AMP: Authentication of Media via Provenance

Paul England, Henrique S. Malvar, Eric Horvitz, Jack W. Stokes, Cédric Fournet, Rebecca Burke-Aguero, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, John Deutscher, Shabnam Erfani, Matt Gaylor, Andrew Jenks, Kevin Kane, Elissa M. Redmiles, Alex Shamis, Isha Sharma, John C. Simmons, Sam Wenker, Anika Zaman
Microsoft
USA and UK

ABSTRACT

Advances in graphics and machine learning have led to the general availability of easy-to-use tools for modifying and synthesizing media. The proliferation of these tools threatens to cast doubt on the veracity of all media. One approach to thwarting the flow of fake media is to detect modified or synthesized media through machine learning methods. While detection may help in the short term, we believe that it is destined to fail as the quality of fake media generation continues to improve. Soon, neither humans nor algorithms will be able to reliably distinguish fake versus real content. Thus, pipelines for assuring the source and integrity of media will be required—and increasingly relied upon. We present AMP, a system that ensures the authentication of media via certifying provenance. AMP creates one or more publisher-signed manifests for a media instance uploaded by a content provider. These manifests are stored in a database allowing fast lookup from applications such as browsers. For reference, the manifests are also registered and signed by a permissioned ledger, implemented using the Confidential Consortium Framework (CCF). CCF employs both software and hardware techniques to ensure the integrity and transparency of all registered manifests. AMP, through its use of CCF, enables a consortium of media providers to govern the service while making all its operations auditable. The authenticity of the media can be communicated to the user via visual elements in the browser, indicating that an AMP manifest has been successfully located and verified.

CCS CONCEPTS

• **Security and privacy → Authentication**.

KEYWORDS

media security, media authentication, provenance, deepfakes

ACM Reference Format:
Paul England, Henrique S. Malvar, Eric Horvitz, Jack W. Stokes, Cédric Fournet, Rebecca Burke-Aguero, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, John Deutscher, Shabnam Erfani, Matt Gaylor, Andrew Jenks, Kevin Kane, Elissa M. Redmiles, Alex Shamis, Isha Sharma, John C. Simmons, Sam Wenker, Anika Zaman. 2021. AMP: Authentication of Media via Provenance. In *12th ACM Multimedia Systems Conference (MMSys '21) (MMSys 21), September 28-October 1, 2021, Istanbul, Turkey.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458305.3459599

MMSys 21, September 28-October 1, 2021, Istanbul, Turkey

ACM ISBN 978-1-4503-8434-6/21/09…$15.00
https://doi.org/10.1145/3458305.3459599

1 INTRODUCTION

Advances in graphics and machine learning have enabled the creation and distribution of easy-to-use tools for synthesizing fake media [36]. These tools enable non-expert users to modify or synthesize audiovisual media that looks convincingly real. Although subtle artifacts may be detected in some cases by experts or by statistical classifiers developed with machine learning, we expect that the march of technical progress will soon make it impossible to distinguish fake media from real. Tools for media synthesis, coupled with wide-scale distribution of social media, threaten to cause harm to individuals, institutions, and nations. More generally, widespread distribution of fake media has the potential to undermine society's trust in all media. With the rise of fake media, what can be done to protect the veracity of media and provide a pathway to trust?

We are pursuing an answer by providing users with reliable information about the source and authenticity of a media object, through a verifiable and trustworthy media authentication service. That should allow the consumer to rely on the reputation of the media producer to make informed decisions about the media's trustworthiness. For example, a media company or publisher can attest that it published a work in accordance with their editorial standards, or content captured at a certain location and time by cameras in the hands of a trusted reporting team.

The simplest building block for proving provenance is to sign the media object digitally. However, the variety of mechanisms for media distribution, with many of them modifying the media files or streams, means that maintaining digital signatures is difficult. Additional challenges are also involved. For example, in a typical redistribution scenario, media content is re-encoded by a content distribution network (CDN). Such re-encodings are needed to address variations in channel bandwidth, rendering device resolution, and other constraints. To preserve provenance information, certificates must be tracked and re-inserted for each transformation.

We present a practical system named AMP (for *a*uthentication of *m*edia via *p*rovenance) aimed at providing robust verification of provenance while supporting a wide variety of production and distribution scenarios at internet scale. The AMP effort brings together expertise in security and media, leveraging advances in cryptography, watermarking, and recently released cloud security and ledger services.

Threats to the integrity of sources include the use of a range of techniques, from simple modifications of timing to more sophisticated uses of graphics and generative models, for manipulating or synthesizing audiovisual content that is perceived by consumers as capturing actual events.

Approaches to securing media from a reputable provider to its consumption include (1) strong authentication and (2) fragile watermarking or fingerprinting. A complementary approach involves (3) the detection of manipulation or synthesis via machine-learned techniques. In AMP, we focus on securing media based on the joint use of (1) and (2), to assess the identity of the media provider.

The AMP system consists of four main modules including the *AMP Service*, the *Media Provenance Ledger*, the *Manifest Database*, and the *AMP Authoring Tools*. AMP authenticates media using a digitally signed data structure called a *manifest*, and the *AMP Service* allows content providers to upload their media manifests to AMP. Manifests are registered in the *Media Provenance Ledger*, which is a public distributed ledger based on the Confidential Consortium Framework (CCF) [24, 25]. CCF is an efficient, open-source framework that can be used to implement permissioned blockchains. Manifests can be distributed together with media contents, whereas the ledger ensures integrity and auditability of the full history of media publishing operations. In addition, manifests are indexed by media fragments in a *Manifest Database* for fast querying. Once a manifest or group of manifests has been uploaded, media players can then use the *AMP Service* to validate the authenticity of the corresponding media contents, even if the content is distributed without its manifest. A set of tools allows content providers to interact with the *AMP Service* when the content is published. In addition to the service and tools, media players (browsers, smartphone applications, etc.) need to be extended to check and display provenance information. We have conducted a separate mixed-method study of the user experience for a media provenance system such as AMP [33]. The study provides evidence that provenance would be very helpful to combat the fake media distribution problem for end users.

Enabling large-scale media provenance requires the cooperation of multiple participants, including content producers, publishers, and technology providers. We have formed the Project Origin [30] coalition between Microsoft and three media publishers (BBC, CBC and New York Times) to assert media provenance based on the AMP system. In addition, we recently co-founded the Coalition for Content Provenance and Authenticity (C2PA) [7] aimed at creating a broadly adopted standard for media provenance verification, leveraging key methods from AMP. In this paper, we make the following novel contributions: 1) We describe an end-to-end solution for media producers to provide provenance information for each media item produced. In particular, the system is designed for *streaming* video including adaptive bit-rate streaming. 2) We describe how these videos can be distributed via CDNs or social media platforms while maintaining the required provenance information. We allow for certificate verification even when the distribution path is unknown, such as unknown CDNs or mild edits. 3) We propose a detached manifest that can be used for detecting provenance in the near-term with existing media standards, and an embedded manifest specification that can be used with future standards. 4) We show how the use of novel ledger techniques can scale to handle the majority of media items produced for distribution on the internet.

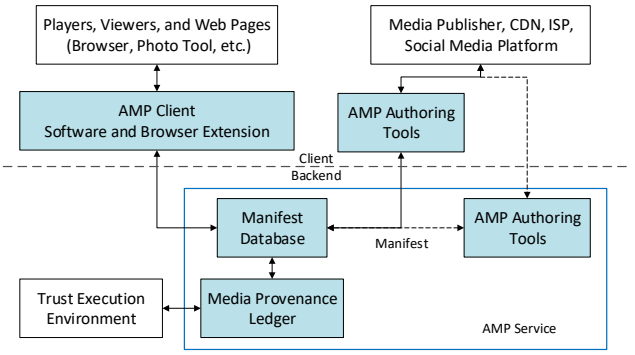

Figure 1: High-level overview of the AMP system components (shaded boxes).

2 AMP SYSTEM OVERVIEW

We provide an overview in this section of the core AMP concepts and how they are composed to form an end-to-end media authentication and verification system. We also specify the threat model. Figure 1 illustrates how the AMP components are integrated into a production, distribution, and rendering pipeline. A content provider uses the *AMP Authoring Tools* to create signed manifests and register them as part of publication, so that they can be authenticated by the *AMP Service*. The manifests can either be created locally by the publisher if they do not want to upload the media to the backend or alternatively in the *AMP Service* itself. Other organizations such as a CDN, social media platform, or Internet service provider (ISP) can similarly record transformations that they apply to the content provider's original media content, using the *AMP Authoring Tools*. The *AMP Service* records the resulting publication metadata in a manifest, signed by the provider (or the transformer), and stored in a *Manifest Database* (DB) for fast verification. One or more cryptographic hashes of the media content are also stored in a verifiable ledger, called the *Media Provenance Ledger*, using CCF. Finally, consumer applications such as browsers, web sites, or media players use AMP manifests and libraries for verifying (i.e., authenticating) that a media item indicated as coming from a content provider has been previously registered in the *AMP Service* by that provider.

2.1 AMP System Components

AMP Manifest. The manifest is the central data structure in AMP. The manifest authenticates media objects (including various cryptographic hashes of their encodings) and binds them to metadata provided by their publisher. Manifests support simple media objects, streaming media, progressive download, and adaptive bitrate streaming. A manifest can also record the attribution of derived works through "back-pointers" to one or more source objects, as well as descriptions of how the original works were transformed.

AMP includes two different sets of modules, one for a near-term solution and another for a long-term solution. The near-term modules indicate provenance using a detached manifest and can work with today's infrastructure. In addition, the long-term solution uses an embedded manifest, which is included in the media stream itself, but it will ultimately require extensions to browser standards.

Media Provenance Ledger. Manifests are recorded on a CCF blockchain. CCF operates the ledger (i.e., blockchain) of published works, which is essentially a list of manifests, relying on trusted hardware and providing high availability via the Raft [29] consensus protocol. Our implementation of CCF supports the registration of new manifests and issues signed manifest receipts. These receipts complement the producer's signatures; they enable any media consumers to independently verify that the media they receive has been published with the corresponding metadata. CCF natively supports online querying and validation of transactions along with their endorsing certificates.

Manifest Database. In the longer-term, we expect manifests to be distributed with the media objects themselves so that their provenance can be verified locally. To support a gradual transition, and to withstand the distribution of media without their associated manifests (e.g., media streamed from YouTube), AMP maintains an indexed manifest database, so that clients can retrieve manifests given media excerpts.

AMP Service. The *AMP Service* exposes the *Manifest Database* and the *Media Provenance Ledger* to client application through a set of REST APIs.

AMP Tools and Libraries. We also provide a set of tools and libraries for interacting with the *AMP Service*. The tools cover: (a) the creation, signing, and ingestion of content/manifests into the AMP system, (b) querying the AMP system for media authentication information and checking that media objects are intact, and (c) AMP service governance (adding/removing members and users, etc.).

Fragile Watermarking. In many cases, the media will be transformed without registering a manifest that records the transformation. To facilitate the retrieval of any manifest for the original media object, the publisher can insert a watermark using the AMP Watermark Tool, which carries a unique manifest identifier that can be used to retrieve the original contents and metadata, or to compare them with the transformed media.

User Experience Components. To provide a good user experience, AMP includes three components including two variants of a browser extension, two demonstration web pages, and a modified Chromium browser capable of displaying a new variant of an HTML video element.

Implementation. AMP has been implemented to run on Windows and Linux (Ubuntu LTS 18.04). The Media Provenance Ledger has been developed and tested on Ubuntu 18.04, the version of Linux currently supported by the CCF framework.

The core AMP components are primarily implemented in C# using .NET Core 3.1 so that the system will run on Linux, MacOS and Windows. The browser extension is implemented in JavaScript and HTML. CCF is primarily written in C++ although it allows applications to be written in C++ and JavaScript. The audio watermarking code is implemented in C. This implementation enables efficient porting to many different processing environments.

2.2 AMP Threat Model

AMP assumes that the computing infrastructures of legitimate content providers, redistributors, and consumers are secure, and that the AMP web service is not compromised. For the ledger, this assumes only that the trusted execution environment (e.g., SGX) and framework (CCF) are not compromised. AMP further assumes that the SHA-256 hash algorithm is collision resistant and that its X.509-based digital signature algorithms are secure.

3 AMP MANIFESTS

An AMP manifest is a data structure that cryptographically authenticates media objects and their associated metadata. Manifests are registered on the Media Provenance Ledger (Section 6), optionally distributed by media providers and distributors, and recorded in a complementary Manifest Database (Section 8). The purpose of manifests is to allow media player clients to quickly and easily verify the publisher (and possibly the distributor) of a media object. The values stored in the manifest data structure are generated by the content provider as it publishes the media object.

AMP supports two types of manifests: static and streaming. A static manifest handles a simple media object (e.g. JPEG) or a collection of objects with different encodings (facsimiles), while a streaming manifest contains an array of cryptographic hashes corresponding to "chunks" of the associated media. For example, a chunk might correspond to one or more seconds of video or audio.

AMP manifests can be used to authenticate original source material, or their transformation from one format to another. Note that checking whether a transformation is faithful is not discussed here.

AMP manifests are signed by publishers, CDNs, etc. The cryptographic hash of a manifest is called its AMP manifest ID (ManifestID). It serves as a unique identifier and a commitment for the manifest. ManifestIDs are also digitally signed by content producers or distributors, and recorded on the ledger. AMP uses X.509-based algorithms for all digital signatures and SHA-256 for all cryptographic hashes.

Static Manifests. Table 1 lists some of the fields included in manifests. The complete set of data structures can be found in the extended version of this work [10]. The publisher assigns a MediaID to identify a particular media object. In addition, the MediaID is encoded into the media object as a watermark and may also be inserted into the media's metadata.

The EncodingInformation field contains a string which indicates the media type (e.g., "JPEG", "MP4"). This field helps to guard against the media's cryptographic hashes being wrongly interpreted.

AMP manifests can also authenticate media objects that are derived from other media objects by means of "back pointers" to one or more source manifests. These "transformation manifests" can be used by publishers or CDNs to record transcoding and recompressions of source material. Transformation manifests can also be used to record the original media objects that were edited together to make a composite derived work.

The OriginManifestID field includes one or more ManifestIDs that describe the source media used to create a derived work. If a media object is a simple transcoding of another media object, this will be a single element array. If a media object is created from several source objects (e.g., a news video created from several

original media objects) then additional ManifestIDs can be recorded in the array. Note that OriginManifestID[] is not authoritative on its own: it should be trusted only if the ManifestID that describes the transform is signed by a trusted party.

The AMP manifest includes a Copyright field which can be used to provide the copyright string associated with the media object. This field provides a simple and legally enforceable way of limiting fake or misleading manifests. Allowed strings may also be dictated in the AMP terms of service.

In the simplest case (e.g., a picture or a text file), the manifest contains the cryptographic hash of the image or text and its associated metadata in the ObjectHash array field. Optionally, the publisher can create and authenticate more than one encoding of a media object to optimize for client screen resolutions or network conditions. We call these alternate representations *facsimiles*.

Streaming Manifests. AMP authenticates media objects with digital signatures. It is straightforward to do this with text and images: we simply generate the cryptographic hash and then sign picture.jpg or doc.html. Streaming media is more challenging because (a) an application should not have to wait to download the entire file before it can check the signature, (b) streaming services support changing the stream resolution to match network constraints (adaptive bitrate streaming), (c) some transport layers are lossy, and (d) users can often navigate back and forth in streams. These issues imply that AMP must authenticate much smaller regions (i.e., "chunks") in the stream.

All of the fields for the streaming manifest match those in the static manifest in Table 1 with the exception of the final field. While a static manifest contains one or more cryptographic hashes of an image or text document in the ObjectHash field, a streaming manifest contains a ChunkDigest which includes an ordered array of chunk-hashes.

Clients must be able to quickly determine where individual chunks start and end in order to be able to calculate the cryptographic hashes of the chunks and compare these against the entries in an AMP manifest. Unfortunately, different media formats and network delivery mechanisms require different chunking strategies.

In one case, the AMP system supports file offset-based chunking, which works well for HTTP GET-based streaming (which is most common on today's Internet). Lossy broadcast streaming requires different chunking strategies, such as I-frame-to-I-frame chunks for an MPEG stream. Practically, streaming players process a cryptographic hash of a chunk every few seconds. In most scenarios, consecutive chunks delivered to the client will map to consecutive chunk-hashes in a single manifest. However, if a server is dynamically switching streams, then more than one manifest may be needed to authenticate a stream.

AMP also supports adaptive bitrate streaming protocols such as DASH and HLS. Adaptive bitrate streaming requires several different encodings of a media object, optimized for different network conditions and client capabilities. Adaptive bitrate streams are supported in AMP either by publishing several manifests authenticating the different encodings, or by using a single manifest that authenticates multiple facsimiles.

Detached and Embedded Manifests. Initially, before encoding standards can be modified, manifests will be stored separately from the media itself; we refer to them as "detached manifests". Long-term, we hope that "embedded manifests" will be contained within the media's metadata and be transported within the media stream itself. We have implemented two versions of AMP utilizing both detached manifests and embedded manifests.

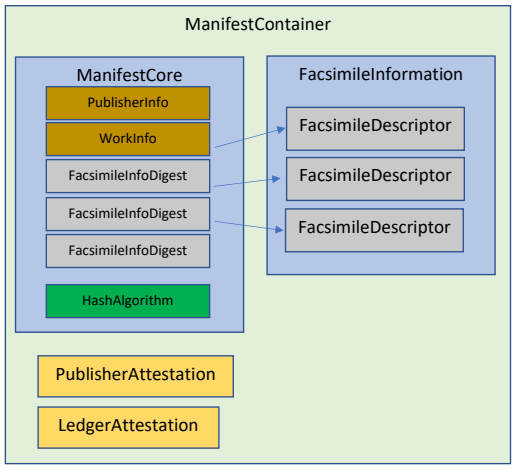

Figure 2: A simplified illustration of a ManifestCore and related data structures. Many fields and some data structures are omitted for clarity.

4 MANIFEST DETAILS

This section provides additional details about the full manifest which is depicted in Figure 2. A manifest is actually a container (ManifestContainer) and includes a core manifest called the ManifestCore, a FacssimileInformation to describe facsimiles of the original media object, and two structures which provide supporting evidence of the publisher (PublisherAttestation) and the ledger (LedgerAttestation). Before describing the manifest details, we next provide an overview of how to use a manifest.

Using Manifests. Manifests can be created by publishers, redistributors (CDNs, ISPs), social media platforms, recording devices, etc. and manifests are signed by the entity that created them. Manifests can also be countersigned by cloud services: for example, the CCF cloud service produces a signed receipt to acknowledge that a Media Manifest has been recorded on a ledger. Manifests are conventionally JSON or CBOR encoded and can be translated back and forth from JSON to CBOR as needed. For ease of development, the manifest authoring tools sign both the CBOR representation (using a COSE signature) and the JSON representation (using a JWT signature). The ManifestID is the hash of the CBOR-encoded manifest. Manifests can be delivered to clients as metadata with the actual media objects. However, since the ecosystem for delivering media is complex, we expect that it will take time for this delivery infrastructure to be widespread. Considering this, AMP provides a *Manifest Database* that clients can use to search for a manifest for a work. There are several ways to query the *Manifest Database*,

Field	Manifest Type	Description
MediaID	Static/Streaming	Publisher-assigned identifier for the media object.
MasterCopyLocator	Static/Streaming	URI of a stable, publisher provided location service or a generic URL redirector service.
EncodingInformation	Static/Streaming	String describing the media type (e.g., "JPEG", "MP4").
OriginManifestID[]	Static/Streaming	One or more ManifestIDs that describe the source media used to create a derived work.
Copyright	Static/Streaming	Copyright string associated with the media object.
ObjectHash[]	Static	Cryptographic hash of the associated simple media object (or collection of related media objects).
ChunkDigest	Streaming	An ordered array of chunk-hashes starting from the beginning of the work.

Table 1: Key manifest fields.

including querying by ManifestID, querying by the hash of the entire work, or an array of hashes of chunks of the work.

Authenticating Works. Each manifest authenticates either precisely one work, or several facsimiles of a work. There are no technical restrictions on what constitutes a facsimile, but the intention is that facsimiles support the very common scenario in which web sites, CDNs, etc. prepare a family of media objects (images, video, audio) that are optimized for different devices and network conditions, but all of which represent the same content – just not the same exact bits.

Manifests broadly contain two classes of data: metadata and media bindings. Metadata is publisher-assigned data, such as a publisher name and a title for the work. Media bindings describe the facsimiles: for example, cryptographic digests of the media, or subsets/chunks of the media and media type information. These fields are described in more detail in the following sections.

Metadata. Most metadata is contained in the structures PublisherInfo and WorkInfo, with the option to include facsimile-specific information in the FacsimileDescriptor structure. This design intentionally limits the metadata that is defined in this structure, and still less is mandatory. A minimal set of metadata would be the name of the publisher and the name of the work. If additional metadata needs to be attached, then it can be expressed in the OtherClaims data structures. The manifest supports an array of OtherClaims structures to be included in PublisherInfo (claims about the publisher), WorkInfo (claims about the work), FacsimileDescriptor (claims about the facsimile), and SourceWork (describing how a source work was transformed to produce a derived work). OtherClaims allows two sorts of claims to be associated with the manifest. Claim-sets can be embedded directly into the manifest, or a URI (or other descriptor) can be used to associate claims outside the manifest. In the case of external claims, OtherData allows the option that the manifest can cryptographically commit to the external claims by including the hash of the external data in the OtherClaims structure. OtherClaims contains a string type descriptor. We define a few standard descriptors such as "XMP", "EIDR", "SCHEMA", and then use a DNS-style namespace to allow extensions.

Media Bindings. *Authentication using an Object Digest.* All facsimiles are authenticated by hashing the entirety of the data that constitutes the facsimile: the hash of the entire file (e.g., PDF, JPG, MP4, OGV). Some commonly used multimedia standards allow multiple streams to be packaged in a single object. In some cases, it still makes sense to authenticate the entire container file or stream. In other cases, a subset of the underlying media file is authenticated. One important example of this is when the manifest is packaged in the media file itself – for example, when the manifest is embedded in an ISO/MPEG container. In all cases, the manifest directly or indirectly specifies exactly what parts of the media object are hashed.

Authentication using Chunking. Most modern media players download and buffer a few seconds of media and then start playing almost immediately, so authenticating a media object based on the hash of the whole file is inappropriate. To support progressive/streaming playback of media, the system supports streaming authentication using a collection of the hashes of "chunks" of the media object. Different media delivery schemes demand different chunking schemes. Two chunking schemes are currently supported: file-offset-based chunking and a Merkle-tree based scheme for MP4-containerized video. Each facsimile can be authenticated using more than one chunking scheme to allow a single work to be delivered in multiple ways.

File-Offset-Based Chunking. The most common media rendering technology on the web today is the HTML5 video element. The simplest way of using an HTML5 video player is to configure the video element to fetch video data from a URL. In this case, the video element performs a sequence of HTTP partial-GET operations to fetch the video data. File-offset-based chunking can be used to do progressive authentication in this case: the manifest contains an array of hashes of (say) 256KB chunks of the underlying video file, and the video player or browser calculates video-stream hashes and checks that they match a manifest. File-offset-based hashing can also work with Adaptive Bitrate (ABR) Streaming in some circumstances. ABR on the web is enabled by video player logic (often a JavaScript library running in the web page) fetching audio and video data from a collection of files encoded at different bandwidths. File-offset-based chunking still works in this case: each of the underlying video files is chunked, hashed, and encoded in the manifest. The SimpleChunkList data structure is used to represent encoded file-offset-based chunking. SimpleChunkLists contain an array of hashes and the size of the underlying chunks. The size of each chunk is recorded in the manifest, but we additionally define some standard lengths to enable chunk-hashes to be calculated by clients when they do not yet have a valid manifest. The final chunk in a file may be less than the chunk size.

MP4-Container Hashing and Merkle Tree Authentication. The MP4 ISO/IEC container format is a widely used standard for encoding

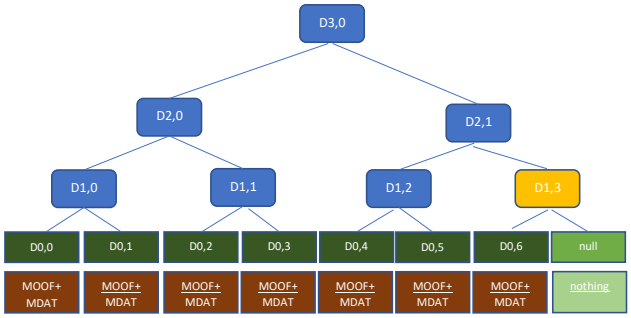

Figure 3: Merkle Hash Tree formed over multimedia data. The "leaves" of the hash tree are the hashes of the media samples, and each row in the hash tree is formed from the hash of the concatenation of the two hashes in the lower layer. The top-hash is called the root of the hash tree. If the number of samples is not a power of two, the leaves of the "missing" samples at the end of the file are null, and are processed according to the rules in this section.

any sort of media object in a file or stream. MP4 defines "box" types for holding multimedia data and metadata. For our purposes, the following box types are important: *MOOV* - Basic stream metadata: one per container, *MDAT* - Video or audio data: typically, a few seconds, and *MOOF* - Describes the samples in the subsequent MDAT. The simplest fragmented MP4 container contains {MOOV [MOOF, MDAT] +}, but most containers have additional boxes. MP4-Container-chunking defines a chunk as a subset of the MOOF data that defines the sample, together with the corresponding video data: i.e., the MDAT. Chunk hashes defined in this way can be embedded in a ManifestCore using the MerkleTreeAuthenticator, which is described next.

Merkle Tree Authentication. Typical chunk sizes for fragmented MP4 are a few seconds long, so the chunk hash data can be quite large. If the authentication data is encoded as a simple array, then the array of chunk hashes must be available in its entirety before authentication can begin. The MerkleTreeAuthenticator is an alternative representation of the chunk-hashes that allows authentication to begin when only a subset of the authentication data is available. This is achieved by encoding part of the authentication data in the manifest, and additional "evidence" in the media stream itself. Together, these allow a player to check that a media chunk is consistent with a manifest. This form of authentication is supported by encoding the authentication data as a Merkle hash tree. A Merkle Tree, depicted in Figure 3, is a binary tree of hashes, where the leaves of the tree are the digests of the [MOOF MDAT] samples, and each row in the tree is the hash of the data or hashes in the row beneath.

The Merkle Tree authenticator is encoded in two parts, which are typically distributed separately. The actual Media Manifest contains one row of hashes from the tree: for example, the D2,0 and D2,1 digests in Figure 3. This would be sufficient to authenticate the video data as long as the player can read and hash all of the data leading up to D2,0 or D2,1, but (in this example) the player would have to read, chunk, and hash half of the file before authentication could begin. To avoid the need for excessive read-ahead, the media can be distributed with the relevant missing parts of the tree, so that the player can validate that a particular chunk hash is consistent with the manifest. For example in Figure 3, to prove that the first sample is consistent with D2,0 the evidence would be D0,1 and D1,2 because these hash values can be used to form the missing parts of the tree.

The tree is formed as follows. The depth of the tree is determined by the number of chunks in the file. In general, the number of leaf hashes is not a power of two. In such cases, the tree depth is calculated by rounding up the number of leaf hashes to the next power of two. For example, if there are 5 chunks, then this rounds up to 8, which leads to a tree depth of 4, including the leaves of the tree. The general rules for forming the tree (in both the power-of-two and non-power of two cases) are as follows:

(1) The leaf hashes are formed from the hash of the chunk data.
(2) The "hash" of non-present chunk is termed null.

To form intermediate node hashes in the tree:

(1) If both inputs are non-null, then output = Hash (LHS|RHS)
(2) If one input (RHS) is null, then output is the other input (LHS)
(3) If both inputs are null, then output = null

The MerkleTreeAuthenticator data structure encodes one row of the hash tree in the Media Manifest, omitting null values. Encoding of the evidence hashes is described in the next section.

Encoding Evidence in an MP4 Container. The evidence that allows a player to determine that a chunk is consistent with an associated manifest is encoded in an MP4 box using standard BMFF extension mechanisms and is included as a peer of the MOOF box in the chunk.

Adaptive Bitrate Streaming. MPEG-DASH and Microsoft Smooth Streaming are adaptive bitrate streaming formats that allow a client player to select between different encodings of the same video object. Stream selection can happen when playing starts but, if network conditions change, it can also happen during playback. These streaming standards are usually enabled by creating a set of underlying compressed media files and dynamically assembling them into HLS or DASH objects with CMAF (MP4) chunks. The individual files are encoded using different bandwidths/compression ratios, and, for each bit rate, the original video is usually split into shorter files to allow client players to switch bandwidths every few seconds.

Adaptive streaming is supported by a set of ManifestCore structures by treating each of the separately encoded constituent files as a facsimile. In some cases, this might be a Transformation Manifest with a back-pointer to the manifest for an original high-definition file that was used to create the ABR streams, and in other cases the ABR streams will all be authenticated using a simple (non-transformation) manifest.

Transformation Manifests. Transformation Manifests are used to authenticate works that are transformed from other works. Transformation Manifests can be authored by the same publisher that created the original work, authored by an entity operating on behalf

of another entity (e.g. a CDN), or created by a completely unrelated entity, tool, or person.[1] Such manifests allow an entity to apply a transformation to a work, establish the original work as its source, and make a signed claim this transformation does not alter the meaning of the content of the original work. The manifest does not itself prove this assertion automatically but provides an auditable trail through which the assertion could be challenged. How such a challenge would be resolved is beyond the scope of this work; the manifest only ensures the transforming entity is accountable for the transformed works it releases.

Transformation Manifests differ from original work manifests in that they specify the ManifestID of the source work or works used to create the derived work, and also include the nature of the transformation applied. The primary initial scenario enabled by Transformation Manifests is re-encoding of a media object after the original manifest is created. However, we have allowed for future extensibility to express more complex sorts of derivation such as editing and media object composition. Such an extension of Transformation Manifests may allow for the meaning of the original work to be altered, but in a specific and documented way they assert what is acceptable and that the transformation does not alter the meaning of the transformed content. For example, a derivative work in the form of a news report might use a clip of a newsworthy event, and the producing entity could both assert the originality of its own content and make a claim that the clip of the event being described is unaltered, or itself transformed in some acceptable way, such as transcoded, or decorated with the entity's chyron or watermark.

Distributing Manifests and Manifest Containers. A simplified representation of a ManifestCore and related data structures in illustrated in Figure 2. The central data structure that cryptographically authenticates media is called the ManifestCore. A ManifestCore directly contains some data items, and cryptographic commitment to external data structures that may be distributed with the manifest or by other means. The ManifestCore uses commitments/hashes rather than embedding the data structure directly when the supplemental data is not always required. For example, the facsimile media authentication information is encoded in one or more external FacsimileDescriptors. This allows a media object to be distributed with only the FacsimileDescriptors that are relevant. For example, if a video object is encoded in WEBM and MP4, and each is encoded in 5 different bit rates and resolutions, this is 10 facsimiles. If a player is just playing one of these streams, then only the appropriate FacsimileDescriptor needs to be available to authenticate the stream.

In addition, there are a wide range of media metadata formats, and there is a wide range of data that a publisher might want to associate with a work; some of which the publisher might not want to distribute. The publisher can cryptographically commit to supplemental data by including the hash of the external data in the manifest.

The consequence of this is that a ManifestCore always needs additional data structures before it can be used to authenticate media. The ManifestContainer data structure is an envelope that allows a ManifestCore to be distributed with supplemental data structures that allow a work to be authenticated. Note that the ManifestCore cannot be modified after it is created because the MediaID would change, and signatures would break. However, ManifestContainers (each of which contains a ManifestCore) can be freely created with just the data needed for the intended purpose. ManifestContainers can also contain signature blocks and certificates from the publisher (PublisherAttestation and LedgerAttestation).

Signing Manifests. Manifests are typically signed by the originator (publisher, redistributor, social media platform, etc.) and may be countersigned by distributed ledger services. Manifest signatures are performed over the hash of a canonical representation of the manifest. JSON and CBOR representations are used by different parts of the system, so the manifest is signed twice: once to produce a JWT signature block (JSON) and once to produce a COSE signature block (CBOR). A PublisherAttestation optionally allows the signer certificate or certificate chain to be bundled in the ManifestContainer.

Canonicalization. ManifestIDs and signatures are over JSON or CBOR canonical encodings. JSON canonicalization follows the IEFT JCS draft. CBOR canonicalization follows RFC7049. COSE signatures follow RFC8152.

5 PROVENANCE BINDING

Authenticating that media has not been altered since the manifest was signed demonstrates the media's integrity, but tying the signer to an identity known and trusted by the consumer is what provides *provenance*, and allows the consumer's trust in that producer to be extended to the media. We have deployed a public key infrastructure (PKI) of X.509 certificates [9] governed and administered by the coalition or some other trusted organization to provide a *root of trust* for establishing identity. The coalition is then trusted to verify the identity of media producing organizations and individuals, and to issue credentials from its Certificate Authority (CA) to those organizations and individuals that can be used to sign manifests and authenticate to the *AMP Service*. We expect that this responsibility will be delegated to Certificate Authorities, who already provide these services for the authentication of secure web sites. They will perform due diligence in establishing the identity of media producer applicants for credentials under contractual obligations to the coalition. The hierarchical nature of a certificate-based identity system allows a single parent credential to be issued to the organization, which can then issue subordinate credentials for individuals or organizational units. The exact structure of the subtree of the PKI for a particular organization is beyond the scope of this design, as this custom-tailoring will be done to suit the specific needs and structure of each media producer.

Initially the root(s) of this PKI will be operated by the coalition and disconnected from the roots of trust currently used for the web PKI. Certificates used by participants will be given Extended Key Usage (EKU) extensions authorizing them for particular purposes. We have identified five uses and therefore five EKUs to use in this PKI: 1) server authentication, used by the *AMP Service* to authenticate itself to clients, 2) client authentication, used by clients to authenticate themselves to the *AMP Service*, 3) manifest signing, which will be used by producers to sign manifests, 4) time stamping, which will be used by the *AMP Service* and ledgers to attest to the

[1] Trust assessments when several parties are involved are not discussed here.

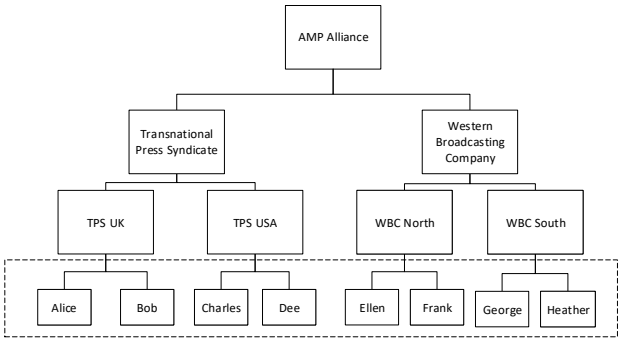

Figure 4: Example Public Key Infrastructure

publication time of a manifest, and 5) ledger registration, which will be used by ledgers to countersign manifests and attest they have been registered on that manifest. Server authentication, client authentication, and time stamping already have standard EKUs defined by the standard, and AMP uses those. Manifest signing and ledger registration EKUs are new purposes for which permanent, unique EKUs have not yet been allocated. We expect some certificates will be issued with multiple purposes: for example, the signer of a manifest will frequently be the client who registers it with the *AMP Service*, and so may use the same certificate for both purposes. Whether or not to combine these purposes in a single certificate becomes a governance decision for the coalition and media-producing entities, and the structure of our PKI allows for both possibilities.

One possible structure for such a PKI is given in Figure 4. A single root operated by the coalition sits at the top, and issues intermediate CA credentials to each participating organization: in this example, the Transnational Press Syndicate (TPS) and the Western Broadcasting Company (WBC). These organizations each in turn issue further credentials to units of their organization: UK and USA bureaus in the case of TPS, and North and South bureaus in the case of WBC. Below each of these intermediates are individuals, but they are enclosed in a dotted-line box because, as described above, these are optional: An organization may wish to issue signing credentials to individuals, in which case the organizational unit credentials are also intermediate CAs. Alternatively, organizations may wish to maintain centralized publication pipelines, ingest media from individuals through a mechanism external to AMP, and sign as part of this process. In this case, the organizational unit credentials themselves are leaf certificates.

6 MEDIA PROVENANCE LEDGER

AMP implements an instance of CCF [26] to build a ledger-based application that securely logs the cryptographic hash and copyright string of every manifest. Any application built with CCF is designed to be administered by a group of consortium members via CCF's governance features. Additionally, AMP utilizes signed receipts as standalone proof that manifests have been registered at a given index in the ledger.

CCF exposes to its users a key-value store. This key-value store provides a simple abstraction of keys being a cryptographic hash of a manifest (i.e., ManifestID), with the value being a signature computed by the publisher over a concatenation of the ManifestID and the copyright string (i.e., Copyright in Table 1). Once written, these key-value pairs are stored in a Merkle tree, and the Merkle tree is replicated and stored on persistent storage. To issue receipts, and to ensure that any tampering of the ledger can be detected, CCF maintains private keys for the service, and frequently uses them to sign the Merkle root of the whole ledger contents.

One of the core features that AMP utilizes from CCF is its universally verifiable receipts. The receipt for a given request validates the query, its response, and, more importantly, it certifies that its execution was recorded on the ledger. The key proposition of a receipt is that it is possible for anyone to cryptographically validate that the signature of the manifest's cryptographic hash and the copyright string were successfully recorded, based on just the manifest, the receipt, and the public key of the CCF service [24] without needing to contact the CCF service.

Our AMP system is designed to be run in a cloud datacenter. In a real-world implementation we expect and have designed the service to be run by an operator (such as Azure). CCF's utilization of trusted execution environments allows for the AMP ledger to be run in a public cloud while maintaining its security, even if the cloud provider or operator gets compromised. Manifests are recorded on a public blockchain using CCF. CCF operates the public ledger (i.e., blockchain) of published works, essentially a list of manifests, relying on a distributed network of replicas running on trusted hardware and synchronized using Practical Byzantine Fault Tolerance (PBFT) [8] or Raft [29]. CCF supports the registration of new manifests and issues signed manifest receipts. These receipts complement the producer's signatures; they enable any media consumers to independently verify that the work they receive has been published with the corresponding metadata. CCF also supports online querying and validation of ledger transactions and their endorsing certificates, as well as the transparent governance of the service by a consortium of media producers.

Governance. CCF provides a flexible governance model. This allows for AMP to define the governance by writing scripts in scripting languages such as JavaScript [16]. These scripts specify rules for actions such as adding new members, adding or removing users, adding and removing nodes from the system, user access control, etc. The specifics of the governance model will be defined as part of the media consortium that controls AMP, and these rules will evolve with time by modifying the governance scripts.

Trust and Integrity. CCF is designed to support two different types of consensus algorithms including Crash Fault Tolerance (CFT) and Byzantine Fault Tolerance (BFT). The CFT variant that CCF supports is a modified version of Raft [29], and the variant of BFT implemented by CCF is a modified version of Practical Byzantine Fault Tolerance (PBFT) [8]. CCF leverages trusted execution environments (TEEs) and specifically Intel's SGX. In CFT mode, both confidentiality and integrity relies on *every* TEE provisioned to run the service. Hence, the ledger is secure as long as Intel's SGX is not compromised. In BFT mode, confidentiality still relies on every TEE, whereas integrity (and progress) relies on *less than* $1/3$ of the TEEs that run the service being comprised. Hence, the ledger can withstand $f + 1$ active compromises out of $3f + 1$ TEEs. This

stronger guarantee can be leveraged by provisioning independently-managed TEEs, e.g., TEEs hosted in different data centers. Critically, both of these consensus protocols offer finality. This property states that, once a transaction has been committed and a receipt has been issued for it, it cannot be reverted.

7 FRAGILE WATERMARKING

We use watermarking to modify the media content in an imperceptible way. Faint noise-like patterns are inserted within the media content at production, and they can be read back at rendering. We tune the watermarking parameters such data media editing that preserves reasonably high fidelity preserves the detectability of watermarks, whereas heavier editing such as partial content replacement or fake media insertions [14, 19]) will render the watermarking indetectable. Hence the term fragile watermarking.

We propose the use of fragile watermarking techniques using a spread-spectrum approach [22], which adds low-level pseudorandom noise patterns within the media payload, be it video, audio, or images. The added noise is low enough (comparable to the small distortions due to the compression formats) and can be embedded in such a way that makes it imperceptible to human eyes and ears.

For each type of media and application scenario, we can design watermarking parameters that influence the thresholds on allowed changes, so that various kinds of minor modifications are considered as benign editing. In addition, we use keyless watermarking for AMP which simplifies system design and makes watermarking detection open, so it can be performed by any entity in the media distribution path.

Watermark Payload and Insertion. Table 2 describes the watermark payload, which is inserted into the media item and contains the following fields: a media object ID (MediaID), a publisher URI (MasterCopyLocator), and a signature over these two fields (WatermarkPayloadSignature). AMP does not provide a centralized database containing the MasterCopyLocator and MediaID. Instead after decoding, the client extracts the payload and submits the MediaID to the publisher using via MasterCopyLocator. Both the MediaID and the MasterCopyLocator are specified by the publisher. The MasterCopyLocator is typically a URI for the publisher's web service which is used to locate the media by their unique MediaID. The watermarking insertion process transforms a media object by embedding a signed watermark before its publication.

Watermark Decoding. The client inputs a media object to the Watermark Verification Module in the AMP libraries to extract the watermark payload fields depicted in Table 2. The Watermark Verification Module uses the MasterCopyLocator to obtain a signing certificate. Then, the Watermark Verification Module uses this signing certificate to check the WatermarkPayloadSignature over the MediaID and MasterCopyLocator. If this cryptographic step succeeds, it finally returns the MediaID and the MasterCopyLocator back to the client. Once the client has recovered the MasterCopyLocator and the MediaID, it can then contact the publisher's provenance service to authenticate that the media is valid. Watermark extraction is keyless: either it fails, or it returns the watermark payload.

To date, we have implemented an audio-only watermark which can be applied to both audio streams and videos with audio tracks. We leave image- and video-based watermarks as future research.

Field	Description
MediaID	Publisher-assigned identifier; same as in Table 1.
MasterCopyLocator	Same as in Table 1.
Watermark Payload Signature	Signature value over the MediaID and the MasterCopyLocator.

Table 2: Watermark payload.

8 MANIFEST DATABASE

Ideally in the future, all AMP manifests and ledger receipts will be delivered as additional metadata with the media objects. Delivering the receipt along with the media allows the client to quickly validate that the media has been previously authenticated without contacting the AMP Service. The widespread use of adding the manifest and receipt to the metadata will most likely require adoption by one or more browser standards. In the meantime, a client can use the *Manifest Database* to map a media object or chunk to a suitable manifest and receipt.

The AMP *Manifest Database* is implemented in Microsoft Azure using MongoDB [27] and contains manifests and receipts. It is exposed as a public service that lets clients obtain one or more AMP manifests and receipts that authenticate a published or transcoded media object. To perform this function efficiently, the *Manifest Database* uses the following indexes: (a) the MediaID delivered via the metadata or a watermark, and (b) the media ObjectHash or, in the case of streaming media, the cryptographic hashes of all of the contained chunks (ChunkDigest).

Media players can quickly and easily extract or calculate the ObjectHash or a ChunkDigest from the media, and then use the *Manifest Database* to find a matching manifest and the corresponding receipt. To validate the legitimacy of any manifest that was retrieved from the *Manifest Database* the following steps need to occur:

(1) The contents of the manifest will be hashed by a predetermined cryptographic hash function.
(2) The receipt will then be checked to ensure that it contains the previously calculated hash.
(3) The validator will then validate that the receipt is *endorsed* by the media provenance ledger via a signature over the receipt by the private key of the CCF service.

These steps ensure the validity of the manifest returned by the *Manifest Database* by proving it is produced and endorsed by the media provenance ledger.

The *Manifest Database* can be centralized or distributed. Because authoritative truth is stored in the ledger, the security requirements for the *Manifest Database* are much less than for the ledger itself. Note that AMP manifests do not address problems that arise from more than one publisher signing the same original content – either the same simple object or one or more ChunkDigests. Similarly, the AMP Service does not stop a rogue CDN from claiming that one media object is a faithful transformation of an original when in fact it has been maliciously authored. We believe that these issues can be

addressed by a combination of client policies (e.g., only consider the oldest manifest of a media object) and server-side terms-of-service.

Transformation Services. A Transformation Service takes one or more media objects and creates a derived object. A CDN is a simple example: CDNs can take a single media object and re-encodes it into several derived objects with different compression parameters to optimize for bandwidth and network losses. AMP manifests support transformation services by allowing entities to indicate the ManifestID of one or more source objects that were used to create the derived object.

Note that a transformation manifest does not in itself guarantee that a derived object is indeed a high-fidelity transformation of a source object. It is entirely possible that the "purportedly derived" object is unrelated to the stated original. Trust assessments should involve the entity that signed the transformation manifest. In the simple case, this might be the original publisher. For example, a media publisher creates a master media object and a dozen copies with different compression factors. A more complex example might be a CDN acting on behalf of the media publisher.

Policies can be developed for transitive trust that work for common scenarios. These policies can be enforced with a combination of client- and server-side rules, as well as server-side terms-of-service. Other entities might create and sign transformation manifests. For example, a third-party service might use heuristics to compare the semantic content of two videos and create and sign transformation manifests for the videos that they determine are semantically identical. Once more, AMP makes no trust assumptions: it is up to clients to use trust policies that are appropriate for a given scenario. In the case of streaming manifests, there is no requirement that source-chunks map 1:1 to transformed chunks: chunks are "natural" for each stream.

Manifest Revocation. As noted previously, CCF's ledger is immutable; once a manifest is stored on the CCF ledger, it cannot be removed. Therefore when a publisher wants to revoke a manifest from the ledger, it must insert a revocation object to the ledger. To enable efficient queries, the *Manifest Database* deletes this manifest in this case.

9 PERFORMANCE EVALUATION

Media Provenance Ledger. We first measure the time required to insert a manifest's relevant data into the *Media Provenance Ledger*. In this test, we insert strings, which consist of an example 256-bit cryptographic hash of a manifest (ManifestID) and a copyright string, into the ledger. These data structures do not need to be addressable in CCF since the fact that they are recorded in the ledger is sufficient. To this end, we measure the maximum sustainable rate at which a manifest's data can be submitted.

Application. We built a C++ application that customizes the CCF framework to produce a *Media Provenance Ledger*. The ledger application is small and can be programmed in a few hundred lines of C++ code. The following is an example of the data that the ledger application stores: "method": "LOG_record", "params": "id": 0, "msg": "88c3ba2b25cef698d9ca6775b7fd5c5e 8bbc246098a55ad51b8078834 c4add44 Copyright (c) CompanyName Corporation. All rights reserved."

Configuration 1	Throughput (tx/s)	Avg. latency (ms)
1 node	34,316	105
3 nodes	31,828	154
5 nodes	30,763	159
7 nodes	30,013	164
Configuration 2	Throughput (tx/s)	Avg. latency (ms)
1 node	34,316	105
3 nodes	32,415	244
5 nodes	31,617	245
7 nodes	30,500	248
Configuration 3	Throughput (tx/s)	Avg. latency (ms)
1 node	57,433	80
3 nodes	52,798	131
5 nodes	52,308	132
7 nodes	49,237	140

Table 3: Media Provenance Ledger throughput and latency.

Experimental Setup. We ran the performance application in three cluster configurations:

(1) *Single Azure Region* - Each computer is an Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz, and the application runs inside a 4 core virtual machine.
(2) *2 Geographically distributed Azure Regions* - Each computer is an Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz, and the application runs inside a 4 core virtual machine. The computers are evenly distributed between the east USA and west Europe Azure regions.
(3) *Controlled Environment* - A cluster that is running in our own datacenter. All computers are under the same 40G switch, and each computer is an Intel(R) Xeon(R) E-2288G CPU @ 3.70GHz which has 8 cores.

All of these VMs are running Ubuntu 18.04, and the results are shown in Table 3. We expect that there will be up to 1 billion entries added to the ledger every day, resulting in an expected load of 11,575 operations per second. We can conclude from these results that the proposed implementation of the *Media Provenance Ledger* can comfortably handle this load. Even with just a few nodes, we can achieve latencies that are low enough to not interfere with consumers' experiences with media consumption.

We also evaluate the possibility of replacing CCF with alternative blockchains. The results shown in Table 4 are either self reported results or results from publications that ran workloads with a similar access pattern to AMP. Additionally, for a fair comparison, we built an AMP-CCF equivalent application using Hyperledger fabric 2.2 [11], using the Go runtime. We ran it on Configuration 3 (Table 3) with 3 nodes, and this yielded a throughput of 1726 transactions/second (tx/s) with an average latency of 832 ms. These systems do not provide a receipt similar to CCF. We estimate that the system needs to support more than 11,000 transactions per second, and we can see that none of the options presented in Table 4 are able to meet this requirement.

System	Throughput (tx/s)	Avgerage latency (ms)
Ethereum Blockchain [4, 6, 12, 28]	15	3.6×10^6
Quorum [5]	1,650	500
Hyperledger Fabric 2.1 [39]	3,534.2	8,490
AMP-CCF on Hyperledger Fabric 2.2	1,726	832
Libra Blockchain [39]	561.7	53,450

Table 4: Throughput and latency measurements of other proposed ledger technologies.

From this comparison we can see the value of AMP using CCF as its backing storage mechanism. CCF provides independently-verifiable receipts, and is able to exceed the throughput and latency requirements of AMP.

AMP Service. To understand the performance of the AMP service, we measure the time required to write and read a manifest from the *Manifest Database* running on MongoDB in Azure. These measurements are provided in Table 5 for four different YouTube videos and show reasonable performance particularly for read operations.

Next, we estimate potential load requirements for the *AMP Service* assuming the following parameters: 10,000 publishers using the *AMP Service*, these active publishers upload 100, 10-minute original video clips uploaded each day, the video is divided into 10 second chunks (10 mins is 60 chunks) and each chunk is cryptographically hashed, and each original video is transformed into 99 (100-1) variants by the CDN.

Using these parameters, this translates into: 365 million original videos/year, 3.65 billion original and transformed videos/year, 22 billion original chunks/year, and 2.2 trillion total chunks/year.

Since the *AMP Service* is independent of the CCF nodes, we can use large-scale VMs for implementing the index. If the index is a 32-byte cryptographic hash and 32 bytes of other data (manifest Copyright field), the total index size for all known chunks is 1.4 TBytes. Azure offers VMs with enough memory and disk to hold the index in a single instance, and therefore the index will not require sharding.

If the *AMP Service* exceeds these estimates, we can shard the index. Scaling through sharding is easy: the indices are cryptographic hashes so they will be uniformly distributed. Therefore, we believe that it will be practical to have the *Manifest Database* indexed on chunk-hashes.

Audio Watermarking. AMP's audio watermarking module inserts a watermark into the frequency domain coefficients of the audio signal. It is important to measure the distortion introduced by the watermark, as we want it to be imperceptible. Table 5 measures the Objective Difference Grade (ODG) [3] for the audio channel of four different YouTube videos. The ODG ranges from 0 (no distortion) to -4 (high perceptual distortion). The mean and standard deviation are computed for five different trials with 1000 random bits of information inserted using 512 chips per information bit. Preliminary experiments show that watermarking generates no audible distortions.

10 COALITION AND STANDARDS

The proposed AMP media provenance certification and verification system can only be successful if it becomes part of a widely adopted industry standard. To this end, we collaborated on the creation of Project Origin [30], a coalition with three international media partners, including Microsoft, the British Broadcasting Corporation (BBC), Canadian Broadcasting Corporation (CBC), and the New York Times. Project Origin has now started to invite other media publishers to join the coalition. To date, AMP has served as the media technology platform for the Origin coalition. We have also co-founded the Coalition for Content Provenance and Authenticity (C2PA) [7]. The purpose of the C2PA is to form an open standard based on the best ideas proposed by Project Origin, including those in AMP, the Content Authenticity Initiative (CAI) [1], and other members. We believe that the implementation of a reliable provenance certification and verification system can be a significant step in increasing trust in media. The methods also promise to benefit the business models of all bona fide entities involved in the creation and distribution of media.

11 DISCUSSIONS

We expect that it will take a number of years before manifests for a large percentage of online media are stored in AMP. We believe this content gap and inability to report on the authenticity of media will be the biggest issue with adoption. An approach to providing useful provenance services in the interim will be to design signaling in the user-interface that informs consumers only when there is valuable information to provide to them. At the point when most media that is consumed does have authentication it would become prudent to report that authentication for some media is missing. A direction for future work is developing deeper understandings of how provenance information should best be conveyed to consumers to help them to evaluate content credibility [32, 38].

AMP does not address the detection of fake media. We believe that the quality of fake media will rapidly improve and become more pervasive. In the short term, algorithms for detecting fake and manipulated media may be able to provide valuable signals and filtering when incorporated into media processing pipelines. A number of academic and industry efforts are currently underway to improve the detection of deepfakes. We see this work as orthogonal to the provenance solution proposed by AMP, and these detection methods can also be included as part of the AMP service.

We have designed AMP to authenticate that a media item was published by a known source. AMP is not a digital rights management (DRM) system that is designed to enforce copyright of the media content providers. Media provenance and AMP are about verifying the producing entity, not verifying/tracking/authorizing the consuming entity. While it is possible to use AMP in this way, functionality such as self-verifiable receipts would work against this, and this is a property we do not intend to change.

AMP has a number of different threat vectors. Some of these include: (1) exploiting SGX vulnerabilities, which have been reported and mitigated in the past; (2) exploiting potential cryptographic weaknesses, such as signing-key compromises or SHA-256 collisions; (3) man-in-the-middle attacks, although AMP uses cryptographic protocols designed to prevent them; (4) malware on the

YouTube ID	Video Length (min:sec)	DB Write (sec)	DB Read (sec)	ODG
XFmn9kmZAWU	43:36	47.79 ± 4.67	0.3712 ± 0.0711	-0.74 ± 0.0066
xn_8UQ1W6_c	30:16	16.51 ± 0.35	0.3159 ± 0.0096	-1.51 ± 0.0043
bF_nULoyi9o	37:32	19.28 ± 1.01	0.3253 ± 0.0197	-0.99 ± 0.0039
iuX826AGXWU	28:50	6.38 ± 0.428	0.3110 ± 0.0073	-1.26 ± 0.0026

Table 5: Write and read times for the *Manifest Database* and Objective difference grade (ODG) scores for audio watermarking for four different YouTube videos of differing lengths.

media publishers', CDNs', and consumers' systems; and (5) malware in AMP's web service.

12 RELATED WORK

Provenance Systems. Using provenance for the prevention of deepfakes is a new and understudied area. One provenance-based system that is closely related to AMP was proposed recently by Hasan [12]. Like AMP, this system also employs blockchain. However, it is based on the Ethereum blockchain and smart contracts. Since AMP utilizes CCF, it is significantly more efficient, allowing the speedup of manifest insertion and queries by several orders of magnitude which is required for widespread deployment. In addition to [12], two startups and an NGO (non-governmental organization) have proposed provenance-based systems including: Amber, TruePic, and Witness.

Amber's technology [2, 28] is most closely related to AMP. The approach is aimed at camera manufacturers and adds a cryptographic hash to the video at a user specified rate. It is important to note that Amber's technology seems to only target the offline scenario and does not consider the streaming video case. Roderick Hodgson, Amber's cofounder, gave a talk on deepfakes at ACM Multimedia 2020 [13] where he reviews deepfake generation methods, considers legitimate and illegitimate uses of synthetic media, and provides an overview of detection and authentication. The talk includes content which discuss hashing in time for authentication. Specifically, they propose hashing based on invariant time windows using a hash of hashes which they claim are invariant to prepending, postpending, and trimming. The talk also includes an overview of their system. Amber signs the hashes with the device key, stores the hashes in a trusted database, and stores a reference to the hash in the file. Hodgson did not mention blockchain in the talk, but an earlier article published February 2019 [28] indicated that these hashes are stored on an Ethereum blockchain. Unlike AMP, the Amber system contains no watermarking. The talk describes including some metadata in the video file which sounds similar to AMP's embedded manifest, but since they did not provide any details, we cannot determine how it compares to the embedded manifest which is proposed in this paper. In addition, Amber does not propose a detached manifest which is critical to media authentication in the short term.

Similarly, Truepic [35] provides a photo and verification service where the cryptographic signature is written to a blockchain although they do not specify which type. Although Truepic's service originally focused on images, their web site now claims to support video though no supporting details are provided. Like Amber, Truepic is a proprietary startup which provides few details on their methodology.

Witness is a non-governmental organization which aims to help ensure that human rights abuses can be documented in a verifiable manner. Witness published the ProofMode Android application [17] in 2017 which stores metadata about images and videos taken by those seeking to provide evidence of human rights abuses. The app includes a hash of the media and its metadata along with a cryptographic signature that helps to ensure the chain of custody.
Provenance Partnerships. Several other partnerships have been created to ensure the provenance of media. The New York Times Company is working with IBM on the News Provenance Project [34] (NPP). That collaboration had been targeting images but appears to have ended. NPP also used the Hyperledger blockchain to provide a provenance solution for the images. Another example is the Content Authenticity Initiative (CAI) with Adobe, The New York Times Company and Twitter [1]. CAI released a whitepaper [15] on their technology in August 2020, but they focus on the content creation workstream whereas AMP targets the media distribution pipeline.
Deepfake Detection. The detection of deepfakes is an alternate method to provenance solutions and relies on the use of pattern recognition methods to detect synthetically generated media. A number of deepfake detection algorithms have been proposed in the literature [18, 20, 21, 23, 31, 37].

13 CONCLUSION

We have proposed, designed, and built the AMP system to help combat the problem of fake media. AMP allows trusted content providers to use services to register and certify the media they produce, allowing applications such as a media player or a browser to provide an indication to users that the source of the content they are viewing has been verified. Thus, instead of attempting to detect fake media, AMP focuses on certifying the source of the media. Beyond the core security pipeline, human factors and design will play an important role in the success of AMP. Inspired by the TLS lock icon, we believe that applications such as browsers and media players need to include UI elements to alert users that the received content can be traced back to its original source. For a provenance solution such as AMP to be successful, it must be formally adopted by a recognized standards body. We are working on the development of such standards, based in part on the AMP system, within the Coalition for Content Provenance and Authenticity (C2PA) effort.

REFERENCES

[1] Adobe. 2019. Introducing the Content Authenticity Initiative. https://theblog.adobe.com/content-authenticity-initiative/.

[2] Amber. 2019. Instilling Trust into Video. https://app.ambervideo.co/.

[3] Michael Arnold. 2002. Subjective and objective quality evaluation of watermarked audio tracks. In *Second International Conference on Web Delivering of Music, 2002. WEDELMUSIC 2002. Proceedings.* IEEE, IEEE Computer Society, USA, 161–167.

[4] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* Association for Computing Machinery, New York, NY, USA, 585–602.

[5] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. 2018. Performance evaluation of the quorum blockchain platform. , 8 pages.

[6] Blockchain. 2020. Ethereum transactions per second. https://blockchair.com/ethereum/charts/transactions-per-second. Accessed: 2020-07-31.

[7] C2PA. 2020. Coalition for Content Provenance and Authenticity. https://c2pa.org/.

[8] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. USENIX Association, USA, 173–186.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://tools.ietf.org/html/rfc5280.

[10] Paul England, Henrique S. Malvar, Eric Horvitz, Jack W. Stokes, Cédric Fournet, Rebecca Burke-Aguero, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, John Deutscher, Shabnam Erfani, Matt Gaylor, Andrew Jenks, Kevin Kane, Elissa Redmiles, Alex Shamis, Isha Sharma, John C. Simmons, Sam Wenker, and Anika Zaman. 2020. AMP: Authentication of Media via Provenance. arXiv:2001.07886 [cs.MM]

[11] Hyperledger Fabric. 2020. A Blockchain Platform for the Enterprise — hyperledger-fabricdocs master documentation. https://hyperledger-fabric.readthedocs.io/en/release-2.2/. (Accessed on 04/01/2021).

[12] Haya R. Hasan and Khaled Salah. 2019. Combating Deepfake Videos Using Blockchain and Smart Contracts. In *IEEE Access*, Vol. 7. IEEE, USA, 41596–41606.

[13] Roderick Hodgson. 2020. Preserving Video Truth: an Anti-Deepfakes Narrative. https://www.youtube.com/watch?v=zR-V1nf-dT0.

[14] https://github.com/deepfakes. 2020. faceswap: FaceSwap is a tool that utilizes deep learning to recognize and swap faces in pictures and videos. https://github.com/deepfakes/faceswap/.

[15] Content Authenticity Initiative. 2020. Creating the standard for digital content provenance. https://contentauthenticity.org/.

[16] javascript.com. 2021. ready to try JavaScript. https://www.javascript.com/.

[17] Dia Kayyali. 2017. ProofMode - Verified Visuals Enabled! https://blog.witness.org/2017/04/proofmode-helping-prove-human-rights-abuses-world/.

[18] Pavel Korshunov and Sebastien Marcel. 2018. DeepFakes: a New Threat to Face Recognition? Assessment and Detection.

[19] Marek Kowalski. 2018. FaceSwap. https://github.com/MarekKowalski/FaceSwap/.

[20] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. 2018. In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking. In *IEEE Workshop on Information Forensics and Security (WIFS).* IEEE Computer Society, USA, 1–7.

[21] Yuezun Li and Siwei Lyu. 2019. Exposing DeepFake Videos By Detecting Face Warping Artifacts. In *Workshop on Media Forensics.* IEEE, IEEE Computer Society, USA.

[22] H. S. Malvar and D. A. F. Florencio. 2003. Improved spread spectrum: a new modulation technique for robust watermarking. *IEEE Transactions on Signal Processing* 51, 4 (2003), 898–905.

[23] Scott McCloskey and Michael Albright. 2018. Detecting GAN-generated Imagery using Color Cues.

[24] Microsoft. 2019. CCF: A Framework for Building Confidential Verifiable Replicated Services. https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf.

[25] Microsoft. 2019. CCF documentation. https://microsoft.github.io/CCF/.

[26] Microsoft. 2019. Confidential Consortium Framework. https://github.com/Microsoft/CCF.

[27] MongoDB. 2020. The database for modern applications. https://www.mongodb.com/.

[28] Lily Hay Newman. 2019. A New Tool Protects Videos from Deepfakes and Tampering. https://www.wired.com/story/amber-authenticate-video-validation-blockchain-tampering-deepfakes/.

[29] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference (ATC).*

[30] Project Origin. 2020. Protecting Trusted Media. https://www.originproject.info/.

[31] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. 2019. FaceForensics++: Learning to Detect Manipulated Facial Images. In *ICCV 2019.* IEEE Computer Society, USA, 1–11.

[32] Julia Schwarz and Meredith Morris. 2011. Augmenting web pages and search results to support credibility assessment. In *Proceedings of the SIGCHI conference on human factors in computing systems.* Association for Computing Machinery, New York, NY, USA, 1245–1254.

[33] Imani N. Sherman, Elissa M. Redmiles, and Jack W. Stokes. 2020. Designing Indicators to Combat Fake Media. arXiv:2010.00544 [cs.HC]

[34] The New York Times. 2019. The News Provenance Project. https://www.newsprovenanceproject.com/.

[35] Truepic. 2019. Photo and video verification you can trust. https://truepic.com.

[36] Luisa Verdoliva. 2020. Media Forensics and DeepFakes: An Overview. In *IEEE Journal of Selected Topics in Signal Processing*, Vol. 14. IEEE, USA, 910–932.

[37] Xin Yang, Yuezun Li, and Siwei Lyu. 2019. Exposing Deep Fakes Using Inconsistent Head Poses. In *IEEE International Conference on Acoustics, Speech and Signal Processing.* IEEE Computer Society, USA, 8261–8265.

[38] Savvas Zannettou, Michael Sirivianos, Jeremy Blackburn, and Nicolas Kourtellis. 2019. The web of false information: Rumors, fake news, hoaxes, clickbait, and various other shenanigans. *Journal of Data and Information Quality (JDIQ)* 11, 3 (2019), 1–37.

[39] J. Zhang, J. Gao, Z. Wu, W. Yan, Q. Wo, Q. Li, and Z. Chen. 2019. Performance Analysis of the Libra Blockchain: An Experimental Study. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN).* IEEE Computer Society, USA, 77–83.