

# Secure Transmission of Electronic Health Records over a Hybrid Blockchain Ledger with Sensitivity-Aware Adaptive Anchoring

Crystal Darling B<sup>1</sup>, V. Rajasekar<sup>2</sup>

<sup>1,2</sup>Department of Computer Science and Engineering, SRM Institute of Science and Technology  
Vadapalani, Chennai, Tamil Nadu, India

## Abstract

Hospitals keep a patient's history in separate systems, and a transfer between them has usually depended on an intermediary. Blockchain schemes remove that intermediary, but a consent check that only releases an unrelated key, an anchor that fires on a fixed interval, and a history the patient cannot check without a consortium member, are still common. HyL-EHR keeps consent, access logs and record fingerprints on a permissioned ledger, stores ciphertext off-chain, and anchors the private ledger to a public chain that sees only Merkle roots. Consent-Bound Key Derivation salts every session key with the consent digest each party reads from its own replica. Sensitivity-Aware Adaptive Anchoring commits a revocation or an emergency access at once and batches the remaining traffic under a daily anchor budget. A dual-ledger proof can be checked against the public chain alone.

A Python prototype and a Solidity anchor contract were measured on one machine, with random bytes in a record envelope and no real patient data. A 4 KiB record is transferred, receipted and logged in 1.118 ms. The custodian's cost stays between 0.695 ms and 0.880 ms from 4 KiB to 64 MiB. With the daily budget set to 288 anchors, the policy used 298 anchors and cut the mean public-exposure window of a critical event from 161.8 s to 6.9 s. Routine traffic waited 35 percent longer. A compact anchor costs 37,372 gas in steady state, and the largest proof measured is 704 bytes. All 21 scripted security cases passed. The figures come from a single-process simulation. A distributed deployment is left for later work.

**Keywords:** Electronic Health Record, Hybrid Blockchain, Consent-Bound Key Derivation, Adaptive Anchoring, Dual-Ledger Proof

## 1. Introduction

A patient's laboratory reports, images and discharge notes are created by different institutions, and no single institution holds the whole history. Sending a record from the hospital that stores it to the hospital that needs it has meant trusting the party in the middle, or trusting that a copy, once released, will still be governed by the consent under which it was released. A public ledger can make that history hard to alter quietly, but it is slow, costly, and careless with metadata. A private consortium ledger is fast and discreet, and it is only as honest as a quorum of its members.

The designs that already put permissions on a chain and records off it still leave three gaps [1]. Consent is checked as a yes-or-no gate, and the key that then opens the record does not depend on the consent version that was checked. Anchoring, when it is done at all, follows a clock or a fixed batch, so a revocation waits

as long as a routine log line. The patient, who is the person least able to read the consortium ledger, is also the person with no self-contained proof.

HyL-EHR is a hybrid-ledger framework for that transfer. Consent entries, access logs and record fingerprints stay on a permissioned ledger. Encrypted records stay off-chain. The private ledger is anchored to a public chain. Three mechanisms are added. Consent-Bound Key Derivation (CBKD) derives the session key of a transfer from an ephemeral Diffie-Hellman secret salted with the consent digest. Sensitivity-Aware Adaptive Anchoring (SAAA) anchors a critical event in the block that contains it and retunes the batching threshold so that the other traffic stays near a budget chosen in advance. A Dual-Ledger Proof (DLP) is a short Merkle path from a transaction to a public anchor.

The measurements cover the cost as well as the gain. The cost of binding the key is reported by record size and against simpler envelopes. The extra wait imposed on routine traffic is reported at an anchor spend close to a fixed interval. A separate check shows what a patient can verify from public-chain data alone, and what remains possible in the interval before a block is anchored. The answers below are from one virtual machine. They are evidence about the mechanisms, not a claim about a hospital network.

## 2. Related Work

MedRec placed permissions and pointers in Ethereum contracts and left the records in the databases of the providers [1]. That split, pointers on-chain and data off-chain, is the pattern later systems keep. The chain in that design is a proof-of-work network, relationship metadata is readable, and nothing protects the record while it is in transit. Ancile moved the same idea onto a permissioned Ethereum network, with six contracts and proxy re-encryption of record keys, and set the access rules against the requirements of the Health Insurance Portability and Accountability Act [2]. The proxies are a point where collusion is possible, and integrity still rests on the consortium. The work was not benchmarked.

FHIRChain stored encrypted pointers to Fast Healthcare Interoperability Resources on-chain and used public keys as identities [3]. Clinical data stayed off-chain, which is the right place for it, and the permission token was not tied to the consent that was current at the moment of use. A role-based Ethereum design stored the records themselves in the InterPlanetary File System (IPFS) [4, 5]. Every write then pays a public fee, the access pattern is public, and consent is a flag that is unrelated to the key. A later patient-controlled design added reputation-based re-encryption oracles in front of IPFS [6]. Permission changes on a public chain are paid for and visible, and the oracles have to be trusted.

A consortium scheme on Hyperledger Fabric combines IPFS, proxy re-encryption and attribute-based signatures, and reports a lower computing and communication cost than the schemes it compares [7]. The comparison is analytical, revocation depends on the proxy, and there is no public anchor. A self-sovereign design encrypts records with the Advanced Encryption Standard (AES), holds the keys under attribute-based encryption, and writes consents to a chain, including an audit of emergency access [8]. Consent is recorded beside the key. It is not an input to the key, and the consent log lives on one chain.

A hybrid construction routes records across three cooperating chains according to five sensitivity levels, with attribute-based encryption, aggregated signatures and IPFS [9]. Reported latency stays under 6 s at 50 transactions per second, and signature aggregation saves storage. The chains are not anchored to each other, and sensitivity decides where a record goes, not when a commitment is published. The closest storage architecture encrypts records with AES-256 in Galois/Counter Mode (GCM), keeps them in IPFS, and anchors Merkle roots of fixed batches on an Ethereum test network [10]. Batching cuts the gas of

anchoring sharply. Every record is treated alike. There is no consent, no revocation, and no provider-to-provider transfer.

Outside the health-record literature, a permissioned ledger can be observed by an outsider if a committee publishes state commitments on a public board, provided one committee member is honest [11]. The commitments in that work are periodic. Between two of them the observer learns nothing, and the method is not specific to health records. Taken together, the consent check in these systems releases a key that does not depend on the consent it was released under [1, 2, 4, 6, 7, 8]. The anchors that do exist fire on a batch or a clock [10, 11]. None of the health-record schemes gives the patient a proof that can be checked from public-chain data alone.

### 3. System Design

#### 3.1 Stores, Roles and Trust Assumptions

Four roles are used. The patient grants and revokes consent. The custodian holds a record and its data key. The requester asks for a copy. Validators order the permissioned ledger. Each participant has a long-term Ed25519 key. The identifier written on the ledger is a pseudonym derived from the public key. Linking that pseudonym to a person is done at enrolment, off the ledger.

Three stores are kept apart, and each holds the least it needs. The off-chain store holds ciphertext and is addressed by the content identifier. The private ledger holds pseudonyms, consent entries, access logs, and the fingerprint, size, type and sensitivity class of each record. The public ledger holds, for each epoch, two Merkle roots, the first and last private-block height, and a code for the event that caused the anchor. The data key stays in the custodian's key store. In the prototype that store is memory. A deployment would use a hardware module or a key service. The content identifier is not copied to the public chain. A fingerprint of a medical record can itself be personal data, and publishing it would let anyone who later obtains the ciphertext tie it to a ledger entry.

**Table 1: Placement of Data across the Three Stores**

| Item               | Off-chain store           | Private ledger | Public ledger          |
|--------------------|---------------------------|----------------|------------------------|
| Record contents    | Ciphertext only           | No             | No                     |
| Data key           | No                        | No             | No                     |
| Record fingerprint | Address of the ciphertext | Yes            | No                     |
| Identities         | No                        | Pseudonyms     | No                     |
| Consent entries    | No                        | Yes            | Merkle root only       |
| Access log         | No                        | Yes            | Inside the ledger root |
| Block hashes       | No                        | Yes            | Merkle root only       |

The cryptographic assumptions are the standard ones. SHA-256 (Secure Hash Algorithm, 256-bit) is treated as collision resistant, AES-256-GCM as a secure authenticated cipher, and the X25519 and Ed25519 problems as hard. Each party keeps its own private keys. A network attacker can read, drop, replay and alter messages. The store can read every ciphertext and can modify or reorder chunks. A requester who holds a registered identity can replay an old request, ask outside the consented scope, or ask after revocation. A partitioning attacker can delay one party's view of the private ledger. At most f of

$n$  validators may sign an invalid block, with  $n = 3f + 1$ , and a block is final only after  $2f + 1$  signatures. A quorum that holds  $2f + 1$  validator keys can re-sign a private block and try to erase a revocation. A custodian who leaks a record it is allowed to read is outside the model, as in any design that must eventually show a record to a clinician. What the mechanisms can do is refuse a transfer that has no live consent, and leave a public trace if a quorum later replaces an anchored block.

### 3.2 Records, Consent and the Private Ledger

A record is cut into chunks of 256 KiB. A fresh 256-bit data key is drawn for that version of that record, and each chunk is encrypted with AES-256-GCM. The chunk index is the nonce. The key is never reused, so the index is a safe nonce. The associated data of a chunk carries the record identifier, the chunk index and the chunk count. A chunk that is moved, reordered, truncated or attached to another record fails authentication. The Merkle root of the ciphertext chunks is the content identifier. The ciphertext is stored under that identifier, and a RECORD\_ADD transaction places the identifier, the size, the type, the sensitivity class and the patient's pseudonym on the private ledger. From that point the ledger, not the store, is the authority on what the ciphertext is.

The private ledger is an in-process model of a Byzantine fault tolerant (BFT) orderer [12]. It orders transactions and collects header signatures. It does not exchange consensus messages, and the present prototype does not run a distributed ledger. A transaction is accepted only when its signature matches the registered key and the type rules hold. Only a provider adds a record. Only the named patient grants or revokes consent. Only the custodian of a record writes its access log. A transaction identifier is accepted once. The transaction types are REGISTER, RECORD\_ADD, CONSENT\_GRANT, CONSENT\_REVOKE, ACCESS\_LOG and EMERGENCY\_ACCESS.

A consent entry is keyed by the pair of patient and grantee. It stores a version, a status, a scope, a purpose, an expiry time, and the identifier of the block in which it last changed. Every grant or revocation increases the version. The consent digest is the hash of those fields together with the hash of that block. Two parties obtain the same digest only when they have seen the same last change in the same block. Expiry does not need a transaction. The check compares the expiry with the current time, and an expired entry simply stops being live. Registration is routine. A grant is sensitive. Revocation and emergency access are critical. A new record and its access log inherit the sensitivity label of the record.

### 3.3 Consent-Bound Key Derivation

A transfer has the shape of an authenticated ephemeral Diffie-Hellman handshake of the sign-and-MAC kind used in TLS 1.3 [13]. Each side contributes a fresh X25519 key and a nonce, and signs its contribution with its long-term Ed25519 key [14, 15]. The handshake carries a wrapped data key. It does not carry the record.

The custodian, on a request, checks the requester's signature against the identity registry. It then checks freshness. It reads the latest public anchor and refuses to continue unless its own replica contains the blocks that anchor commits to. It then checks that a live consent from the patient to this requester covers the record type. Only after those checks does it read the consent digest from its own replica and derive the session keys with the HMAC-based Key Derivation Function (HKDF) [16].

$K_{enc} \parallel K_{conf} = \text{HKDF}(\text{X25519 secret}; \text{salt} = \text{cd}; \text{info} = \text{label} \parallel \text{aref}(e) \parallel \text{th})$  (1)

HKDF with SHA-256 returns 64 bytes, split into an encryption key  $K_{enc}$  and a confirmation key  $K_{conf}$ . The input keying material is the Diffie-Hellman secret, which supplies confidentiality and a fresh key for this session. The salt is the consent digest  $\text{cd}$ . The info field carries a fixed protocol label, a reference  $\text{aref}(e)$  to the latest public epoch  $e$ , and the transcript hash  $\text{th}$ . The transcript binds the record identifier,

the content identifier, both parties, both ephemeral public keys and both nonces. The data key is wrapped under Kenc with the transcript hash as associated data, using AES-256-GCM [17]. The custodian sends the ephemeral key, the nonce, the epoch, a signature, the wrapped key and a confirmation MAC under Kconf.

The requester does not take the consent digest from the message. It computes its own digest from its own replica and derives the keys with the same formula. The wrapped key opens only when the two digests match and both parties name the same anchor. A difference in any consent field, in the version, or in the block of the last change, produces an authentication failure rather than a wrong plaintext. The requester then fetches the ciphertext, checks that its Merkle root equals the content identifier on the ledger, and decrypts.

The requester signs a receipt that names the record, the content identifier, the consent digest and the transcript hash, and includes its own confirmation MAC. The custodian checks the receipt and submits an ACCESS\_LOG transaction containing the consent digest and the hash of the receipt. The ledger records not only that the requester received the record, but which version of the patient's consent was relied on. The requester cannot later deny that receipt.

The custodian wraps 32 bytes. Its public-key work does not grow with the record. Proxy re-encryption is used elsewhere for the same economy, and it introduces a proxy [18]. Here the wrap uses ordinary symmetric and elliptic-curve operations, and the ephemeral key is new on every transfer, so a later compromise of a long-term key does not open an earlier session. The requester does learn the data key of that one record version. The key protects only that version, so nothing beyond the entitled plaintext is revealed.

### 3.4 Sensitivity-Aware Adaptive Anchoring

After every private block, an anchoring service decides whether to publish a new anchor. Each class has a weight: 1 for a routine transaction, 4 for a sensitive transaction, and 16 for a critical transaction.  $S$  is the sum of the weights of transactions that have not yet been anchored. Three conditions publish an anchor. Publish an anchor if the block is critical, or  $S$  is at least  $\theta$ , or the oldest unanchored transaction has waited  $T_{\max}$ . (2)

A block that contains a revocation or an emergency access is anchored at once. Otherwise the anchor waits until the accumulated score reaches a threshold  $\theta$ , or until the oldest unanchored transaction has waited  $T_{\max}$  seconds.  $T_{\max}$  is a hard cap on the exposure of ordinary traffic. When a daily budget  $B$  is set,  $\theta$  is retuned every window  $W$ . The retune compares the anchors spent in the last window, and the anchors spent since the start, with the pro-rata share of  $B$ , and multiplies  $\theta$  by a damped ratio. Critical transactions are not delayed by that retune. In the runs reported here,  $T_{\max}$  is 1,800 s,  $W$  is 600 s, and the damping exponent is 0.5.

An anchor carries the Merkle root of the private block hashes in the epoch, the Merkle root of the current consent digests, the height range, and a trigger code. Leaves and inner nodes of both trees are hashed with different prefixes, and each tree is padded to a power of two with a fixed empty leaf, so a leaf cannot be presented as an inner node [19]. The contract AnchorRegistry, in Solidity 0.8, accepts an anchor only from the consortium account, rejects a gap in the height range, and keeps a hash chain over successive anchors. A compact call leaves the roots in the event log and stores only the hash chain. The public chain receives no patient identifier, no record fingerprint and no key.

### 3.5 Dual-Ledger Proof

A proof that a transaction belongs to the anchored history has two hops. The first is a Merkle path from

the transaction to the hash of its private block. The second is a Merkle path from that block hash to the ledger root stored for the epoch. The patient, or anyone else, checks the second hop against the public anchor. The contract exposes the same check as a view function, so the second hop can also be evaluated on-chain. A quorum that rewrites an anchored block can still produce a full set of validator signatures on the replacement. The public proof of the original block then fails. The same rewrite, performed on a block that has not yet been anchored, leaves no public evidence. The length of that interval is the exposure window measured in Section 5.

#### 4. Prototype and Experimental Setup

The prototype is a Python 3.13 package. Cryptographic operations go through the cryptography library, version 50.0, and from there to OpenSSL. The primitives are X25519, Ed25519, HKDF-SHA-256, AES-256-GCM and SHA-256. RSA-2048 with OAEP and PSS padding is used only as a baseline. No primitive was implemented by hand. The anchor contract was compiled with solc 0.8.26 and executed by a local Ethereum virtual machine, version 10.1 of @ethereumjs/vm, under Cancun gas rules. No ether was spent, and no public test network was contacted.

Every timing was taken on one cloud virtual machine: an Intel Xeon with two virtual CPUs at 2.10 GHz and AES-NI, 7.8 GB of memory, and Ubuntu Linux 24.04. The network interface was not used. All parties run in one process, so no figure includes message delay or disk delay. Primitive times are the best of five runs. Transfer times are medians of 15 runs. Ledger times are medians of five runs. Anchoring figures are means over 10 random seeds. Proof times are medians of 1,000 runs. Gas figures are exact.

Six record sizes were used, from 4 KiB to 64 MiB. The payloads are random bytes inside a FHIR-style envelope. No patient record was used. The anchoring workload is synthetic, because no public trace of consortium health-record traffic was available. Arrivals are Poisson, with a mean of 2 transactions per second on the default run, of which 0.1 percent are critical and 5 percent are sensitive. A private block is formed every 2 s. The public chain is reduced to a slot of 12 s that includes every submitted anchor. Three simulated days are generated and the last two are scored. The critical share and the offered load are swept, because the class mix is an assumption.

Four transfer schemes share the library and the records. CBKD is the proposed handshake. A plain signed ephemeral exchange wraps the data key with no consent binding, no ledger check and no receipt. An RSA-2048 envelope wraps the data key to the requester's static RSA key and returns a signed receipt. A re-encryption baseline decrypts the stored record and encrypts it again under the session key. For anchoring, the clock-based policy publishes one anchor every 150 private blocks, which is every 300 s and 288 anchors a day on this block interval. Anchoring every non-empty block is the upper bound on public cost. A later deployment would replace the simulator with Hyperledger Fabric, the store with IPFS, and the local virtual machine with a public test network [5, 20]. None of those components produced the numbers below.

## 5. Results

### 5.1 Cryptography and Transfer Time

An Ed25519 signature takes 38.6 microseconds and a verification takes 121.1 microseconds. An X25519 shared secret takes 40.7 microseconds. HKDF-SHA-256 takes 4.2 microseconds, and a five-field SHA-256 takes 1.7 microseconds. The extra hash that feeds the consent digest into HKDF is small beside the public-key operations around it. RSA-2048 signing takes 395.9 microseconds and OAEP decryption takes

386.4 microseconds, close to ten times an elliptic-curve operation. On a 32 MiB buffer, AES-256-GCM encrypts at 1,762 MiB/s and SHA-256 runs at 1,278 MiB/s. Preparing a record in 256 KiB chunks and computing its content identifier in the same pass reaches 624 MiB/s. That cost is paid once per record version, not once per transfer.

**Table 2: CBKD Transfer Time by Stage, in Milliseconds**

| Record size | Request | Response | Root check | Decrypt | Receipt | Total   |
|-------------|---------|----------|------------|---------|---------|---------|
| 4 KiB       | 0.096   | 0.319    | 0.008      | 0.303   | 0.375   | 1.118   |
| 64 KiB      | 0.102   | 0.327    | 0.056      | 0.293   | 0.351   | 1.152   |
| 1 MiB       | 0.125   | 0.352    | 0.855      | 0.809   | 0.360   | 2.533   |
| 8 MiB       | 0.150   | 0.369    | 6.825      | 4.812   | 0.390   | 12.527  |
| 32 MiB      | 0.223   | 0.420    | 27.812     | 37.993  | 0.409   | 66.845  |
| 64 MiB      | 0.239   | 0.447    | 55.342     | 83.045  | 0.423   | 139.801 |

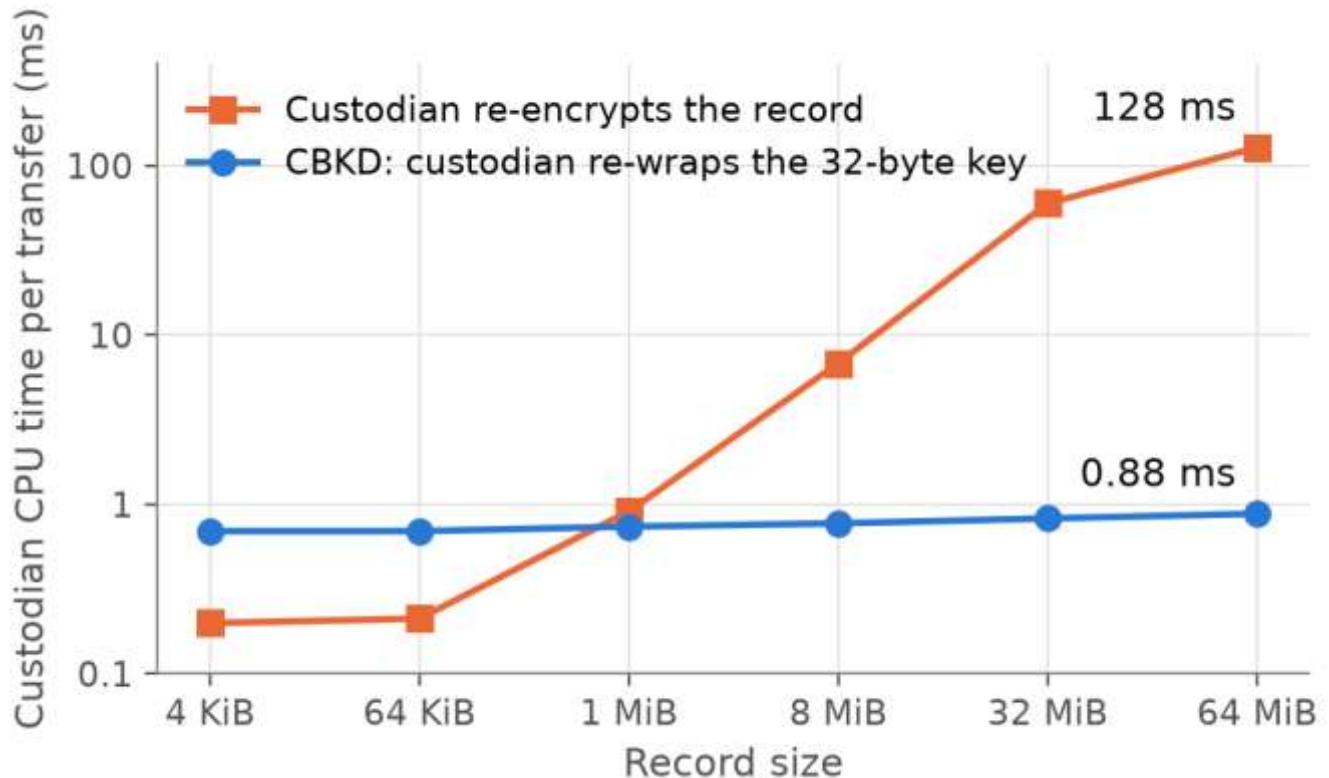
Up to 64 KiB, the transfer from the request through the access-log entry takes 1.118 ms to 1.152 ms, and most of that time is the handshake and the receipt. From 1 MiB upward the content-identifier check and the decryption grow with the record, while the handshake and the receipt stay near 1 ms. A 64 MiB record is delivered, checked and decrypted in 139.801 ms, which is 457.8 MiB/s on this machine. The protocol overhead is then a small fraction of the transfer.

**Table 3: Total Transfer Time and Custodian Time by Scheme, in Milliseconds**

| Record size | CBKD    | ECDH only | RSA envelope | Re-encrypt | Custodian, CBKD | Custodian, re-encrypt |
|-------------|---------|-----------|--------------|------------|-----------------|-----------------------|
| 4 KiB       | 1.118   | 0.577     | 1.717        | 0.563      | 0.695           | 0.199                 |
| 64 KiB      | 1.152   | 0.612     | 1.821        | 0.583      | 0.695           | 0.212                 |
| 1 MiB       | 2.533   | 1.692     | 2.930        | 1.784      | 0.739           | 0.909                 |
| 8 MiB       | 12.527  | 9.500     | 10.877       | 11.959     | 0.773           | 6.802                 |
| 32 MiB      | 66.845  | 66.678    | 68.345       | 83.086     | 0.826           | 59.873                |
| 64 MiB      | 139.801 | 141.661   | 142.452      | 175.186    | 0.880           | 127.958               |

At 4 KiB, where the bulk cipher is negligible, CBKD takes 1.118 ms and the plain exchange takes 0.577 ms. Key establishment alone, without the receipt and the ledger entry, takes 0.873 ms for CBKD and 0.578 ms for the plain exchange. The RSA envelope takes 1.717 ms at the same size and does not bind the key to a consent digest. Its key-establishment time is 1.880 ms. The custodian's CBKD time moves only from 0.695 ms at 4 KiB to 0.880 ms at 64 MiB. The same custodian, when it re-encrypts the record, takes 0.199 ms at 4 KiB and 127.958 ms at 64 MiB.

**Figure 1: Custodian Time per Transfer against Record Size**



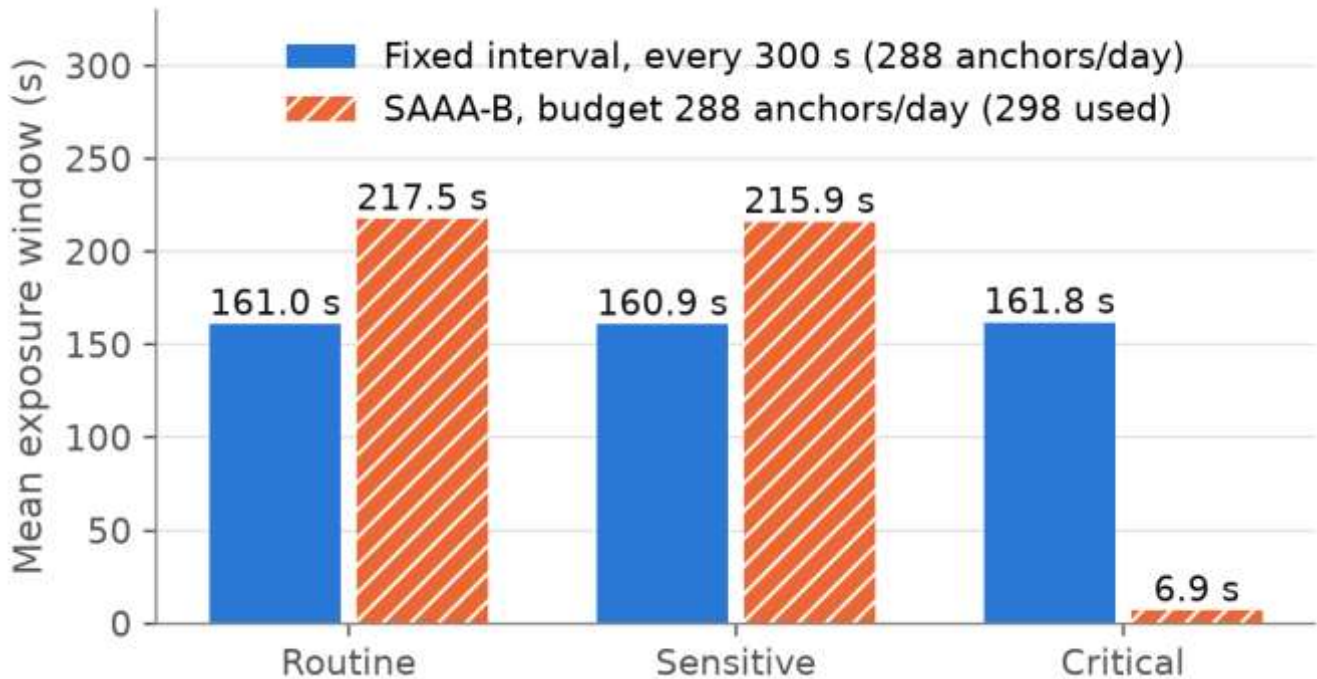
## 5.2 Private Ledger

Admission of one transaction stays near 0.14 ms across the whole grid, from 4 to 16 validators and from 100 to 2,000 transactions in a block. With 4 validators, a block of 100 transactions commits in 43.1 ms. Sequential throughput with 4 validators stays between 1,737 and 1,779 transactions per second as the block grows from 100 to 2,000 transactions. With 16 validators the same range falls to between 565 and 586 transactions per second, because every signature is verified in the one process. Dividing the block work by the quorum, as if each validator had its own processor, leaves a per-node figure between 3,286 and 3,599 transactions per second. These are ordering and signature costs. They do not include a network or a real consensus round.

## 5.3 Anchoring Policy

On the default workload a fixed interval of 150 blocks uses 288 anchors a day and 10.76 million gas a day, at the measured compact-anchor cost of 37,372 gas. The mean exposure is 161.0 s for routine traffic, 160.9 s for sensitive traffic and 161.8 s for critical traffic. The 95th percentile of a critical event is 296.2 s and the maximum is 310.0 s. SAAA with the budget set to 288 anchors a day used 298 anchors and 11.14 million gas. Critical events then wait 6.9 s on average, 12.0 s at the 95th percentile and 12.0 s in the worst case, which is one simulated public slot. Routine traffic waits 217.5 s, and sensitive traffic waits 215.9 s. The routine window is 35 percent longer than 161.0 s. The critical window is about 23 times shorter than 161.8 s.

**Figure 2: Mean Exposure Window at a Budget of 288 Anchors a Day**



The budget is held only while critical events are rare. At a critical share of 0.01 percent, the budgeted policy uses 291 anchors a day and the routine window is 159.8 s, close to the fixed interval, while the critical mean is still 6.9 s. At 0.20 percent the immediate anchors push the spend to 344 a day. At 0.50 percent the spend is 850, and at 1 percent it is 1,665. The critical mean stays near 7 s at every share in the sweep. Past the point where critical events alone would fill the budget, the policy spends more than it was given, because a revocation is not deferred.

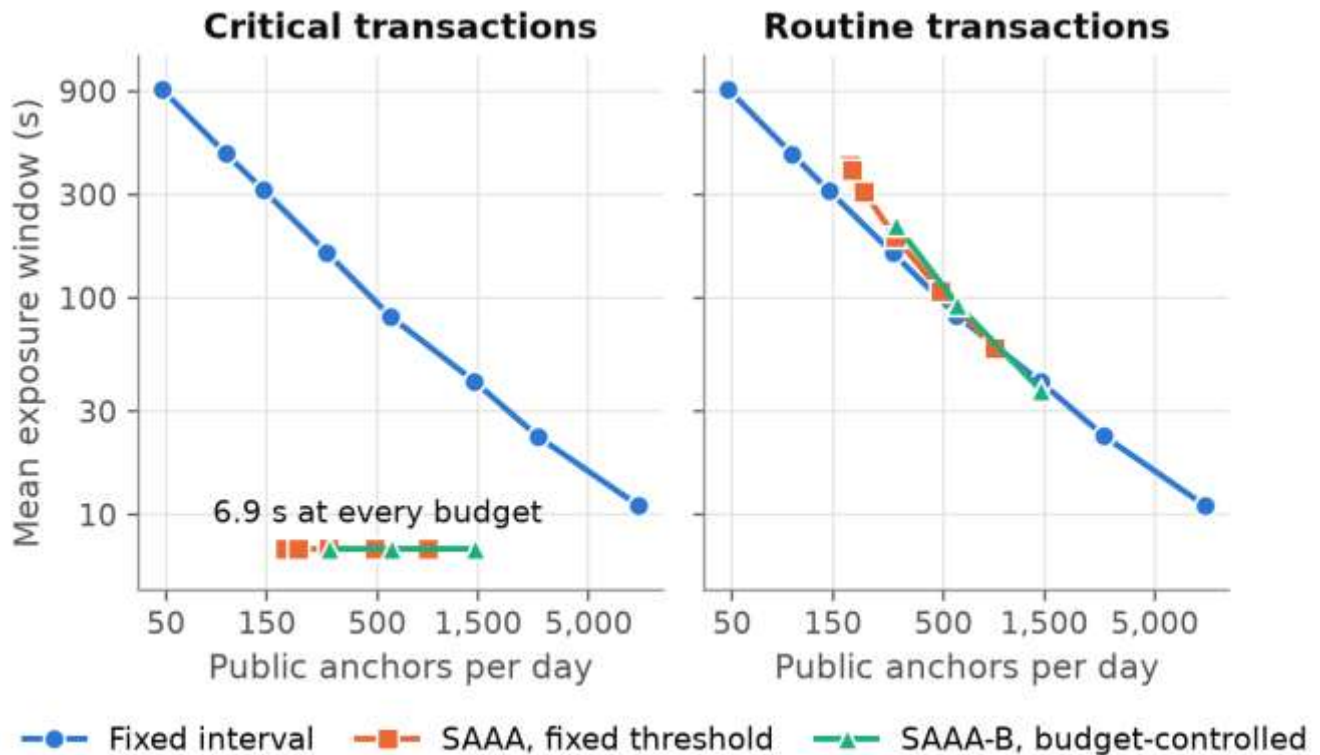
**Table 4: Anchor Spend and Exposure as the Critical Share Changes, Budget 288 a Day**

| Critical share (percent) | Anchors per day | Routine window (s) | Critical mean (s) |
|--------------------------|-----------------|--------------------|-------------------|
| 0.01                     | 291             | 159.8              | 6.9               |
| 0.05                     | 293             | 178.0              | 7.0               |
| 0.10                     | 298             | 217.5              | 6.9               |
| 0.20                     | 344             | 242.8              | 7.0               |
| 0.50                     | 850             | 106.0              | 7.0               |
| 1.00                     | 1,665           | 56.5               | 7.0               |

When the offered load moves from 0.5 to 6 transactions per second and the budget is 576 anchors a day, the budgeted policy stays between 573 and 587 anchors a day. A fixed threshold, tuned at 2 transactions per second and not retuned, rises from 147 anchors a day at the light load to 1,710 at 6 transactions per second. Anchoring every block on the default workload uses 39,809 anchors a day and 1,487.74 million gas a day, and brings every class to about 7 s. That is the cost of refusing to batch. Figure 3 plots the same

comparison across anchor rates. The critical-event curve for both forms of SAAA sits near 6.9 s at every budget, while the fixed-interval curve falls only as the anchor rate rises.

**Figure 3: Mean Exposure Window against Anchors per Day**



## 5.4 Proofs, Gas and Scripted Cases

Proof size follows the depth of the two trees. A small epoch is a few hundred bytes. At 128 blocks in an epoch and 1,024 transactions in a block, the proof is 17 hashes and 704 bytes. It is built in 1.61 ms and checked in 14.0 microseconds. That 704-byte proof is the largest in the measured grid. A private-ledger transaction stays almost constant as the record grows: 572 bytes for a 2 KiB record and 584 bytes for a record of about 48 MiB. The calldata of one public anchor is 81 bytes, two 32-byte roots, two heights and a trigger.

**Table 5: Dual-Ledger Proof Size and Local Verification Time**

| Blocks per epoch | Transactions per block | Hashes | Proof (bytes) | Check (microseconds) |
|------------------|------------------------|--------|---------------|----------------------|
| 1                | 16                     | 4      | 288           | 7.0                  |
| 16               | 256                    | 12     | 544           | 11.3                 |
| 128              | 1,024                  | 17     | 704           | 14.0                 |
| 1,024            | 64                     | 16     | 672           | 13.5                 |

Deployment of the anchor contract costs 616,805 gas. The call that stores the anchor in contract state costs 118,298 gas on first use and 104,293 gas once storage has been initialised. The compact call costs 71,277

gas on first use and 37,372 gas in steady state. On-chain checking of a block path costs 25,593 gas for an epoch of one block and 38,611 gas for an epoch of 1,024 blocks. A forged path is rejected. An anchor that leaves a gap in the block heights is rejected. Python and the contract agree on the roots of the four epochs that were cross-checked.

All 21 scripted cases passed. Eighteen are attacks that the prototype detected or blocked. A request with no consent, a request after revocation, a request outside the consented scope, and a request after expiry are refused before any key is derived. A custodian whose replica has missed an anchored revocation is refused by the freshness check. When that check is switched off, the consent-bound unwrap still fails, because the two parties salt the key with different digests. A swapped ephemeral key, an unregistered signer, a wrapped key replayed into a new session, a flipped ciphertext bit, a reordered chunk, a forged transaction signature, a replayed transaction, a provider granting consent on the patient's behalf, a proof of a transaction that was never committed, and an emergency access by a clinician without that role are all rejected.

Two further cases describe the boundary rather than a blocked attack. An honest transfer under a live consent completes, and the plaintext matches. A patient can show a revoked consent entry against the anchored consent root, and a superseded entry is rejected. A colluding quorum can produce valid signatures on a replacement for an anchored block, and the public proof of the original block is then rejected. The same replacement of a block that is not yet anchored leaves no public evidence. That last case is the negative control. It is the exposure window, measured rather than assumed away.

## 6. Discussion

Binding the session key to the consent digest does not need a new primitive. The key-establishment times, 0.873 ms against 0.578 ms, differ by the consent hash and the HKDF call on top of the same X25519 exchange. The wider gap between a full CBKD transfer at 1.118 ms and a plain exchange at 0.577 ms also contains the receipt and the ledger entry, which the plain exchange does not write. The custodian's time stays under 1 ms up to 64 MiB because the stored record is not opened again. Table 3 isolates that choice from re-encryption of the ciphertext, which reaches 127.958 ms at the same size.

The anchoring gain is real on this workload, and it is not free or uniform. At the default mix, critical events wait about 23 times less than under the fixed interval that spends 288 anchors a day, and the budgeted policy spends 298. Routine events wait 35 percent longer. The penalty almost disappears at a critical share of 0.01 percent, and the budget is exceeded from 0.20 percent upward. A deployment in which emergency access is a large fraction of traffic would have to raise the budget or accept the extra anchors. The 12 s floor on the critical window is the public slot used in the simulation. It is not a measurement of a live chain.

The proof is small enough to hand to a patient, and the on-chain check is a few tens of thousands of gas. The public chain does not by itself list the patient's transactions. It makes a supplied path binding. The patient still has to obtain the path, from the custodian or from a replica, at the moment the proof is needed. Several limits follow from the method. Message delay, disk delay and a real BFT round are absent, and on a network those terms will dominate the millisecond figures of Section 5.1. The anchoring study uses an assumed arrival process and a public chain that includes every anchor in a fixed 12 s slot. Fee markets, congestion and reorganisations were not simulated. The security suite is a list of scripted cases, not an argument that every attack outside the list fails. Random payloads are not clinical records. Replacing the simulator with a distributed ledger, the local virtual machine with a public test network, and the random

bytes with synthetic patient histories is future work. Until that is done, the numbers show that the three mechanisms work and what they cost on one machine.

## 7. Conclusion

HyL-EHR keeps encrypted health records off-chain, consent and audit state on a permissioned ledger, and only Merkle roots on a public chain. The session key of a transfer is derived from the consent digest each party reads locally and from the latest public anchor, so a stale or revoked consent does not open the record. Critical events are anchored in the block that contains them, and the remaining traffic is held to a daily budget. On the reported workload, a budget of 288 anchors a day, of which 298 were used, cut the mean public-exposure window of a critical event from 161.8 s to 6.9 s and lengthened the routine window by 35 percent. A proof of at most 704 bytes can be checked against the public chain, and one compact anchor costs 37,372 gas. The custodian's cost stayed below 1 ms from 4 KiB to 64 MiB. The prototype is a single-machine simulation. The same protocols on a distributed ledger and a public test network are the next measurement.

## Acknowledgement

The facilities of the Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani Campus, Chennai, are acknowledged.

## References

1. A. Azaria, A. Ekblaw, T. Vieira, A. Lippman, "MedRec: Using Blockchain for Medical Data Access and Permission Management", Proceedings of the 2nd International Conference on Open and Big Data, August 2016, 25–30. <https://doi.org/10.1109/OBD.2016.11>
2. G.G. Dagher, J. Mohler, M. Milojkovic, P.B. Marella, "Ancile: Privacy-Preserving Framework for Access Control and Interoperability of Electronic Health Records Using Blockchain Technology", Sustainable Cities and Society, May 2018, 39, 283–297.
3. P. Zhang, J. White, D.C. Schmidt, G. Lenz, S.T. Rosenbloom, "FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data", Computational and Structural Biotechnology Journal, 2018, 16, 267–278.
4. A. Shahnaz, U. Qamar, A. Khalid, "Using Blockchain for Electronic Health Records", IEEE Access, 2019, 7, 147782–147795.
5. J. Benet, "IPFS: Content Addressed, Versioned, P2P File System", arXiv, July 2014. <https://arxiv.org/abs/1407.3561>
6. M.M. Madine, A.A. Battah, I. Yaqoob, K. Salah, R. Jayaraman, Y. Al-Hammadi, S. Pesic, S. Ellahham, "Blockchain for Giving Patients Control over Their Medical Records", IEEE Access, 2020, 8, 193102–193115.
7. G. Liu, H. Xie, W. Wang, H. Huang, "A Secure and Efficient Electronic Medical Record Data Sharing Scheme Based on Blockchain and Proxy Re-Encryption", Journal of Cloud Computing, February 2024, 13 (1). <https://doi.org/10.1186/s13677-024-00608-w>
8. M. Tcholakian, K. Gorna, M. Laurent, H.K. Ben Ayed, M. Naghmouchi, "Self-Sovereign Identity for Consented and Content-Based Access to Medical Records Using Blockchain", arXiv, July 2024. <https://arxiv.org/abs/2407.21559>

9. G. Han, Y. Ma, Z. Zhang, Y. Wang, “A Hybrid Blockchain-Based Solution for Secure Sharing of Electronic Medical Record Data”, *PeerJ Computer Science*, January 2025, 11, e2653.
10. S. Sharmin, M.S. Arefin, P.K. Dhar, Z. Sultana, S. Akter, “A Scalable and Privacy-Preserving Hybrid Blockchain Architecture for Secure Healthcare Data Management”, *International Journal of Advanced Computer Science and Applications*, 2025, 16 (8), 988–997.
11. E. Abebe, Y. Hu, A. Irvin, D. Karunamoorthy, V. Pandit, V. Ramakrishna, J. Yu, “Verifiable Observation of Permissioned Ledgers”, *arXiv*, December 2020. <https://arxiv.org/abs/2012.07339>
12. M. Castro, B. Liskov, “Practical Byzantine Fault Tolerance”, *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, February 1999, 173–186.
13. E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, RFC 8446, August 2018. <https://www.rfc-editor.org/rfc/rfc8446>
14. D.J. Bernstein, “Curve25519: New Diffie-Hellman Speed Records”, *Public Key Cryptography, PKC 2006*, Lecture Notes in Computer Science, Volume 3958, Springer, 2006, 207–228.
15. D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, B.-Y. Yang, “High-Speed High-Security Signatures”, *Journal of Cryptographic Engineering*, September 2012, 2 (2), 77–89.
16. H. Krawczyk, “Cryptographic Extraction and Key Derivation: The HKDF Scheme”, *Advances in Cryptology, CRYPTO 2010*, Lecture Notes in Computer Science, Volume 6223, Springer, 2010, 631–648.
17. M. Dworkin, “Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC”, NIST Special Publication 800-38D, November 2007.
18. G. Ateniese, K. Fu, M. Green, S. Hohenberger, “Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage”, *ACM Transactions on Information and System Security*, February 2006, 9 (1), 1–30.
19. R.C. Merkle, “A Digital Signature Based on a Conventional Encryption Function”, *Advances in Cryptology, CRYPTO 87*, Lecture Notes in Computer Science, Volume 293, Springer, 1988, 369–378.
20. E. Androulaki, et al., “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”, *Proceedings of the 13th EuroSys Conference*, April 2018, article 30. <https://doi.org/10.1145/3190508.3190538>