.r.t. C and vk: $<$ **40 ms**

Paillier encryption: $\approx$ **5.09 s**.



Implementation: Proof Size, Verification & Decryption

Proof size for 4096 BLS12-381 field elements (≈ 128 kB):

Exponential ElGamal encryption ($k = 8$):

- Ciphertexts & proofs total size: $\mathbf{1,56}$ **MB**
- $\approx \mathbf{12}\times$ bandwidth overhead
- Proof has constant size ($6|\mathbb{G}|$).

Paillier encryption:

- Ciphertexts & proofs total size: $\mathbf{6,55}$ **MB** ($\lambda = 128, \log_2 N = 3072$)
- $\approx \mathbf{50}\times$ bandwidth overhead
- Proof has linear size in the data.

Verification & decryption time for 4096 BLS12-381 field elements (≈ 128 kB):

Exponential ElGamal encryption ($k = 8$):

- Verification time: $\mathbf{34.15}$ **s**
- Decryption time: Quick with lookup tables.

Paillier encryption:

- Verification time: $\mathbf{19.45}$ **s**
- Decryption time: $\mathbf{9.54}$ **s**

Implementation: On-chain Costs

- Constant in the size of the exchanged data: 3 signatures, 1 vk, 1 sk!

Ethereum (L1) costs:

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 $

Registers server's verification key with the on-chain contract, sets the data price. It

Locks client's payment according to the server's verification key

Enables the server to withdraw the payment

On L2, can be made 200 × less!

Conclusion and Future Directions

FDE and VECK protocols for KZG commitments with application to Danksharding.

More in the Paper:

- Using Bitcoin as the TTP via adaptor sigs.
- Supporting VECK for subset openings.
- Framework to design alternative FDE & VECK protocols for different commitment schemes.

Future Directions:

- Paillier-encryption with constant-size proofs.
- Other commitment and encryption schemes?
- Server griefing?
- Pricing the data?

Blog post

Paper

Open-source implementation

‘FairSwap: How To Fairly Exchange Digital Goods’ CCS 2018
‘Fair Peer-to-Peer Content Delivery via Blockchain’ ESORICS 2021
‘FileBounty: Fair Data Exchange’ IEEE EuroS&P 2020
‘ZKCPlus: Optimized Fair-exchange Protocol Supporting Practical and Flexible Data Exchange’, CCS 2021

Appendix: Related Work

	#rounds	Data struct.	$\lvert\pi_{\mathbf{disp}}\rvert$	S to C	C to B	S to B	Online?
FairSwap	5	Merkle tree	$3\log(k)\,\lvert\mathbb{H}\rvert$	$(k+1)\lvert\mathbb{H}\rvert$	$2\lvert\mathbb{H}\rvert+\lvert\sigma\rvert$	$2\lvert\sigma\rvert+\lvert\mathbb{G}\rvert$	✓
FileBounty	k	Merk.-Dam.	$3\lvert\mathbb{G}\rvert$	$k(\lambda+\lvert\mathbb{H}\rvert)$	$k\lvert\sigma\rvert$	$2\lvert\sigma\rvert$	✗
FairDownload	k	Merkle tree	$\log(k)\,\lvert\mathbb{H}\rvert$	$k\lvert\mathrm{ct}\rvert$	$k\lvert\sigma\rvert$	$2\lvert\sigma\rvert+O(\log k)\lvert\mathbb{H}\rvert$	✗
FDE-ElGamal	3	KZG	N/A	$k\lvert\mathbb{G}\rvert+6\lvert\mathbb{G}\rvert$	$\lvert\sigma\rvert$	$2\lvert\sigma\rvert+\lvert\mathbb{G}\rvert+\lvert\mathbb{F}_p\rvert$	✓
FDE-Paillier	3	KZG	N/A	$k(2\lvert\mathrm{ct}\rvert+\lvert\mathbb{F}_p\rvert)$	$\lvert\sigma\rvert$	$2\lvert\sigma\rvert+\lvert\mathbb{G}\rvert+\lvert\mathbb{F}_p\rvert$	✓

$\lvert\mathbb{H}\rvert$, $\lvert\mathbb{G}\rvert$ and $\lvert\mathbb{F}_p\rvert$ refer to the size of a single hash function output, (an elliptic curve or) group element, and field element, respectively, whereas $\lvert\sigma\rvert$ refers to the signature size. If there is a dispute protocol involved, $\lvert\boldsymbol{\pi}_{\mathbf{disp}}\rvert$ is the size of the submitted proof.

ZKCPlus: MiMC-p/p block cipher in CTR mode + commit-and-prove NIZKs (CP-NIZKs)

Appendix: ElGamal-based VECK for KZG Commitments

Prover (Server):
$\mathbf{Enc}(\boldsymbol{C}, \boldsymbol{\phi}(\boldsymbol{X})) \rightarrow (\boldsymbol{sk}, \boldsymbol{vk}, \boldsymbol{ct}, \boldsymbol{\pi})$

Verifier (Client):
$\mathbf{Verify_{ct}}(\boldsymbol{C}, \boldsymbol{vk}, \boldsymbol{ct}, \boldsymbol{\pi}) \rightarrow \mathbf{0/1}$

1. Sample $s \leftarrow_R \mathbb{F}_p$, set $sk := s, vk := h^s, \boldsymbol{ct} := \left\{h_i^s g_1^{\phi(i)}\right\}_{i=0}^{l}$

$\boldsymbol{vk}, \boldsymbol{ct}$

Can be made non-interactive with Fiat-Shamir

$\alpha \leftarrow_R \mathbb{F}_p$

To ensure zero-knowledge

2. Compute $C_\alpha := g_1^{\phi(\alpha)+s(\tau-\alpha)}$
3. Compute KZG proof π_α for $\phi(X) - s(X-\alpha)$ at α.
4. Compute a proof π_{LIN} that $C_\alpha = g_1^u (g_1^{\tau-\alpha})^v$ and $vk = 1^u h^v$ for the same v and known u.
5. Compute $Q := g_1^{-(\tau-\alpha)} \prod_{i=0}^{l} h_i^{L_{i,l}(\alpha)}$
6. Compute DLOG eq. proof π_{DLeq} for (Q, Q^s, h, vk).

$\boldsymbol{\pi} = (C_\alpha, \pi_\alpha \pi_{LIN}, \pi_{DLeq})$

Consistency of C and C_α

Consistency of vk and C_α

Consistency of $\boldsymbol{ct}$ and C_α

1. Verify π_{LIN} against $g_1, g_1^{\tau-\alpha}, h, C_\alpha, vk$.
2. Verify π_α : $e(C/\, C_\alpha, g_2) = e(\pi_\alpha, g_2^{\tau-\alpha})$
3. Compute Q, $Q^* := C_\alpha^{-1} \prod_{i=0}^{l} \boldsymbol{ct}_i^{L_{i,l}(\alpha)}$
4. Verify π_{DLeq} against (Q, Q^*, h, vk).
5. Output 1 iff all checks succeed.

Appendix: ElGamal-based VECK for KZG Commitments

Why is it secure?

- $\boldsymbol{sk} \coloneqq s, \boldsymbol{vk} \coloneqq h^s, \boldsymbol{ct} \coloneqq \left\{h_i^s g_1^{\phi(i)}\right\}_{i=0}^{l}$
- $\boldsymbol{C_\alpha} \coloneqq g_1^{\phi(\alpha)+s(\tau-\alpha)}$
- $\boldsymbol{\pi_\alpha}$: KZG proof for $\phi(X) - s(X-\alpha)$ at α.
- $\boldsymbol{\pi_{LIN}}$: proof that $C_\alpha = g_1^u (g_1^{\tau-\alpha})^v$ and $vk = 1^u h^v$ for the same v and known u.
- $\boldsymbol{Q} \coloneqq g_1^{-(\tau-\alpha)} \prod_{i=0}^{l} h_i^{L_{i,l}(\alpha)}$
- $\boldsymbol{\pi_{DLeq}}$: DLOG eq. proof for (Q, Q^s, h, vk).

1. π_{LIN} proves that $C_\alpha = g_1^{u+s(\tau-\alpha)}$ given $vk = h^s$.
2. Given (1), π_α proves that $C_\alpha = g_1^{\phi(\alpha)+s(\tau-\alpha)}$.
3. Given (1) and 2, π_{DLeq} proves that $\boldsymbol{ct}$ are ElGamal encryptions of $\phi(i), i = 0, \dots, l$; since then

$$Q^* = C_\alpha^{-1} \prod_{i=0}^{l} ct_i^{L_{i,l}(\alpha)}$$

$$= g_1^{-\phi(\alpha)-s(\tau-\alpha)} \prod_{i=0}^{l} h_i^{sL_{i,l}(\alpha)} g_1^{\phi(i)L_{i,l}(\alpha)}$$

$$= g_1^{-\phi(\alpha)-s(\tau-\alpha)} g_1^{\phi(\alpha)} \prod_{i=0}^{l} h_i^{sL_{i,l}(\alpha)} \left(g_1^{-(\tau-\alpha)} \prod_{i=0}^{l} h_i^{L_{i,l}(\alpha)}\right)^s = Q^s$$

Appendix: Multi-client VECK Protocol

Prover saves work by moving parts of the proof generation to a preprocessing step.

Prover (Server) Preprocessing:
$\mathbf{Prep}(\boldsymbol{C}, \boldsymbol{\phi}(\boldsymbol{X})) \rightarrow (\mathbf{aux}, \mathbf{msk})$

1. Run $\mathrm{Enc}(C, \phi(X)) \rightarrow (sk, vk, ct, \pi)$
2. Output $\mathrm{aux} = (vk, ct, \pi)$ and $\mathrm{msk} = sk$

Prover (Server) per client c:
$\mathbf{Enc}(\mathbf{aux}, \mathbf{msk}) \rightarrow (\boldsymbol{sk_c}, \boldsymbol{vk_c}, \boldsymbol{ct}, \boldsymbol{\pi_c})$

1. Sample $\delta_c \leftarrow_R \mathbb{F}_p$ To rerandomize sk
2. Calculate $Q = \prod_{\{i \in [n]\}} h_i^{e_i}$, where $e_i := H(h^{\delta_c}, i)$.
3. Compute DLOG eq. proof π_{DLeq} for $(Q, Q^{\delta_c}, h, h^{\delta_c})$.
4. $sk_c := sk + \delta_c$, $vk_c := vk \cdot h^{\delta_c}$,

$$\pi_c = \left(\pi, D_c := h^{\delta_c}, \pi_{DLeq}, \left(h_{c,i} := h_i^{\delta_c}\right)_{i=0}^{l}\right)$$

Consistency of $h_{c,i}$ and D_c

Verifier (Client):
$\mathbf{Verify}_{\mathbf{ct}}(\boldsymbol{C}, \boldsymbol{vk_c}, \boldsymbol{ct}, \boldsymbol{\pi_c}) \rightarrow \mathbf{0/1}$

1. Parse $\left(\pi, D_c, \pi_{DLeq}\right) \leftarrow \pi_c$
2. Compute Q, $\mathrm{Q}^* := \prod_{i=0}^{l} h_{c,i}^{e_i}$
3. Verify π_{DLeq} against (Q, Q^*, h, D_c).
4. Output 1 if all checks succeed & $\mathrm{Verify}_{\mathrm{ct}}(C, vk_c / D_c, ct, \pi) = 1$