A Waterlog for Detecting and Tracing Synthetic Text from Large Language Models

Brennon Brimhall*, Orion Weller*, Matthew Green*, and Ian Miers**

*Johns Hopkins University
{brimhall, oweller, mgreen}@cs.jhu.edu

**University of Maryland
imiers@umd.edu

Abstract

We propose waterlogs, a new direction to detect and trace synthetic text outputs from large language models based on transparency logs. Waterlogs offer major categorical advantages over watermarking: it (1) allows for the inclusion of arbitrary metadata to facilitate tracing, (2) is publicly verifiable by third parties, and (3) operates in a distributed manner while remaining robust and efficient.

Waterlogs rely on a verifiable Hamming distance index, a novel data structure that we describe, to efficiently search multi-dimensional semantic hashes of natural language embeddings in a verifiable manner. This data structure may be of independent interest.

We implement a waterlog, which we call DREDGE, and benchmark it using synthetic text generated by GPT-2 1.5B and OPT-13B; embeddings are generated via OpenAI's text-embedding-ada-002 model [23]. We provide empirical benchmarks on the efficiency of appending text to the log and querying it for matches. We compare our results to watermarking and outline areas for further research.

1 Introduction

Large language models (LLMs) are popular neural networks that generate synthetic English natural language text. A number of commercial LLM offerings are available to users over the internet in a chat-based interface [48, 52, 5, 3, 15, 4]. LLMs are trained to minimize information-theoretic differences between their output and the distribution of English natural language. This is the primary reason why distinguishing their outputs from humans is challenging; sufficiently powerful LLMs do not leave signals to detect by design.

The availability of commercial LLMs has raised concerns around plagiarism, impersonation, and and dissemination of misinformation [45, 47, 30]. In particular, LLMs have been used to seed false information on social media networks and to impersonate individuals by formulating offensive statements in their voice [22]. The challenge in these cases is that it is often difficult to prove that a given piece of text is the output of an LLM. Furthermore, the widespread availability of undetectable LLM output creates other problems related to copyright [27, 11, 19, 10], and can inadvertently bias the data available to train future LLMs [1].

Watermarking. The difficulty of recognizing LLM-generated output has motivated a number of solutions. A popular proposal is to watermark LLM outputs [1, 32, 26, 33, 25]: this involves inserting a steganographic signal into model outputs that uniquely identifies their source. Watermarking has been enthusiastically endorsed by policymakers: for example, it is explicitly mentioned as a recommended practice in US Executive Order 14110 [7, 8].

The most straightforward approach to watermarking is known as *symmetric watermarking.* In this setting, the model operator is responsible for both watermarking and verifying the watermark, typically

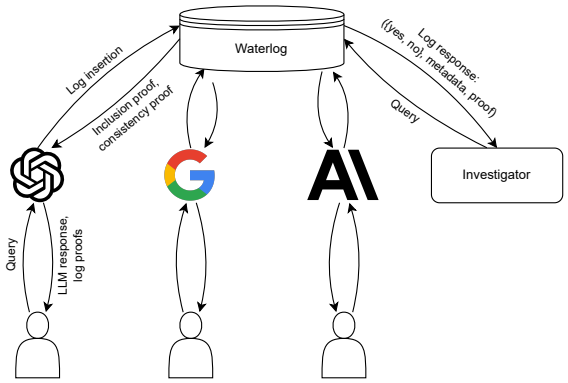


Figure 1: Diagram of waterlog workflow.

using a secret key held by the operator. These approaches exploit the fact that modern attention-based LLMs output a weighted vector of candidate tokens at each step; the generator then samples one token at random. Most symmetric schemes operate by subtly altering this distribution in a detectable way. For example, Kirchenbauer et al. [32] propose dividing the output tokens produced by an LLM at each token output cycle into a "green" and "red" list generated by a secret key. The LLM prioritizes sampling from the green list and de-prioritizes sampling from the red list. Watermark detection works by tokenizing the text, counting the number of "green" and "red" tokens, and computing a statistical test. If the test indicates a statistically significant result, then the given text is considered to embed a watermark. Later work has extended this approach in various ways.[1]

A related technique known as *asymmetric watermarking* allows for public verification of watermarks, but the existing schemes are extremely inefficient in practice (see §8.2).

A critical feature of watermarking schemes is that they should be *robust* against changes to the text, including word modifications, rephrasing and truncation of textual examples. A number of works have considered the robustness of symmetric watermarking in the presence of humans who attempt to paraphrase and edit the text [26, 33, 25]; these works find that given model outputs of sufficient length, many watermarking approaches are robust against several common modification strategies.

Properties of an ideal tracing scheme. We note that an ideal synthetic text detection scheme would have the following properties, which we formalize in §3:

Metadata recovery Any detection system should include the ability to derive actionable metadata from the text, such as the identity of the creator and/or the prompt (or at minimum, a reference that can be used to recover this data in some other database).

Non-repudiation Third parties must be persuaded by a positive or negative verification of a symmetric watermark. This guarantee should hold in the event that a malicious LLM provider (verifier) chooses to claim or deny that a given piece of text is watermarked.

Unframeability To reduce the possibility of framing an innocent user, it should be difficult for an adversary to create non-model outputs that will be recognized as watermarked.

Decentralization While current systems rely on centralized, trusted providers to both inject and verify watermarks, it is useful for providers to be able to outsource one or both tasks. This allows many providers to share their nfrastructure for watermark detection.

Quality	DREDGE/waterlogs (this work)	Symmetric watermarks [32]	Asymmetric watermarks [21]
Metadata	✓	✗	✗
Non-repudiation	✓	✗	✓
Unframeability	✓	✗	✗
Federation	✓	✗	✓
Robust	✓	✓	✗
Efficient	✓	✓	✗

Table 1: Summary of qualitative differences between waterlogs and prior watermarking approaches.

A concern with existing watermarking schemes is that they lack many of these features. First, verifying symmetric watermarks depends on a keyed verification process which requires interested parties to trust a service's assertion that a piece of text is or is not watermarked. For high-stakes matters, this assertion may not be sufficient to convince third parties that the identification is valid. Moreover, most existing watermarks encode only a single "bit" of information: namely the fact that the text is marked or not. This does not immediately facilitate any further action on marked text, such as identifying the creator or prompt that led to the textual output. This can be addressed if the LLM generator produces a new secret key for every user, or encodes additional data into the watermark, but this greatly increases verification and encoding time and may negatively affect robustness.[2] Even in this setting, practical systems willlikely require the watermark to be combined with a separate database containing model outputs and metadata. The need to maintain a database raises the question of whether the watermark is needed in the first place.

Existing symmetric watermark schemes are also *equivocal*, which means that a malicious verifier can incorrectly claim or disclaim a watermark. For example, watermarking approaches are subject to false denials. Suppose an LLM generates harmful speech. Even if the LLM provider detects their watermark, they may not wish to claim provenance. They may also admit framing via false accusations: LLM providers can claim a watermark exists when it does not to falsely accuse that the output was generated by an LLM. Prior work [31] has identified watermark stealing attacks that allow third parties who do not have the secret watermarking key to create text that would be identified as synthetic. For example, the Kirchenbauer et al. scheme relies on pairs of tokens. The first token, combined with the secret key, is what generates the "green" and "red" token lists for the second token. An adversary that observes a sufficient number of text outputs can incorporate these token pairs into its own text to claim that the text is synthetic. For example, a speechwriter for a public figure could create a speech that would verify as watermarked.[3]

Several of these issues affect the robustness properties of the symmetric watermark. Suppose we attempted to thwart watermark stealing attacks by increasing the window of tokens hashed to compute the perturbations. This change makes the symmetric watermark less robust to adversarial edits. Any edit by an adversary in the window would cause the following token to not be verified as watermarked. We describe that this attack also affects asymmetric watermarks in §8.2. We conjecture that this implies an inherent robustness-equivocation tradeoff for any watermarking scheme.

Finally, it not obvious how to federate symmetric watermarks. Existing techniques can only be verified in a centralized setting. Disclosing the secret key to a third party — such as if an LLM provider wished to (1) delegate watermark verification to a third party or (2) provide model weights to allow a user to generate watermarked synthetic text on device — allows that third party to defeat the watermark.[4]

[1]For example, [56, 16, 9, 42, 28, 39] reduce distortions and biases in watermarked text, [12, 49, 58] provide improved robustness guarantees, and [49, 54, 9] encode multiple watermark bits into the LLM output.

[2]Some watermarking schemes use hashing techniques to encode additional data into the watermark [49, 54, 9], in some cases using error correcting codes to support robustness. However, these approaches dramatically increase the cost of the watermark verification and generation and can substantially reduce the robustness of the watermark.

[3]One asymmetric watermark construction in literature [40] claims unforgability empirically but does not prove their puported unforgability in a formal cryptographic sense. It is possible that this construction is also subject to equivocation.

[4]Some work [29], uses trusted hardware and secure multiparty computation techniques to overcome this limitation for

Waterlog (this work)	Symmetric watermarks [32]	Asymmetric watermarks [21]
• Includes arbitrary metadata	• No metadata	• No metadata
• No equivocation possible	• Equivocation possible	• No equivocation possible
• Unframeable	• Framing possible	• Framing possible
• Publicly verifiable	• Only verifiable by LLM provider	• Publicly verifiable
• Operates on semantic space	• Operates on token space	• Operates on token space
• LLM output unchanged	• Alters LLM output distribution	• Alters LLM output distribution
• Federated	• Centralized	• Decentralized
• Provably robust	• Heuristically robust	• Not robust
• $O(n)$ space server-side	• $O(n)$ space client-side	• $O(n)$ space client-side

Table 2: Detailed comparison between waterlogs and prior watermarking approaches.

Many of the limitations of watermarking schemes are caused by the fact that watermarks attempt to *outsource* the storage of provenance information to end-users by encoding it into model outputs. This creates a robustness challenge due to the fact that watermarked text is held by possibly-adversarial users. Moreover, any investigatory process beyond simple verification still relies on some additional local storage, such as an operator database that logs model outputs, prompts, and user information. The need to operate such databases raises a simple question: why not simply replace symmetric watermarks with a verifiable database?

1.1 Waterlogs: Technical Intuition

In this work we propose an alternative approach that offers providers the means to verify the provenance of an LLM output, while also addressing many of the concerns listed above. Our key observation is as follows: if an LLM provider already logs information about model outputs (and will use that information in an investigative process), then *why not use this mechanism in place of the watermark?* In order to satisfy our desired security properties, this requires that we construct a searchable log that is highly efficient, robust to adversarial modifications, and *verifiable* to third parties. We refer to the resulting system as a *waterlog* which we formalize in §3.

Our construction begins with the concept of a transparency log [37, 43, 41]. These logs, most famously exemplified by Certificate Transparency [36], allow publishers to publicly commit to a large number of entries, and then later prove to third parties that the values exist within the log. Transparency logs have been deployed for a variety of purposes, including verification of key material for end-to-end encrypted messaging services [43, 41]. In a waterlog, we propose that centralized model operators will commit a searchable digest of model outputs and metadata (information such as the prompt and requesting user ID) to a special form of confidential transparency log: this log provides a public commitment to all model outputs, but allows providers to maintain the privacy of the committed data. When providers are asked to determine whether their system generated a given model output, the provider must first (1) look up the output and its metadata in their database; and critically they can (2) provide a (possibly zero-knowledge) proof of its inclusion or non-inclusion within the log.

The key technical challenge in constructing a waterlog is the need for an efficient and *robust* search protocol. This is critically important, because both the search and verification process require a means to efficiently locate matches and/or rule out non-matches at scale. We note that prior art, such as transparency logs, only support locating exact matches.

Our approach combines several existing tools. First, we make use of similarity-preserving hashes (e.g., SimHash [13]) in order to reduce complex textual model outputs into a relatively short digest; the key feature of this construction is that closely-related textual outputs will be "close", i.e., two likely matches will have zero or small Hamming distance. Next, we devise a novel protocol that allows an operator to perform

symmetric watermarks.

a structured search for closely-matching digests. The main feature of this protocol is that it is efficient enough that providers can efficiently provide an inclusion or non-inclusion proof demonstrating the presence or absence of a given match, even when the database becomes large.

By combining the cryptographic transparency log with an efficient search technique, we can now devise an efficient proof technique that allows operators to convince third parties that they have searched all model outputs and the identifier results (especially lack of results) represents a truthful response to any search query.

A final advantage of the centralized logging approach versus simple watermarks is that we are no longer restricted to specific operations conducted over tokens. This means that rather than evaluating similar matches by evaluating he distribution of tokens, we can instead identify likely candidates using semantic embeddings, a promising technique that has been explored in previous work [35].

2 Preliminaries

Before detailing our construction, we briefly review several cryptographic primitives that we use in this work. In addition, we define our main ingredients of transparency logs and semantic hashing. We also introduce verifiable Hamming distance indexes.

2.1 Cryptographic building blocks

Our constructions make use of several standard cryptographic primitives, including (1) a collision-resistant hash function $\mathcal{H}$, (2) a secure commitment scheme for multi-bit messages with output range $\{0,1\}^\lambda$, and (3) Merkle trees.

Commitment schemes. Let $c, \mathsf{opening} \leftarrow \mathsf{Commit}(m)$ represent the randomized commitment of message m under random coins r. A commitment can be opened by revealing $m, \mathsf{opening}$ and re-computing the commitment.[5] In our implementation we make use of hash-based commitments, where the hash function is modeled as a random oracle.

Merkle trees. A Merkle tree is a cryptographic data structure that produces a single digest over a sequence of values. The values are hashed to produce "leaves" of a binary tree. The internal nodes of the tree are created by recursively applying a collision-resistant hash function over the values of its children until it creates a single root hash.

The construction permits a succinct proof-of-membership that shows a particular value exists at a particular location. The proof consists of a logarithmic sequence of hashes from the value to the root hash. A verifier checks that hashing the provided sequence of hashes results in the root hash.

2.2 Transparency Logs

Transparency logs were introduced in 2013 in response to unauthorized certificates being issued by compromised certificate authorities [37, 36]. Under certificate transparency, certificate authorities submit all certificates they issue to a third-party *transparency log*. The log is constructed using a dense Merkle tree structure that provides an append-only invariant that can be verified via cryptographic inclusion and consistency proofs:

- An *inclusion proof* is a sequence of cryptographic hashes along the path from the tree leaf to the tree root. The verifier hashes the leaf and the hashes given in the proof and checks that the final value given matches the root of the tree.

[5] In practice, the commitment scheme may also have global parameters such as a CRS. We omit this in our presentation, but it should be assumed.

- A *consistency proof* verifies the append-only property by demonstrating to the verifier that a new version of the tree completely contains the old tree. The verifier checks that the hash of the proof elements matches the old tree root.

We call a combination of an inclusion and consistency proof an *insertion proof.*

These insertion proofs permit *log monitors* to audit the log. The tree structure requires only a modest constant factor of additional storage.

Transparency logs need not only be used for certificates: for example, the Go programming language uses transparency logs to help mitigate supply chain attacks in the Go module ecosystem [17, 51].

Work by Eijdenberg et al. describes a *verifiable map*, which is similar to the Merkelized Patricia trie stuctures used in the Ethereum blockchain [20]. Verifiable maps were originally motivated to support efficient verifiable certificate revocation, but can support any arbitrary data. A verifiable map uses a sparse Merkle tree structure and stores a cryptographic hash of an element in a location determined by the cryptographic hash of its key. This structure permits inclusion proofs, but not consistency proofs.

Verifiable maps can augmented with a transparency log to allow audits of the sequence of operations used to produce the map. These are referred to as *log-based maps.* Later work, such as CONIKS [43], builds upon this verifiable map structure to provide key transparency for end-to-end encrypted messaging systems.

2.3 Semantic Hashing

Unlike cryptographic hash functions, semantic hash functions — also known as similarity-preserving hash functions — are designed to produce outputs that are similar for similar inputs. Their existence is derived from the Johnson-Lindenstrauss lemma, which shows that techniques for dimensionality reduction exist that preserve the Euclidean distance from the projected dimension within some error bound.

One particularly popular construction in the information retrieval community is SimHash, proposed by Charikar [13]. SimHash uses random projections to compute similarity-preserving hashes. The Hamming distance between two SimHashes provides an estimate of the angular distance between these vectors in the original space. Henzinger and others identify that SimHash is a state-of-the-art method for identifying plagiarized content in large web corpora, outperforming alternative similarity preserving hashing algorithms [24]. We provide a sketch of SimHash in Algorithm 1.

Algorithm 1: SimHash

```
def InitSimHash(embeddingDim, simHashDim, r):
    matrix = FloatMatrix(embeddingDim, simhashDim);
    for element ∈ matrix :
        element = SampleUnitGaussian(r);
    return matrix
def SimHash(matrix, embedding):
    output = MatrixVectorMultiply(matrix, input);
    output = ElementwiseSign(output);
    return output
```

3 Definitions

We now formalize our definition of a waterlog. For convenience we will describe these parties and interactions in terms of an LLM service because we are motivated by the need to determine the provenance of text-based content. We note, however, that this is without loss of generality. These definitions apply to any service providing any modality of synthetic content.

3.1 Parties

A waterlog is a protocol between four parties: the users $\mathcal{U}$, the synthetic content providers $\mathcal{P}$, investigators $\mathcal{I}$, and the log operator $\mathcal{L}$.

Providers $\mathcal{P}$. LLM providers are hosted providers of a language model, and respond to prompts chosen by users. Our setting may support one or more providers. In our definitions below, LLM providers are assumed to execute a protocol semi-honestly at the time of text generation. Once the LLM provider has responded to the user it may behave adversarially.

Users $\mathcal{U}$. LLM users are consumers of an LLM provider. They submit queries (prompts) to the LLM provider and obtain a response of synthetic text. The users are semi-honest at the time of their query to the LLM provider, but may become adversarial to some extent after they receive a response.

Investigators $\mathcal{I}$. Each deployment may have one or more investigator, which is a party that wish to ascertain the provenance of text content they observe. We assume that investigators are rate-limited at a minimum but are otherwise untrustworthy. For example, they may adaptively query the log based on log responses.

Log Operator $\mathcal{L}$. The log operator is a single entity, unlike the other parties heretofore mentioned. Unlike the LLM provider and LLM user, the log operator is untrusted and is permitted to behave adversarially after setup.

3.2 Protocol Phases

There are three phases to a waterlog protocol:

Setup phase. In the initial phase, the log operator performs any needed setup; this is assumed to be performed honestly.

Generation phase. This phase of the the protocol encompasses the user's initial query to the synthetic content provider and the provider's response, along with any interactions with the log operator. Queries may occur many times.

Investigation phase. This phase occurs after the synthetic content has been received and verified by the user. During this period, the user is free to make bounded adversarial edits to the synthetic content. An investigator may observe (possibly adversarially modified) text from the user and query the log operator for the modified text.

3.3 Threat Model

Our threat model assumes the LLM providers $\mathcal{P}$ and LLM users $\mathcal{U}$ do not collude in the **Setup** or **Generation** phases. This assumption is standard for all watermarking schemes: if $\mathcal{P}$ and $\mathcal{U}$ collude then it is trivial to generate synthetic text that is not watermarked, because $\mathcal{P}$ and $\mathcal{U}$ can simply choose to forgo the watermarking protocol.

Most practical constructions of symmetric watermarks and asymmetric watermarks do not consider adversary access to quality and perturbation oracles. Some recent work shows that if the adversary has access to these oracles they can defeat virtually any watermark [55]. Access to a verification oracle presents a similar issue. An adversary who wishes to defeat a watermark can do so by applying various perturbations until the watermark fails to verify.

To address these infeasibility results, our threat model assumes that investigators are rate-limited and closely monitored.

3.4 Algorithms

A waterlog is a tuple of polynomial-time probabilistic (p.p.t.) algorithms, as described below:

$\mathsf{Setup}(1^\lambda)$**:** performs any needed setup to initialize the log using the provided randomness source.

$\mathsf{CreateEntry}(\mathsf{content}, \mathsf{metadata}) \to \mathsf{entry}, \mathsf{opening}$**:** generates a candidate log entry using the provided content and metadata.

$\mathsf{InsertEntry}(\mathsf{entry}) \to (\pi_{\mathsf{insert}}, \mathsf{root})$**:** inserts the log entry, generates a new root, and generates a cryptographic proof of inclusion and proof of consistency.

$\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root}) \to \{0, 1\}$**:** verifies the result from InsertEntry.

$\mathsf{Search}(\mathsf{query}) \to (\pi_{\mathsf{search}}, \mathsf{root}, \mathsf{candidates})$**:** searches for entries similar to the query, generates a proof and possible candidate matches.

$\mathsf{VerifySearch}(\pi_{\mathsf{search}}, \mathsf{root}) \to \{0, 1\}$**:** verifies the proof output by Search.

$\mathsf{VerifyMetadata}(\mathsf{entry}, \mathsf{metadata}, \mathsf{opening}) \to \{0, 1\}$**:** verifies the metadata from InsertEntry.

3.5 Security Properties

We formalize the properties outlined in §1 through the following definitions.

Notation. Let $\mathcal{S}$ be a waterlog scheme. Let $\mathcal{M}$ be the space of possible metadata. Let $\mathcal{C}$ be the space of all possible content and let $d_{\mathcal{C}}$ be a metric defined over $\mathcal{C}$. Let $\mathcal{A}$ be a p.p.t. adversary, let $\mathcal{B}$ be a p.p.t. challenger, and let $\mathcal{V}$ be a p.p.t. verifier. We also define λ to be a security parameter and $negl(\cdot)$ to be a negligible function.

Assumptions. We assume that correctness of VerifySearch holds with probability one. Specifically, this means that if an entry is correctly inserted as defined by InsertEntry then $\mathsf{VerifySearch}(\pi_{\mathsf{search}}, \mathsf{root})$ always succeeds for $\pi_{\mathsf{search}}, \mathsf{root}, \mathsf{candidates} \leftarrow \mathsf{Search}(\mathsf{query})$.

Definition 1 (δ-Correctness)**.** *Let* $e, \mathsf{opening} \leftarrow \mathsf{CreateEntry}(c, m)$ *and let* $(\pi_{\mathsf{insert}}, \mathsf{root}) \leftarrow \mathsf{InsertEntry}(e)$*. Finally, let* $(\pi_{\mathsf{search}}, \mathsf{entries}) \leftarrow \mathsf{Search}(o)$*. A scheme* $\mathcal{S}$ *is* δ*-correct if for all* $m \in \mathcal{M}$ *and* $c \in \mathcal{C}$*, both of the following hold:*

$$P(e \in \mathsf{entries}) \geq 1 - \delta$$

and

$$P(\mathsf{VerifySearch}(\pi_{\mathsf{search}}) = 1) = 1$$

Robustness. We provide two related definitions of robustness: ϵ-recall robustness and $(\epsilon, \delta_1, \delta_2)$ probabilistic robustness. We define recall robustness to be that an entry is not retreived only with negligible probability even under bounded adversarial edits; we formalize probabilistic robustness as a weaker (but more commonly used) notion of robustness.

Definition 2 (ϵ-Recall Robustness)**.** *Consider the following experiment between an adversary* $\mathcal{A}$ *and a challenger.*

1. $\mathcal{A}$ *selects some* content *and* metadata *and provides them to the challenger.*

2. *The challenger computes* $\mathsf{entry}, \mathsf{opening} \leftarrow \mathsf{CreateEntry}(\mathsf{content}, \mathsf{metadata}, r)$ *honestly (i.e. with true random coins) and then returns* $(\pi_{\mathsf{insert}}, \mathsf{root}) \leftarrow \mathsf{InsertEntry}(\mathsf{entry})$ *to* $\mathcal{A}$ *with* $\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root}) = 1$.

3. $\mathcal{A}$ *next outputs* $\mathsf{content}'$ *with the restriction that* $d_{\mathcal{C}}(\mathsf{content}', \mathsf{content}) < \epsilon$.

4. $\mathcal{A}$ *provides* $\mathsf{content}'$ *as* query *to* $\mathcal{O}$*, which then returns* $(\pi_{\mathsf{search}}, \mathsf{root}, \mathsf{candidates}) \leftarrow \mathsf{Search}(\mathsf{query})$.

We say $\mathcal{S}$ *is recall-robust if the following holds:*

$$P(\mathsf{entry} \in \mathsf{candidates}) \geq 1 - negl(\lambda)$$

Remark 3. *Observe that recall robustness is trivially met by returning every element of the log. At scale, this is unlikely to be useful. This motivates a definition of* probabilistic robustness *which has been implicitly used in the machine learning community [32].*

Definition 4 ($\epsilon, \delta_1, \delta_2$-Probabilistic Robustness)**.** *Consider the following experiment between an adversary* $\mathcal{A}$ *and a challenger.*

1. $\mathcal{A}$ *selects some* $\mathsf{content}_1$, $\mathsf{content}_2$, $\mathsf{metadata}_1$, *and* $\mathsf{metadata}_2$ *such that* $d_{\mathcal{C}}(\mathsf{content}_1, \mathsf{content}_2) > \epsilon$ *and provides them to the challenger.*

2. *The challenger computes* $\mathsf{entry}_1 \leftarrow \mathsf{CreateEntry}(\mathsf{content}_1, \mathsf{metadata}_1)$ *and then returns* $(\pi_{\mathsf{insert}}, \mathsf{root}) \leftarrow \mathsf{InsertEntry}(\mathsf{entry}_1)$ *to* $\mathcal{A}$.

3. $\mathcal{A}$ *modifies* content *to create some* $\mathsf{content}'$ *such that* $d_{\mathcal{C}}(\mathsf{content}', \mathsf{content}_1) < \epsilon$.

4. $\mathcal{A}$ *provides* $\mathsf{content}'$ *as* query *to* $\mathcal{O}$*, which returns* $(\pi_{\mathsf{search}}, \mathsf{root}, \mathsf{candidates}) \leftarrow \mathsf{Search}(\mathsf{query})$.

We say $\mathcal{S}$ *is* $\epsilon, \delta_1, \delta_2$*-probabilistic-robust if we have:*

$$P(\mathsf{entry}_1 \in \text{candidates}) \geq 1 - \delta_1$$

and

$$P(\mathsf{entry}_2 \notin \text{candidates}) \geq 1 - \delta_2$$

Prior work, such as [32], generally reports robustness as the true positive rate r_{tpr} at a given false positive rate r_{fpr}. Such a framework is probabilistic-robust with $\delta_1 = r_{\mathsf{tpr}}$ and $\delta_2 = r_{\mathsf{fpr}}$.

Definition 5 (Metadata Privacy)**.** *Consider the following randomized experiment between an adversary* $\mathcal{A}$ *with access to an oracle* $\mathcal{O}_{\mathsf{mdp}}$.

1. *The challenger selects a random bit* $b \in \{0, 1\}$.

2. *As many times as it wishes,* $\mathcal{A}$ *queries* $\mathcal{O}_{\mathsf{mdp}}(m_0, m_1, c)$ *where* $m_0, m_1 \in \mathcal{M}$ *and* $c \in \mathcal{C}$.

3. *The oracle returns the pair* (e_0, e_1) *where* $e_b \leftarrow \mathsf{CreateEntry}(c, m_0)$ *and* $e_{1-b} = \mathsf{CreateEntry}(c, m_1)$.

4. *At the conclusion of the experiment,* $\mathcal{A}$ *outputs a guess* b'.

We say $\mathcal{S}$ *is metadata-private if we have:*

$$|P(b = b') - \frac{1}{2}| \leq negl(\lambda)$$

Definition 6 (Metadata Authenticity)**.** *Consider the following randomized experiment between an adversary $\mathcal{A}$ and a verifier $\mathcal{V}$:*

1. *$\mathcal{A}$ selects $c_1, m_1, m_2, \mathsf{opening}_1, \mathsf{opening}_2$ with $m_1 \neq m_2$ and these values to $\mathcal{V}$.*
2. *$\mathcal{V}$ computes $v = \mathsf{VerifyMetadata}(c_1, m_2, \mathsf{opening}_2) \wedge \mathsf{VerifyMetadata}(c_1, m_1, \mathsf{opening}_1)$, where $\wedge$ denotes logical and.*

We say $\mathcal{S}$ is metadata-authentic if we have:

$$P(v = 1) \leq negl(\lambda)$$

Definition 7 (Non-Repudiation)**.** *Consider the following randomized experiment between a malicious adversary $\mathcal{A}$ and an honest verifier $\mathcal{V}$:*

1. *Phase 1: for i in $[0, t-1]$:*
   (a) *$\mathcal{A}$ chooses some $\mathsf{entry}_i, \pi_{\mathsf{insert}}, \mathsf{root}$ and provides these to $\mathcal{V}$.*
   (b) *$\mathcal{V}$ runs $\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root})$ and aborts if it returns 0.*
2. *Phase 2:*
   (a) *$\mathcal{A}$ selects an index i in $[0, t-1]$, an $\mathsf{entry}'_i \neq \mathsf{entry}_i$, and an π_{insert}.*
   (b) *$\mathcal{A}$ sends $(i, e'_i, \pi_{\mathsf{insert}})$ to $\mathcal{V}$.*
   (c) *$\mathcal{V}$ sets v to be the result of $\mathsf{VerifyInsertion}(e'_i, \pi_{\mathsf{insert}}, \mathsf{root})$.*

We say $\mathcal{S}$ is non-repudiable if we have:

$$P(v = 1) \leq negl(\lambda)$$

Definition 8 (Unframeability)**.** *Consider the following randomized experiment between a malicious adversary $\mathcal{A}$ and a verifier $\mathcal{V}$:*

1. *Phase 1: for i in $[0, t-1]$:*
   (a) *$\mathcal{A}$ chooses some $\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root}$ and provides these to $\mathcal{V}$.*
   (b) *$\mathcal{V}$ runs $\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root})$ and aborts if it returns 0.*
2. *Phase 2:*
   (a) *$\mathcal{A}$ selects an index i in $[0, t-1]$, some $m'_i \neq m_i$, and some $\mathsf{opening}$.*
   (b) *$\mathcal{A}$ sends $(i, m'_i, \mathsf{opening})$ to $\mathcal{V}$.*
   (c) *$\mathcal{V}$ sets v to be the result of $\mathsf{VerifyMetadata}(e_i, m_i, \mathsf{opening})$.*

We say $\mathcal{S}$ is unframeable if we have:

$$P(v = 1) \leq negl(\lambda)$$

4 Verifiable Hamming Distance Index

A waterlog relies on a novel data structure called a *verifiable Hamming distance index* to identify semantic hashes that are within a specific Hamming distance bound from a SimHash query. We detail the structure below, and describe how this is used to facilitate fast lookup in §5.

A common technique in succinct probabilitistic proof systems is the use of error correcting codes, such as Reed-Solomon encodings, to amplify distances. In our setting we wish to do the reverse — we wish to use techniques from error-correcting codes to dampen or reduce the distances between nearby SimHash vectors to facilitate neighborhood lookup.

One natural candidate for this is a *perfect code*, which is a code where every possible message can be efficiently decoded to a codeword. A well known result in coding theory is that there are only two classes of perfect codes over binary fields: Hamming codes and Golay codes. Recall that a binary Hamming code can correct a single error.

We recall from the SimHash design [13] that the probability two SimHashes h_a and h_b agree on n bits is given by

$$p^n = \left(1 - \frac{\arccos(\theta)}{\pi}\right)^n$$

Suppose we interpret these n bits as a (potentially noisy) codeword in a Hamming code $\mathcal{C}$ with message length m and decode routine decode. We show that this leads to a construction of an ϵ-recall robust index.

Lemma 9. *The recall of a Verifiable Hamming Index has a lower bound of*

$$p^n + \frac{2\,(1-p)\,p^{n-1}2^m}{2^n} + \frac{(1-p)^2\,p^{n-2}2^{m-1}}{2^n}$$

where p is a lower bound on the probability that two bits of SimHash agree.

Proof. Let w_H be the Hamming weight and let $\oplus$ refer to bitwise exclusive-or. Let the angular similarity between $\mathcal{A}$ and b be bounded below by p. Then there are four cases where they will decode to the same codeword:

- *Case 1:* $h_a = h_b$. Then we have $\mathsf{decode}(h_a) = \mathsf{decode}(h_b)$ trivially. This occurs with probability p^n.
- *Case 2:* $h_a \in \mathcal{C}$ and $w_H(h_a \oplus h_b) = 1$. Then we have $\mathsf{decode}(h_a) = \mathsf{decode}(h_b)$ because h_b is one error from h_a, which is a codeword. This case occurs with probability $(1-p)\,p^{n-1}\left(\frac{2^m}{2^n}\right)$.
- *Case 3:* $h_b \in \mathcal{C}$ and $w_H(h_a \oplus h_b) = 1$. Then we have $\mathsf{decode}(h_a) = \mathsf{decode}(h_b)$ because h_a is one error from h_b, which is a codeword. This case also occurs with probability $(1-p)\,p^{n-1}\left(\frac{2^m}{2^n}\right)$ due to symmetry.
- *Case 4:* There exists some $c \in \mathcal{C}$ such that $w_H(h_a \oplus c) = 1$, $w_H(h_b \oplus c) = 1$ and $w_H(h_a \oplus h_b) = 2$. Then we have $\mathsf{decode}(h_a) = \mathsf{decode}(h_b)$ because h_a is one error from codeword $\mathcal{C}$ and h_b is also one error from codeword $\mathcal{C}$. This case occurs with probability $(1-p)^2\,p^{n-2}\left(\frac{2^{m-1}}{2^n}\right)$.

Because these cases are mutually exclusive, the total probability q is simply the sum of each of these cases occurring:

$$\begin{aligned} q &= P\left(\mathsf{decode}(h_a) = \mathsf{decode}(h_b)\right) \\ &\geq p^n + \frac{2\,(1-p)\,p^{n-1}2^m}{2^n} + \frac{(1-p)^2\,p^{n-2}2^{m-1}}{2^n} \end{aligned}$$

□

Theorem 10. *A Verifiable Hamming Index is ϵ-Recall Robust.*

Proof. Suppose we wish to verify that SimHashes h_a and h_b of length l differ at most by $\lceil pl \rceil$ bits. Then we select parameters for a Hamming code to obtain q following Lemma 9. Suppose, without loss of generality, we wish to achieve a security level λ. We repeat the procedure t times:

$$t = \left\lceil \log_{1-q}\left(2^{-\lambda}\right) \right\rceil = \left\lceil \frac{\ln 2^{-\lambda}}{\ln 1 - q} \right\rceil = \left\lceil \frac{-\lambda \ln 2}{\ln 1 - q} \right\rceil$$

Note that we can repeat this procedure over multiple independent combinations of the bits of h_a and h_b to create these repetitions. □

To make the index verifiable, we combine this with the verifiable map structure we described in §2.2. The map key is the cryptographic hash of the combination and decoded codeword. The map value is a set of pointers to the values whose combinations decode to the same map key.

4.1 Audit Protocol

This construction permits an audit protocol to ensure the index is constructed correctly. An auditor can examine values in the log and calculate which index locations they should be located at. The audtior can then ask for an inclusion proof from those values to the root of the index root hash. Likewise, an auditor can examine codewords in the index and verify that every element pointed to by the index decodes to the selected codeword.

5 Construction

Our construction of a waterlog, which we name Distributed Record of End-user metaData and GEnerative output (DREDGE), is given in Algorithms 3, 4, 5, 6, and 7. Our construction is conceptually simple: we create cryptographic commitments over metadata and a SimHash of the synthetic text, log the commitment-SimHash pair in a transparency log, and then provide the synthetic text to the user.

We implement the waterlog scheme as follows:

1. The log operator $\mathcal{L}$ runs Setup.
2. The user queries the LLM provider $\mathcal{P}$ with a particular prompt.
3. The LLM provider $\mathcal{P}$ generates high-entropy synthetic text.
4. The LLM provider $\mathcal{P}$ creates a SimHash of the synthetic text runs $\mathsf{entry} \leftarrow \mathsf{CreateEntry}(\mathsf{synthetictext}, \mathsf{metadata})$. This is implemented by creating a cryptographic commitment of the metadata. The LLM provider submits the entry to the log operator $\mathcal{L}$.
5. The waterlog $\mathcal{L}$ runs $\pi_{\mathsf{insert}}, \mathsf{root} \leftarrow \mathsf{InsertEntry}(\mathsf{entry})$ and returns inclusion and consistency proofs to the LLM provider $\mathcal{P}$. This is implemented as a standard transparency log insertion operation.
6. The LLM provider $\mathcal{P}$ verifies the proofs by running $\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root})$. The LLM provider halts if then proofs are invalid.
7. The LLM provider returns the synthetic text, inclusion proofs, and consistency proofs to the user.
8. The user verifies the proofs by running $\mathsf{VerifyInsertion}(\mathsf{entry}, \pi_{\mathsf{insert}}, \mathsf{root})$. The user broadcasts a complaint if the proofs are invalid.

Algorithm 2: DREDGE Setup($1^\lambda, r$)

```
def Setup(1^λ, r):
    InitSimHash(embeddingDim, simHashDim, r);
```

Algorithm 3: DREDGE CreateEntry

```
def CreateEntry(r):
    while entropy < minEntropy :
        output, entropy = LLM(prompt; r);
    commitment, opening = Commit(metadata);
    embedding = GenerateEmbedding(output);
    simHash = SimHash(embedding);
    return WaterlogInsert(simHash, commitment);
```

Algorithm 4: DREDGE VerifyMetadata

```
def VerifyMetadata(entry, metadata, opening):
    return H(metadata —— opening) == entry.commitment;
```

Algorithm 5: DREDGE InsertEntry

```
def InsertEntry(simHash, commitment):
    index, proofs = TransparencyLogInsert(simHash, commitment);
    for i, c ∈ GetSimHashCombinations() :
        simHashSubset = c(simHash);
        correctedSubset = Decode(simHashSubset);
        mapKey = H(i —— correctedSubset);
        proofs = proofs ∪ VerifiableMapUpdate(mapKey, index);
    return (index, proofs);
```

Algorithm 6: DREDGE Search

```
def Search(text, threshold):
    embedding = GenerateEmbedding(text);
    simHash = SimHash(embedding);
    candidates = {};
    for i, c ∈ GetSimHashCombinations() :
        simHashSubset = c(simHash);
        correctedSubset = Decode(simHashSubset);
        mapKey = H(i, correctedSubset);
        correctedSubset = Decode(simHashSubset);
        mapKey = H(i, correctedSubset);
        for j, proof ∈ VerifiableMapGet(mapKey) :
            proofs.map.append(proof);
            candidates[j] += 1;
    commitments ← {};
    for j ∈ candidates :
        if candidates[j] ¿ threshold then
            proof, simHash, commitment = TransparencyLogGet(j);
            proofs.log.append(proof);
            commitments = commitments ∪ commitment;
        end
    return (proofs, commitments);
```

Algorithm 7: DREDGE VerifySearch

```
def VerifySearch(proofs, root):
    for proof ∈ proofs.vmap :
        if Verify(proof, root) ≠ 1 then
            return 0;
        end
    for proof ∈ proofs.log :
        if Verify(proof, root) ≠ 1 then
            return 0;
        end
    return 1;
```

This process is diagrammed in Figure 1 and detailed in Algorithm 3. An investigator can naïvely query the log for all outputs in a given time range and request that the commitments be opened.

To enable quicker lookups we create a verifiable Hamming distance index, which we detailed in §4. In brief, each key in the verifiable map is a unique combination of the semantic hash bits interpreted as a codeword in a perfect code, then decoded. We append the location of each matching log entry in the leaf of the verifiable map. We prove that these indexes are sound in §4.

The search protocol (i.e. the implementation of Search) is as follows:

1. The investigator observes content, then calculates the embedding and SimHash.
2. The investigator computes various combinations of SimHash, interprets them as a perfect codeword, and decodes them to the nearest codeword.
3. For each codeword, the investigator computes the cryptographic hash of the combination and the codeword. The investigator uses this as an index into the verifiable Hamming index's structure.
4. The verifiable Hamming index returns a set of pointers to candidate log entries and cryptographic proofs of inclusion and consistency (i.e. π_{search}).
5. The investigator runs VerifySearch to verify the index proofs.
6. The investigator queries the transparency log for each of the candidate log entries identified by querying the index.
7. DREDGE returns the SimHash, commitment pairs, inclusion proof, and consistency proof for each log entry (i.e. π_{insert}).
8. The investigator asks the LLM provider to open the commitments.

Thus, an investigator only need look up a small number number of log entries, assuming a uniform distribution of semantic hashes. This protocol is diagrammed in Figure 1 and detailed in Algorithm 6.

Security of DREDGE. A security analysis of DREDGE is given in Appendix F. Each of the security properties reduces to a contradiction in the hiding or binding properties of the underlying commitment scheme, or the collision resistance of the hash function used to build the Merkle tree structure.

6 Discussion

In contrast to symmetric watermarking, zero-knowledge symmetric watermarking, and asymmetric watermarking approaches we are able to achieve all of our ideal features:

Metadata A commitment is included alongside each SimHash in DREDGE. An investigator can request that the LLM provider open the commitment after detection. This balances metadata privacy with investigator verifiability.

Non-repudiation DREDGE prevents repudiation. The underlying transparency log provides an append-only property that is verifiable via inclusion and consistency proofs. This prevents an entry in DREDGE from being removed at a later time.

Unframeability A waterlog does not perturb any synthetic text outputs. Thus, DREDGE categorically prevents watermark stealing attacks. The append-only and metadata properties of DREDGE prevents framing text is synthetic. Suppose an adversary observes some speech it wishes to frame as synthetic. The adversary gets an LLM provider to generate the same text, but it is included in the log *a*fter the original text was observed. Thus, an adversary has to anticipate the text that it wishes to frame beforehand and get an LLM provider to generate it. We conjecture that this is difficult for high-entropy text outputs. Furthermore, a commitment of the adversary's metadata would be logged.

Robust The soundness proofs of the verifiable index include an angular distance parameter. There exists a mathematical relationship between the angular distance and the cosine distance which we describe in §B. Thus, a waterlog provides a tunable robustness parameter. Suppose an adversary is bounded and only able to perturb the synthetic text within a particular cosine distance. Then we can set the detection radius of DREDGE to match the adversary. We experimentally show robustness under various attacks in §7.2.

Federation DREDGE supports entries of synthetic text from multiple LLM providers.

Efficient DREDGE insertion and detection takes on the order of a few seconds. See §7.1 for details.

The construction of DREDGE and the concept of waterlogs is very different from prior watermarking approaches. See Table 2 for a comparison of both qualitative and quantitative differences. In spirit, a waterlog's semantic search approach is most similar to the work of Krishna et al. [35], but the additional cryptographic logging features forbid equivocation by the model operator.

We remark that is possible to improve precision to be perfect via auxiliary zero-knowledge proofs. A user who queries and retrieves commitments from DREDGE can ask the LLM provider to prove that the commitment does not open to a particular set of people. In this case the number of commitments retrieved by the SimHash correspond to the number of constraints in the proof.

6.1 Collusion

As discussed in §3.3, a waterlog requires that the LLM provider and user do not collude.

Additionally, non-collusion requires that the SimHash matrix and combinations be determined after the LLM is trained. This prevents any LLMs from being adversarially trained to produce outputs that cause the index to fail — either producing outputs that SimHash to the same value or by producing outputs that evade the bounds given in §4.

6.2 Rate-Limiting Investigators

As discussed in §3.3, unfettered access to a verification oracle allows an adversary to defeat a synthetic text detection scheme. A waterlog, as with any other publicly verifiable watermark, requires that investigators are rate-limited.

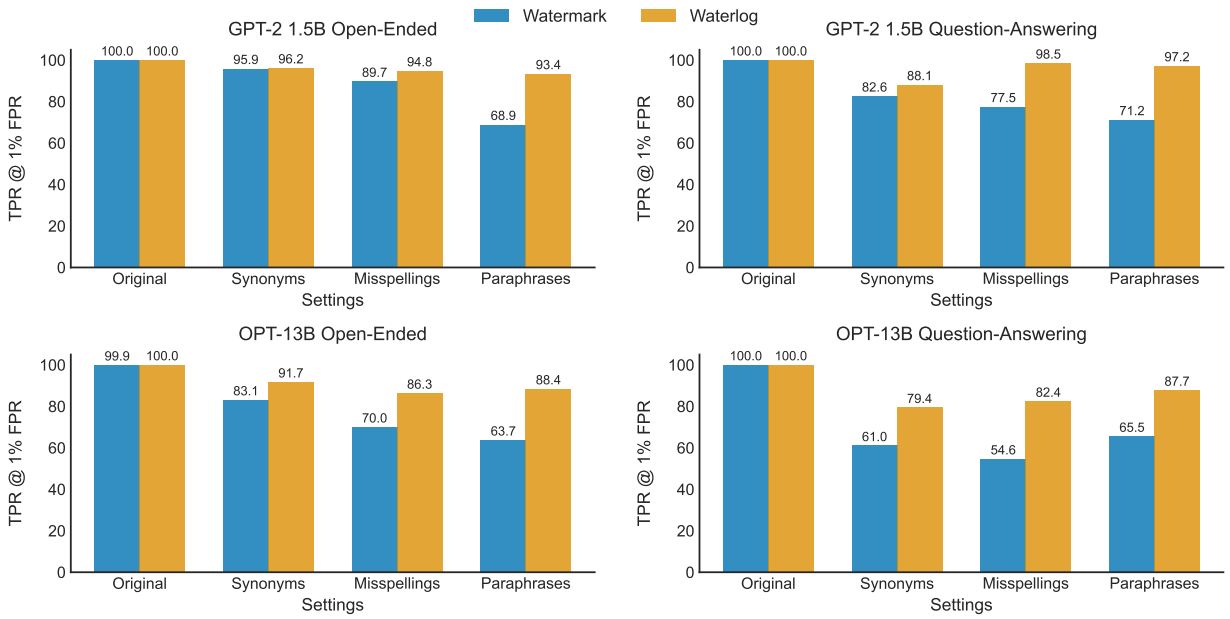


Figure 2: Robustness results for watermarks [32] compared to DREDGE. Results show the true positive rate at 1% false positive rate following the setup of [35]. These methods are tested on open-ended generations from Wikipedia in the Wikitext-103 [44] dataset as well as Long-Form Question Answering from the ELI5 subreddit. Misspellings are done for a random 50% of the text, synonyms with GPT-4o, and paraphrases with the Dipper model [35].

6.3 Split View Attacks

Because a waterlog is based upon standard transparency logs, it is possible to conduct a split view attack. This occurs where the waterlog presents two inconsistent views (roots) to clients. This prevents clients in partition A from seeing entries in partition B.

This is mitigated by gossiping waterlog state to all users. Selecting a gossip protocol is out of scope for this work.

6.4 Refusal to Open Commitments

Suppose the investigator queries the index and is learns a set of candidate entries that are within the log's angular similarity radius and its query. At this point the investigator has learned that the text is likely synthetic, even if the LLM provider refuses to open the commitments.

We believe that learning the text is likely synthetic is acceptable — indeed, this is the property both symmetric and asymmetric watermarks attempt to achieve.

6.5 Information Leakage from SimHashes

Morris et. al [46] demonstrate that embeddings can be inverted in many cases. We build off their work and demonstrate that we can invert our sentence-by-sentence embeddings with a high degree of accuracy — thus embeddings empirically preserve a large amount of entropy.

Because we leak bits of SimHash to the investigator, the confidentiality of the generated text in Waterlog rests on the entropy of outputs and the distribution of synthetic text outputs.

Suppose synthetic text outputs are always generated such that their embeddings are elementwise independent and identically distributed. Then the sum of the elements will tend towards a Gaussian distribution

by the central limit theorem. Furthermore, since elements of the SimHash matrix are drawn from Gaussian distributions centered at 0 then each the sum of the elements will also be centered at zero. Thus, the probability the sum's sign will be negative or positive will approach 50% as embedding vector dimensions approach infinity. Then the entropy of the distribution of a bit of SimHash will be one Shannon.

As discussed in §6.4, we believe logging a potentially invertible SimHash is acceptable because only a cryptographic commitment of the metadata is logged.

7 Results

We instantiate our construction in Sage 10.2 and Python 3.11. Prior literature uses a true positive rate at false positive rate metric to evaulate robustness; to match this we use 478 combinations interpreted as Hamming(31,26) codes and consider items that are found in at least 304 of the 478 verifiable Hamming distance indexes. We generate watermarked text following [32, 35] using Facebook's Open-Pretrained-Transformer (OPT) 13B model [57] as well as GPT-2 1.5B [50]. For our embedding model we use sentence-by-sentence embeddings generated by OpenAI's text-embedding-ada-002 [23] and concatenated together, then SimHashed to be 1024 bits long. Our transparency log is based on v1.3.12 of Google's Trillian [18], with some slight modifications to the serialization format of log roots.

7.1 Efficency

We provide microbenchmarks along various DREDGE operations in Figure 4. Microbenchmarks were run on a 10-core 3.2 GHz M1 Max MacBook Pro running macOS Sonoma 14.4.1. Memory backend microbenchmarks were conducted with a single core; Trillian microbenchmarks were run with DREDGE, Trillian, and MariaDB distributed across multiple cores.

Much of the time taken to index an entry in Figure 4 is an artifact of Trillian's internal implementation of a verifiable map; the time to index and time to retrieve an index scales poorly with the number of keys used. Additionally, versions of Trillian after v1.3.12 have removed verifiable map functionality. We note a long line of work [41, 14, 6, 38, 2] to improve the scalability and performance of verifiable maps for the purposes of key transparency and we believe that there exists a path to improved scalability for DREDGE.

7.2 Robustness

We compare DREDGE to the standard watermarking scheme proposed by Kirchenbauer et. al. 2023 [32]. Following previous work, we show the true positive rate (TPR) at the false positive rate of 1%. Watermarks simulate this tradeoff by altering the number of index entries we match and perform a search over the threshold parameter to match the 1% false positive rate.

We show the base performance of these methods, along with their performance under three attacks chosen from the most effective attacks in the watermarking literature [32, 35, 26]: attacks where we change out words for human-like synonyms, attacks which intentionally misspell words, and attacks that paraphrase. We change out or misspell 50% of words in the text and paraphrase the whole text. We prompt GPT-4o to generate synonyms in order to most closely match how a human would create synonyms (see Appendix D for the prompt). For misspellings, we randomly sample from one of four common mistakes: substituting similar sounding characters (e.g. "k" for "c"), omitting double letters when present, omitting a character or adding a random character, or transposing letters in a word. We use the Dipper paraphrase model from [35]. Examples of these attacks can be found in Appendix E.

We see the results of these robustness attacks in Table 3 (and in table form in Table 3). We find that watermarks are particularly susceptible to strong paraphrasing synonym, and misspelling attacks, similar to previous work.

In general, we find that DREDGE has a much higher true positive rate and lower false positive rate because it does not alter the distribution of synthetic text. We note that DREDGE, in contrast to symmetric

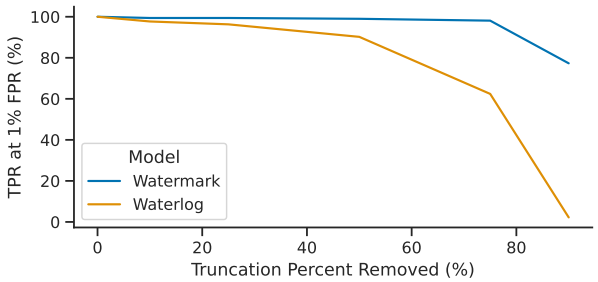


Figure 3: Results under a truncation attack that removes the first $N\%$ of the generation. We see that both do well until after 50% truncation, at which DREDGE does significantly worse because the truncation causes meaningful semantic changes.

watermarks, is most robust in the presence of paraphrasing attacks because of its reliance on natural language embeddings.

8 Comparison with Alternatives

In this section we compare our approach to two possible alternatives: zero-knowledge proofs of symmetric watermarks and asymmetric watermarks. We show that they fail to fully support the metadata, non-repudiation, unframeability, robustness, federation, and efficiency features outlined in §1.

8.1 Zero-Knowledge Proofs of Symmetric Watermarks

For our first possible alternative approach, we consider performing zero-knowledge proofs over a symmetric watermark. We are unaware of any proposals in the literature and believe we are the first to articulate such a scheme.

A zero-knowledge symmetric watermark is conceptually similar to regular symmetric watermarks. The LLM provider commits to a secret watermarking key and generates synthetic text under a symmetric watermark as normal. The LLM provider shows that the synthetic text is watermarked under the committed key by producing a proof-of-verification. Critically, the proof is only over the watermark verification algorithm and not over the entire inference process. The latter would require billions of constraints and make a proof computationally expensive to generate. At verification time a similar proof is computed over the text to verify with respect to the committed symmetric watermarking key.

Such an approach would address some — but not all — of the limitations we outlined with standard symmetric watermarks. For example, it would provide non-repudiation, inherit the original symmetric watermarking scheme's robustness, and be efficient. But it would still not provide us with the ideal features we identified in §1:

Metadata A zero-knowledge symmetric watermarking scheme does not include metadata.

Unframeability The proof does not mitigate watermark stealing attacks [16, 31]. An adversary can create text that would verify as watermarked.

Federation As with unaltered symmetric watermarks, verification must be done by the LLM provider. Delegating watermark verification to a third party requires providing that third party with the secret key.

8.2 Asymmetric Watermarks

In contrast to symmetric watermarking schemes, asymmetric watermarks are publicly verifiable because they embed a digital signature into text output. Verification is performed with a public verification key. The

construction outlined in [21] relies on a message-signature pairs. A signature on a previous block of tokens is embedded into the tokens of the following block. As long as one message-signature pair verifies, the entire text is considered to be synthetic.

While asymmetric watermarks provide non-repudiation and are publicly verifiable in the offline setting — and thus implicitly federated — they still do not provide the ideal features we identified in §1:

Metadata As currently constructed, asymmetric watermarks do not support metadata. However, it may be possible to encode additional metadata alongside the signature.

Unframeability As with zero-knowledge proofs of symmetric watermarks, an asymmetric watermark does not prevent adversaries from stealing the watermark [16, 31]. For example, an adversary could take a message-signature pair and embed this into their own text that would verify as watermarked.

Robustness Asymmetric watermarks are not robust and are trivial to defeat. It is sufficient to manipulate a single token in each message-signature pair to remove the watermark; verification relies on at least one message-signature pair remaining after adversarial edits. Because the verification algorithm is public, adversaries know the lengths of message-signature pairs. One solution may be to restrict adversaries to oracle-only access to verification. But this defeats the ability to detect the watermark in the offline setting and creates the same strange verification process outlined for symmetric watermarks — if a watermark is only verifiable by a centralized party, why outsource storage to adversary-controlled generated text?

Efficient Asymmetric watermarks are computationally expensive to generate. Experimental results in [21] indicate it can take up to 17 minutes to generate watermarked synthetic texts using LLMs with 1.3 billion parameters. This is due to overhead from rejection sampling. But modern LLMs are much larger. We extrapolate that the smallest version of Llama 2 (7 billion parameters) would take up to 90 minutes to generate watermarked output; larger sizes would take 2.8 hours (13 billion parameters) or 15 hours (70 billion parameters) [53]. This high overhead prevents practical deployment.

9 Extensions

As discussed in §C, we believe a natural extension of our work is to compute a set of zero knowledge proofs over the candidates returned by the verifiable index. We also acknowledge the undesirable requirement that the SimHash matrix be computed after the LLM is trained (see §6.1). We suggest that this can be eliminated using multiparty computation techniques. We leave both of these extensions to future work.

While we focus on synthetic text in this work, it would be trivial to extend to multimedia where appropriate embedding models exist. For example, it would be possible to extend this work to images with an image embedding model, audio with a audio embedding model, and video with a video embedding model. It may also be possible to support multimodal outputs in one waterlog instance via multimodal embedding models.

10 Conclusion

We have presented waterlogs, a new approach to synthetic text detection that provides several advantages over watermarking schemes. We have also demonstrated DREDGE, an implementation of a waterlog. We believe that the myriad advantages of waterlogs over watermarks demonstrates the need to further research other, non-watermarking schemes to detect synthetic content.

In addition, we believe our verifiable Hamming index construction may be of independent interest. We believe this new data structure is the first to provide verifiable searches over multi-dimensional data structures and motivates additional work into verifiable search techniques.

References

[1] Scott Aaronson. "Neurocryptography". Plenary Talk at Crypto'2023. Aug. 2023.

[2] *Advancing iMessage security: iMessage Contact Key Verification*. 2023. URL: https://security.apple.com/blog/imessage-contact-key-verification/.

[3] AI@Meta. "Llama 3 Model Card". In: (2024). URL: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[4] Ebtesam Almazrouei et al. *The Falcon Series of Open Language Models*. 2023. arXiv: 2311.16867 [cs.CL].

[5] Anthropic. *The Claude 3 Model Family: Opus, Sonnet, Haiku*. Mar. 2024. URL: https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

[6] Evan Au et al. *AKD: An implementation of an auditable key directory*. 2023. URL: https://github.com/facebook/akd.

[7] Diane Bartz and Krystal Hu. *OpenAI, Google, others pledge to watermark AI content for safety, White House says*. July 2023. URL: https://www.reuters.com/technology/openai-google-others-pledge-watermark-ai-content-safety-white-house-2023-07-21/.

[8] Joseph R. Biden. *Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence*. Oct. 2023. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence/.

[9] Massieh Kordi Boroujeny et al. *Multi-Bit Distortion-Free Watermarking for Large Language Models*. 2024. arXiv: 2402.16578 [cs.CL]. URL: https://arxiv.org/abs/2402.16578.

[10] Blake Brittain. *US Copyright Office denies protection for another AI-created image*. Sept. 2023. URL: https://www.reuters.com/legal/litigation/us-copyright-office-denies-protection-another-ai-created-image-2023-09-06/.

[11] Jon Brodkin. *US judge: Art created solely by artificial intelligence cannot be copyrighted*. Aug. 2023. URL: https://arstechnica.com/tech-policy/2023/08/us-judge-art-created-solely-by-artificial-intelligence-cannot-be-copyrighted/.

[12] Patrick Chao, Edgar Dobriban, and Hamed Hassani. *Watermarking Language Models with Error Correcting Codes*. 2024. arXiv: 2406.10281 [cs.CR]. URL: https://arxiv.org/abs/2406.10281.

[13] Moses S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 380–388. ISBN: 1581134959. DOI: 10.1145/509907.509965. URL: https://doi.org/10.1145/509907.509965.

[14] Melissa Chase et al. *SEEMless: Secure End-to-End Encrypted Messaging with less trust*. Cryptology ePrint Archive, Paper 2018/607. https://eprint.iacr.org/2018/607. 2018. URL: https://eprint.iacr.org/2018/607.

[15] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[16] Miranda Christ, Sam Gunn, and Or Zamir. *Undetectable Watermarks for Language Models*. Cryptology ePrint Archive, Paper 2023/763. https://eprint.iacr.org/2023/763. 2023. URL: https://eprint.iacr.org/2023/763.

[17] Russ Cox. *Transparent Logs for Skeptical Clients*. Mar. 2019. URL: https://research.swtch.com/tlog.

[18] Al Cutter et al. *Trilian*. URL: https://github.com/google/trillian.

[19] Benj Edwards. *US rejects AI copyright for famous state fair-winning Midjourney art*. Sept. 2023. URL: https://arstechnica.com/information-technology/2023/09/us-rejects-ai-copyright-for-famous-state-fair-winning-midjourney-art/.

[20] Adam Eijdenberg, Ben Laurie, and Al Cutter. *Verifiable Data Structures*. Nov. 2015.

[21] Jaiden Fairoze et al. *Publicly Detectable Watermarking for Language Models*. Cryptology ePrint Archive, Paper 2023/1661. https://eprint.iacr.org/2023/1661. 2023. URL: https://eprint.iacr.org/2023/1661.

[22] Ben Finley. *Athletic director used AI to frame principal with racist remarks in fake audio clip, police say*. Apr. 2024. URL: https://apnews.com/article/ai-artificial-intelligence-principal-audio-maryland-baltimore-county-pikesville-853ed171369bcbb888eb54f55195cb9c.

[23] Ryan Greene et al. *New and improved embedding model*. 2022. URL: https://openai.com/blog/new-and-improved-embedding-model.

[24] Monika Henzinger. "Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms". In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '06. Seattle, Washington, USA: Association for Computing Machinery, 2006, pp. 284–291. ISBN: 1595933697. DOI: 10.1145/1148170.1148222. URL: https://doi.org/10.1145/1148170.1148222.

[25] Abe Bohan Hou et al. "k-SemStamp: A Clustering-Based Semantic Watermark for Detection of Machine-Generated Text". In: *arXiv preprint arXiv:2402.11399* (2024).

[26] Abe Bohan Hou et al. "Semstamp: A semantic watermark with paraphrastic robustness for text generation". In: *arXiv preprint arXiv:2310.03991* (2023).

[27] Beryl A. Howell. *Memorandum Opinion*. Aug. 2023.

[28] Zhengmian Hu et al. *Unbiased Watermark for Large Language Models*. 2023. arXiv: 2310.10669 [cs.CR]. URL: https://arxiv.org/abs/2310.10669.

[29] Devriş İşler et al. *Puppy: A Publicly Verifiable Watermarking Protocol*. 2023. arXiv: 2312.09125 [cs.CR]. URL: https://arxiv.org/abs/2312.09125.

[30] Diane Jeantet and Maurico Savarese. *Brazilian city enacts an ordinance that was secretly written by ChatGPT*. Sept. 2023. URL: https://www.reuters.com/legal/litigation/us-copyright-office-denies-protection-another-ai-created-image-2023-09-06/.

[31] Nikola Jovanović, Robin Staab, and Martin Vechev. *Watermark Stealing in Large Language Models*. 2024. arXiv: 2402.19361 [cs.LG].

[32] John Kirchenbauer et al. *A Watermark for Large Language Models*. 2023. arXiv: 2301.10226 [cs.LG].

[33] John Kirchenbauer et al. *On the Reliability of Watermarks for Large Language Models*. 2023. arXiv: 2306.04634 [cs.LG].

[34] Kalpesh Krishna et al. *Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense*. 2023. arXiv: 2303.13408 [cs.CL].

[35] Kalpesh Krishna et al. "Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense". In: *Advances in Neural Information Processing Systems* 36 (2024).

[36] Ben Laurie. "Certificate Transparency". In: *Commun. ACM* 57.10 (Sept. 2014), pp. 40–46. ISSN: 0001-0782. DOI: 10.1145/2659897. URL: https://doi.org/10.1145/2659897.

[37] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962. June 2013. DOI: 10.17487/RFC6962. URL: https://www.rfc-editor.org/info/rfc6962.

[38] Sean Lawlor and Kevin Lewi. *Deploying key transparency at WhatsApp*. 2023. URL: https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/.

[39] Aiwei Liu et al. *A Semantic Invariant Robust Watermark for Large Language Models*. 2024. arXiv: 2310.06356 [cs.CR]. URL: https://arxiv.org/abs/2310.06356.

[40] Aiwei Liu et al. *An Unforgeable Publicly Verifiable Watermark for Large Language Models*. 2024. arXiv: 2307.16230 [cs.CL]. URL: https://arxiv.org/abs/2307.16230.

[41] Harjasleen Malvai et al. *Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging*. Cryptology ePrint Archive, Paper 2023/081. https://eprint.iacr.org/2023/081. 2023. DOI: 10.14722/ndss.2023.24545. URL: https://eprint.iacr.org/2023/081.

[42] Minjia Mao et al. *A Watermark for Low-entropy and Unbiased Generation in Large Language Models*. 2024. arXiv: 2405.14604 [cs.CL]. URL: https://arxiv.org/abs/2405.14604.

[43] Marcela S. Melara et al. *CONIKS: Bringing Key Transparency to End Users*. Cryptology ePrint Archive, Paper 2014/1004. https://eprint.iacr.org/2014/1004. 2014. URL: https://eprint.iacr.org/2014/1004.

[44] Stephen Merity et al. "Pointer sentinel mixture models". In: *arXiv preprint arXiv:1609.07843* (2016).

[45] Sara Merken. *New York lawyers sanctioned for using fake ChatGPT cases in legal brief*. June 2023. URL: https://www.reuters.com/legal/new-york-lawyers-sanctioned-using-fake-chatgpt-cases-legal-brief-2023-06-22/.

[46] John X. Morris et al. *Text Embeddings Reveal (Almost) As Much As Text*. 2023. arXiv: 2310.06816 [cs.CL].

[47] Larry Neumeister. *Lawyers blame ChatGPT for tricking them into citing bogus case law*. June 2023. URL: https://apnews.com/article/artificial-intelligence-chatgpt-courts-e15023d7e6fdf4f099aa122437db

[48] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[49] Wenjie Qu et al. *Provably Robust Multi-bit Watermarking for AI-generated Text via Error Correction Code*. 2024. arXiv: 2401.16820 [cs.CR].

[50] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[51] Filippo Valsorda Russ Cox. *Proposal: Secure the Public Go Module Ecosystem*. Apr. 2019. URL: https://golang.org/design/25530-sumdb.

[52] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL].

[53] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[54] KiYoon Yoo, Wonhyuk Ahn, and Nojun Kwak. *Advancing Beyond Identification: Multi-bit Watermark for Large Language Models*. 2024. arXiv: 2308.00221 [cs.CL]. URL: https://arxiv.org/abs/2308.00221.

[55] Hanlin Zhang et al. *Watermarks in the Sand: Impossibility of Strong Watermarking for Generative Models*. Cryptology ePrint Archive, Paper 2023/1776. https://eprint.iacr.org/2023/1776. 2023. URL: https://eprint.iacr.org/2023/1776.

[56] Ruisi Zhang et al. *REMARK-LLM: A Robust and Efficient Watermarking Framework for Generative Large Language Models*. 2023. DOI: 10.48550/arXiv.2310.12362.

[57] Susan Zhang et al. "Opt: Open pre-trained transformer language models". In: *arXiv preprint arXiv:2205.01068* (2022).

[58] Xuandong Zhao et al. *Provable Robust Watermarking for AI-Generated Text*. 2023. arXiv: 2306.17439 [cs.CL]. URL: https://arxiv.org/abs/2306.17439.

TPR @ 1% FPR		GPT-2 1.5B		OPT-13B	
Method	Setting	Open-Ended (↑)	Question-Answering (↑)	Open-Ended (↑)	Question-Answering (↑)
Watermark [32]	Original	100.0	100.0	99.9	100.0
	+ Synonyms	95.9	82.6	83.1	61.0
	+ Misspellings	89.7	77.5	70.0	54.6
	+ Paraphrases	68.9	71.2	63.7	65.5
Waterlog (ours)	Original	100.0	100.0	100.0	100.0
	+ Synonyms	96.2	88.1	91.7	79.4
	+ Misspellings	94.8	98.5	86.3	82.4
	+ Paraphrases	93.4	97.2	88.4	87.7

Table 3: Robustness results for Kirchenbauer et al. [32] compared to DREDGE. Results show the true positive rate at 1% false positive rate following the setup of [34]. These methods are tested on open-ended generations from Wikipedia in the Wikitext-103 [44] dataset as well as Long-Form Question Answering from the ELI5 subreddit. Misspellings and synonym swaps are done for a random 50% of the text while paraphrases are done with the Dipper model [35].

A Summary of Main Results

A tabular form of the main results can be found in Table 2. Histograms of microbenchmarks are given in Figure 4.

B Cosine Similarity and Angular Similarity

We remark here that SimHash operates on angular similarity, not cosine similarity. The relationship between angular similarity (S_θ) and cosine similarity ($S_{\cos\theta}$) is given by:

$$S_\theta = 1 - \frac{S_{\cos\theta}}{\pi}$$

Or, equivalently:

$$S_{\cos\theta} = \frac{\cos(\pi - \pi S_\theta)}{2}$$

Because $\frac{d}{d\theta}\cos\theta$ is 1 near $\theta = 0$, these similarity metrics are approximately equal at small angles. Additionally, the difference between cosine similarity and angular similarity is bounded across all angles. Thus, it is common in literature use cosine similarity as a stand-in for angular similarity because computing cosine similarity is less computationally expensive at scale.

C Verifiable Hamming Index Precision

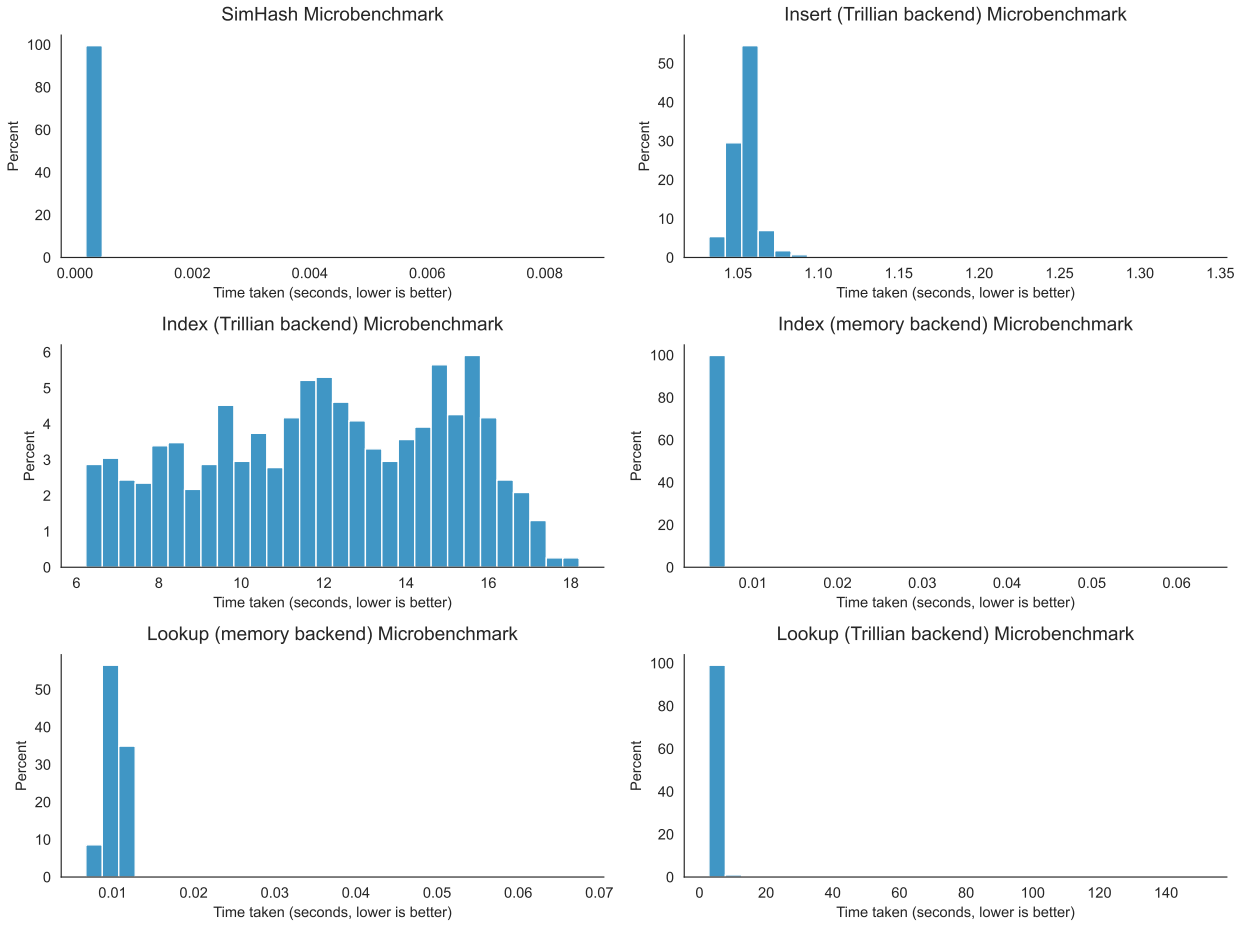
Observe also that the proofs we give in §4 only apply to recall and not precision. Thus, assuming adversary edits are bounded in their angular distance and DREDGE has been instantiated with parameters to detect that angular bound, DREDGE should always return matches.

The precision of the index is related to the choice of angular distance detection radius, parameters of the perfect code used in the verifiable Hamming index, and the distribution of the SimHashes.

Understanding the angular distance detection radius parameter is straightforward. If the angular distance radius increases, then recall will increase at the cost of precision. Unfortunately, the perfect code parameters and SimHash distribution are interrelated and more difficult to understand intuitively.

Suppose the message space of the perfect code is $\{0,1\}^m$ and SimHashes are distributed uniformly. Then expectation of the number of SimHashes at each index bucket is $n \times 2^{-m}$, where n is the total number of

Figure 4: Waterlog microbenchmark histograms. Microbenchmarks were run on a 3.2 GHz 10-core M1 Max MacBook Pro running macOS Sonoma 14.4.1. To capture the time taken by Trillian's implementation of a verifiable map, we also show microbenchmarks on an unauthenticated memory backend.



SimHashes. Then for $\mathcal{C}$ different combinations, we expect $c \times n \times 2^{-m}$ total SimHashes to be retrieved, some of which may be duplicates.

For the sake of concreteness, further suppose there are $2^{23} \approx 8 \times 10^6$ generation operations per day and SimHashes are distributed uniformly. Then using 478 Hamming(31, 26) codes, we would expect $478 \times 2^{23} \times 2^{-26} = 60$ entries to be returned in each DREDGE query per day of log activity. This results in $60 \times 365 = 21,900$ total results back per year of log activity.

Now suppose that the SimHashes are not distributed uniformly. It is possible to come up with a pathological case: suppose all SimHashes are the same value. Then the index returns every possible SimHash in DREDGE and is not helpful.

D Prompts

We use the following prompt to create the synonyms with GPT-4o, after randomly tagging 50% of words with "¡tag¿/¡tag¿" symbols. Note that although GPT-4o did not replace every word that was tagged, it was the only model which could somewhat perform this task – weaker models would ignore the tags.

Synonym Attack Prompt

Replace the following with synonyms where there are ¡tag¿ tags. E.g. replace "¡tag¿dog¡/tag¿" with "canine" in "the ¡tag¿dog¡/tag¿ was fun" to generate "the canine was fun". Be sure to remove the ¡tag¿ and ¡/tag¿ in the generated paragraph. You will be replacing every tagged word with a synonym and there will be lots of tags. Only replace words in tags.

Paragraph To Replace: INSERTED PARAGRAPH HERE

Be sure to replace **every** tagged word with a synonym and then remove the tags. Leave words the same if they do not have a tag. Return only the new text, nothing else.

E Example Attacks

In Table 4 we show an example generation and the attacks performed on it.

F Proof Sketches

Correctness and Robustness. Recall from Theorem 10 that a verifiable Hamming index can be constructed to be ϵ-recall robust. It is trivial to show that ϵ-recall robustness also implies $(\epsilon, \delta_1, \delta_2)$-probabilistic robustness.

Theorem 11. *Suppose a waterlog scheme $\mathcal{S}$ is ϵ_1-recall robust. Then $\mathcal{S}$ is also $(\epsilon_2, \delta_1, \delta_2)$-probabilistic robust with $\epsilon_1 = \epsilon_2$, $\delta_1 = negl(\lambda)$ and $0 \leq \delta_2 \leq 1$.*

We also note that that both ϵ-recall robustness and $(\epsilon, \delta_1, \delta_2)$-probabilistic robustness trivially imply δ-correctness (see Definition 1) if there exists a sound $\mathsf{VerifySearch}$ routine:

Theorem 12. *Suppose a waterlog scheme $\mathcal{S}$ is ϵ_1-recall robust. Then $\mathcal{S}$ is also δ-correct with $\delta = negl(\lambda)$ if $P(\mathsf{VerifySearch}(\pi_{\mathsf{search}}) = 1) \geq 1 - negl(\lambda)$.*

Theorem 13. *Suppose $\mathcal{S}$ is $(\epsilon, \delta_1, \delta_2)$-probabilistic robust. Then $\mathcal{S}$ is also δ-correct with $\delta = \delta_1$ if $P(\mathsf{VerifySearch}(\pi_{\mathsf{search}}) = 1) \geq 1 - negl(\lambda)$.*

Because prior work emphasizes both and precision, we configure DREDGE to only be $(\epsilon, \delta_1, \delta_2)$-probabilistically robust. This facilitates better comparison with prior art. See Table 2 for parameters.

Security Proof Sketches. Let $\mathcal{H}$ be a cryptographic hash function with first and second preimage resistance. We show that the security of DREDGE reduces to the preimage resistance of $\mathcal{H}$.

Theorem 14. *DREDGE is Metadata Private (see Definition 5).*

Proof Sketch. Let us assume that $\mathcal{A}$ can determine the random bit b encoded in the pair of $\mathsf{CreateEntry}$ oracle queries with non-negligible advantage. Note that in our experiment, the bit b is reflected only in the generation of a commitment.

If $\mathcal{A}$ is able to determine b with non-negligible advantage, then we can construct an attacker $\mathcal{A}'$ that succeeds with similar advantage in breaking the hiding property of the commitment scheme. For reasons of space, we omit the full reduction. □

Theorem 15. *DREDGE is Metadata Authentic (see Definition 6).*

Proof Sketch. Suppose that DREDGE is not metadata authentic. This means that malicious $\mathcal{A}$ can produce c and two pairs $m_1, \mathsf{opening}_1$ and $m_2, \mathsf{opening}_2$ where $m_1 \neq m_2$ such that c_1 is consistent with both pairs verify according to VerifyMetadata.

It is easy to show that this would require the production of a collision in the underlying commitment scheme used by CreateEntry, i.e., two distinct messages m_1, m_2 that each open to the same commitment. This is a clear violation of the binding property of the commitment scheme; if $\mathcal{A}$ succeeds in achieving this with non-negligible probability, then this directly implies an adversary that violates the security of the commitment scheme. □

Theorem 16. *DREDGE is Non-Repudiable (see Definition 7).*

Proof Sketch. Proof by contradiction. Suppose that DREDGE is repudiable. Thus, $\mathcal{A}$ can either (1) create multiple entries with CreateEntry that have the same hash or (2) can create the same Merkle root after running InsertEntry on different sequences of entries. We showed in the proof of theorem 15 that (1) creates a contradiction. But if the adversary is able to do (2), this implies that the second preimage resistance of $\mathcal{H}$ does not hold. But we have that the second preimage resistance of $\mathcal{H}$ does hold. □

Theorem 17. *DREDGE is Unframeable (see Definition 8).*

Proof Sketch. Proof by contradiction. Suppose that DREDGE is repudiable. Thus, $\mathcal{A}$ can either (1) create multiple entries with CreateEntry that have the same hash or (2) can create the same Merkle root after running InsertEntry on different sequences of entries. We showed in the proof of theorem 15 that (1) creates a contradiction and we showed in the proof of theorem 16 that (2) also creates a contradiction. □

Original	The opening song features Madonnaś vocals, plus backing vocals from Ellie Goulding. The teaser trailer released in February 2009 confirms that a ballroom routine took part at the beginning of the video, and also confirms that the sequence that was shot at the United States Armed Forces Institute at West Point was subsequently cut in the final version. The Chinese and French versions of the video feature two lines at the end of the title song that are missing from the English version. Both versions end with the phrase: "Come with me if you want to get married!" followed by "James Bond 007." The promotional film features Madonna appearing in a Caribbean palace, where a group of terrorists are holding her captive. She escapes, slips on a water gun, and defeats all the terrorists. In the end she is surrounded by her team, and says her line in the end: "Come with me if you want to get married." James Bond skydives into the party wearing a gold wingsuit to rescue her. In late September 2008, Bond 24-themed pub quiz nights began to appear across the UK (during the run of the Beatles-themed pub quiz show Have I Got News for You.) The questions were themed to the movie, with the grand prize being a meet and greet with the Queen of England and Daniel Craig. As of 2015, the last edition of this pub quiz had now been aired, most probably reflecting the postponement of the release date to January.
Synonyms	The opening song highlights Madonna's singing, plus supporting vocals from Ellie Goulding. The teaser preview released during February 2009 verifies that a ballroom performance occurred at the start of the video, and also verifies that the segment which was filmed at the United States Armed Forces Academy at West Point was eventually removed in the final version. The Chinese and French editions of the video include two lines at the conclusion of the title song that are absent from the English version. Both editions conclude with the phrase: "Come along with me if you desire to get wed!", followed by "James Bond 007." The promotional film highlights Madonna appearing in a Caribbean mansion, where a band of terrorists are holding her captive. She escapes, slips on a water gun, and defeats all the terrorists. In the end she is surrounded by her team, and says her phrase in the conclusion: "Come along with me if you desire to get wed." James Bond skydives into the celebration wearing a gold wingsuit to rescue her. During late September 2008, Bond 24-themed bar trivia nights began to appear across the UK (during the run of the Beatles-themed bar trivia show Have I Got News for You.) The questions were themed to the film, with the grand reward being a meet and greet with the Queen of England and Daniel Craig. As of 2015, the last version of this bar trivia had now been broadcast, most probably reflecting the postponement of the release schedule to January.
Misspellings	The obening osng features Madonnaś vocals, plus packinj vocals from lleie Goulding. the teaseyr trailer rleased in February 2900 confirms htat a balloom routine took part at the peginninj of eht video, and azlso confirms that thie seuence that aws shot at the United States Armed Forces inztitute ta West Point waz subequently cut in the final version. the Chinese and French versions op the video feature two lines atz the end of the ittle zong taht are missing fvrom the English version Both versions end with the pharse: "Come wiith me if you want t get married!",w followed yb "jasem Bond 007." The prnmotiooal pilm feaures Madonna appearing ni a cbribaean palace, where a group of terrorits are hnldiog ehr aptive. She escapes,y slspi on a ratew gun, and duefeats all the trrroeists. ni the end she is surrounded yb hr team, andx says her line in the end: "Come with em fi you want to gekt married." James bond skydives iotn the barty wearing a gold wingsuit to rescue her. in eatl September 2008, Bond 24-themed pub quiz nights began to aeppar across the uc (during the run of te Beatles-themed pub quizz sho have I Got newz for You.) eht questions were thedem to the movie, with the grand prize being a meet and geet wih the queen fo enjland and danile Craig. as o 2015, the last edition of tis pub cuiz hda now bene aired, mos probably reflcting teh postponement of the releae date to January.
Paraphrasing	The song features Madonna's vocals, with backing vocals by Ellie Goulding. A teaser trailer released in February 2009 confirms that the title sequence began with a ballroom dance sequence, and that a sequence shot at West Point, was cut from the final version. The Chinese and French versions of the title sequence feature two additional lines of lyrics, which are not in the English-language version. Both versions end with the words: "Come with me if you want to get married!" and the logo for the James Bond film. Madonna's performance of the song features her surrounded by terrorists in a Caribbean palace. She escapes, puts a water pistol to her head and defeats the terrorists. In the end she is surrounded by her team and delivers the line, "Come with me if you want to get married!" James Bond then parachutes in wearing a golden wingsuit and rescues her. In late September 2008, the BBC began to air a series of special editions of Have I Got News for You, themed on James Bond, with the main prize a meeting with the Queen and Daniel Craig. By the time the series was cancelled in January 2015, this was probably because of the change of release date to 1 November.
Truncation-50	The opening song features Madonnaś vocals, plus backing vocals from Ellie Goulding. The teaser trailer released in February 2009 confirms that a ballroom routine took part at the beginning of the video, and also confirms that the sequence that was shot at the United States Armed Forces Institute at West Point was subsequently cut in the final version. The Chinese and French versions of the video feature two lines at the end of the title song that are missing from the English version. Both versions end with the phrase: "Come with me if you want to get married!", followed by "James Bond 007." The promotional film features Madonna appearing in a Caribbean palace, where a group of terrorists are holding her captive. She escapes, slips
Truncation-90	The opening song features Madonna's vocals, plus backing vocals from Ellie Goulding. The teaser trailer released in February 2009 confirms that a ballroom routine took

Table 4: An example generation from OPT-13B from the Open-Ended dataset showcasing the effect of different attacks. Note that as synonyms are done automatically via hypernyms they are only weak synonyms.