A BLAKE3-Verified Metered Data Exchange Protocol

Jay Logelin

April 16, 2025

Abstract

We present a protocol for exchanging data in a metered fashion between two parties, a Provider (P) and a Consumer (C). The protocol ensures fair payment for each data chunk, leveraging ephemeral encryption keys and chunk-level BLAKE3 hashing. Additionally, an overall BLAKE3 hash enforces end-to-end data integrity verification. Our protocol limits the impact of misbehavior by either party to a single chunk at a time and enables streaming verification of large data sets.

1 Introduction

In many scenarios, one party (the *Provider*, P) possesses data to be sold, while another party (the *Consumer*, C) wishes to purchase it. A straightforward approach where C pays upfront for the entire data and then trusts P to deliver the correct content is fraught with risks:

- P might not deliver the data after receiving payment.
- P might deliver corrupted or incomplete data.
- C, on the other hand, might refuse to pay for data after obtaining it.

To mitigate these issues, we propose a **metered exchange protocol** in which data is chunked, encrypted, and sold in small increments. Each increment is validated for integrity, and payment is made before the decryption key is released. Moreover, chunk-level verification is supplemented by an overall BLAKE3 integrity check after reconstructing all chunks.

2 System Model & Notation

2.1 Parties

- P: The **Provider**, who owns the full data D of length $|D|$ bytes.
- C: The **Consumer**, who wishes to acquire D in exchange for tokens.

2.2 Notation

- D: The full data (byte string) of length $n = |D|$.
- m: A chosen *chunk size* in bytes.
- $N = \lceil n/m \rceil$: The total number of chunks.

- D_i: The i-th chunk of D, where $D_i \in \{0,1\}^m$ (except possibly the last chunk if n is not a multiple of m).
- K_i: An *ephemeral key* (symmetric) for encrypting D_i. Typically, $K_i \in \{0,1\}^k$ (e.g., $k = 128$ or 256 bits).
- $\mathsf{Enc}(K_i, D_i)$: A secure encryption function returning $(C_i, nonce_i, tag_i)$, e.g., AES-GCM.
- $\mathsf{Dec}(K_i, C_i, nonce_i, tag_i)$: The corresponding decryption function.
- $\mathsf{H}_{\mathrm{B3}}(\cdot)$: The BLAKE3 hash function (returning a fixed-size digest).
- $h_i = \mathsf{H}_{\mathrm{B3}}(D_i)$: The BLAKE3 hash of the *plaintext* chunk D_i.
- $H_{\mathrm{full}} = \mathsf{H}_{\mathrm{B3}}(D)$: The BLAKE3 hash of the entire data D, often computed in a streaming manner.
- p_i: The *price* in tokens that C must pay to obtain the key K_i for chunk i.

2.3 Assumptions

Assumption 1. *Both* P *and* C *have a secure communication channel to exchange data chunks and ephemeral keys. This can be done over a network with TLS, or via a secure messaging protocol.*

Assumption 2. *Payments are* atomic *from* C *to* P*, meaning* C *either pays exactly* p_i *tokens or does not pay at all. We assume an out-of-band ledger, payment channel, or on-chain mechanism can guarantee that once a payment is sent,* P *observes it irrevocably.*

Assumption 3. *BLAKE3 is collision-resistant and cryptographically secure for the chunk-level and overall hash verifications. AES-GCM (or any secure AEAD scheme) is used for encryption, ensuring confidentiality and integrity of each chunk.*

3 Protocol Specification

3.1 Preparation by P

P1. Chunking: Split D into N chunks $\{D_1, D_2, \ldots, D_N\}$, each of size m bytes (the last chunk may be shorter).

P2. Hashing: For each chunk D_i, compute $h_i = \mathsf{H}_{\mathrm{B3}}(D_i)$.

P3. Ephemeral Keys: Generate N random keys $\{K_1, K_2, \ldots, K_N\}$ using a secure random generator.

P4. Chunk Encryption: For each chunk D_i, compute $(C_i, nonce_i, tag_i) = \mathsf{Enc}(K_i, D_i)$.

P5. Overall Hash: Compute $H_{\mathrm{full}} = \mathsf{H}_{\mathrm{B3}}(D)$ (or do it in streaming fashion).

P6. Set Prices: Assign a cost p_i for each chunk i (for simplicity, set $p_i = p$ constant or vary as desired).

At the end of the preparation, P holds:

$$\{(C_i, nonce_i, tag_i), K_i, h_i, p_i\}_{i=1}^{N} \quad \text{and} \quad H_{\mathrm{full}}.$$

3.2 Chunk Acquisition by C

C1. Request Encrypted Chunk: For some chunk index i, C requests $(C_i, nonce_i, tag_i)$ from P.

C2. Verification of Metadata: (Optional) C can check if $(C_i, nonce_i, tag_i)$ hash matches P's commitment to detect trivial tampering. (In practice, C may rely on h_i *after* decryption.)

C3. Payment for Key: C transfers p_i tokens to P for chunk i. If C lacks sufficient tokens, the protocol aborts.

3.3 Key Disclosure by P

P2. Upon observing the payment for chunk i, P reveals the ephemeral key K_i to C.

3.4 Decryption & Verification by C

C2. Decrypt Chunk: C computes $\widehat{D_i} = \mathsf{Dec}(K_i, C_i, nonce_i, tag_i)$.

C2. Chunk-Level Hash Check: C verifies that $\mathsf{H}_{\mathrm{B3}}(\widehat{D_i}) = h_i$. If it does not match, C aborts and refuses to download (and pay for) further chunks.

C2. Streaming Hash Update: C appends $\widehat{D_i}$ to a local reassembly buffer or streaming hasher for final verification. Specifically, C maintains a BLAKE3 streaming state S. After decrypting chunk i, C runs:

$$S \leftarrow S \| \widehat{D_i}.$$

3.5 Final Assembly and Overall Hash Verification

C3. After acquiring all N chunks in sequence or in any chosen order, C finalizes the local BLAKE3 streaming hasher over the concatenation $\widehat{D_1} \| \widehat{D_2} \| \ldots \| \widehat{D_N}$.

$$\widehat{H_{\text{full}}} = \mathsf{H}_{\mathrm{B3}}\big(\widehat{D_1} \| \cdots \| \widehat{D_N}\big).$$

C3. C compares $\widehat{H_{\text{full}}}$ to H_{full}, which P advertised initially. If $\widehat{H_{\text{full}}} \neq H_{\text{full}}$, C concludes the data was not consistent with P's original commitment.

4 Security Analysis

4.1 Correctness

The protocol is correct if:

1. For each chunk i, C pays p_i to P and obtains K_i.
2. The decryption of C_i using K_i yields D_i without error.
3. The final assembled data $\widehat{D_1} \| \cdots \| \widehat{D_N}$ matches the original data D if no adversarial corruption occurs.

4.2 Fairness & Chunk-by-Chunk Payment

Because the protocol exchanges each key K_i for payment p_i, C only risks losing the funds for that *one chunk* in the event P withholds the correct key. If C detects cheating (e.g., a bad chunk hash), it can stop downloading further chunks and not lose more tokens. Likewise, P only risks revealing a single chunk's key if C fails to pay for the next chunk.

4.3 Integrity via BLAKE3 Hashing

BLAKE3 hashing is used at two levels:

- **Chunk-level hash** h_i ensures each decrypted chunk $\widehat{D_i}$ matches D_i and detects any in-transit corruption or tampering.
- **Overall hash** H_{full} ensures the final reassembly $\widehat{D}$ is the same as P's original data.

Any mismatch in h_i or H_{full} indicates malicious behavior or corruption.

4.4 Limitations

1. The protocol does not, by itself, enforce that P must provide all keys (a malicious P can stop at any time). However, chunk-by-chunk exchange minimizes the financial impact on C.
2. A purely off-chain approach might require trust or external arbitration. For fully trustless operation, integrate with a blockchain-based payment system or a channel that atomically exchanges tokens for ephemeral keys (e.g., hashed time-locked contracts).

5 Conclusion

We have presented a metered data exchange protocol featuring ephemeral encryption keys, chunk-level BLAKE3 integrity verification, and an overall BLAKE3 streaming integrity check. By processing data chunk by chunk, the protocol makes cheating less profitable and reduces reliance on trust assumptions. Integrating with secure payment channels or on-chain logic can further enhance fairness, making this protocol suitable for large-scale data exchanges in potentially adversarial environments.

References

[1] J. Oeschlin, J. Teutsch, and J. Blandy. *BLAKE3: One function, fast everywhere.* 2020. `https://github.com/BLAKE3-team/BLAKE3`

[2] Rogaway, P. *Evaluation of some blockcipher modes of operation.* 2011.