AMP: Authentication of Media via Provenance

Paul England, Henrique S. Malvar, Eric Horvitz, Jack W. Stokes, Cédric Fournet, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Shabnam Erfani, Kevin Kane, Alex Shamis
Microsoft

***Abstract*—Advances in graphics and machine learning algorithms have led to the general availability of easy-to-use tools for modifying and synthesizing media. The proliferation of these tools threatens democracies around the world by enabling widespread distribution of false information to billions of individuals via social media platforms, and by casting doubt on the veracity of all media. One approach to thwarting the flow of fake media is to detect modified or synthesized media via the use of pattern recognition methods, including statistical classifiers developed via machine learning. 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. Within a short period of time, 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 will be increasingly relied upon. We propose AMP, a system that ensures *a*uthentication of a *m*edia content's provider via *p*rovenance. AMP creates one or more 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, also 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 an icon or other visual elements in the browser, indicating that a AMP manifest has been successfully located and verified.**

I. Introduction

Over the last five years, leaps in graphics and machine learning methods have enabled the creation and distribution of efficient and easy-to-use tools for synthesizing fake media. These tools enable non-expert users to modify or synthesize audiovisual media that is indistinguishable from the media capture of real-world events. 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 advances will soon make it impossible to distinguish real from fake media. A perfect storm is brewing with the creation and distribution of disinformation via the coupling of increasingly powerful tools for modifying and generating media with the dominance of social media platforms. Tools for media synthesis, coupled with wide-scale distribution of social media threatens to undermine the Fourth Estate of journalism and cause harm to individuals, institutions, and nations. More generally, widespread distribution of fake media has the potential to undermine societys trust in the veracity of all video, audio, and imagery. Given the rising sea of fake media, what can we do to protect the veracity of media and provide a pathway to trust? We are pursuing an answer via harnessing a set of advances in computer security: We seek to provide users with reliable information about the source and authenticity of a media object via a verifiable, trustworthy media authentication service. Knowing the source of a media object, and having a certification that any alterations to the media are verifiable, allows the consumer to rely upon the reputation of the media producer to make informed decisions about the medias trustworthiness. For example, a media company or publisher can attest that it published a work in accordance with their editorial standards, or a camera in the hands of a reporter can attest that it recorded a video clip at a certain location and time. The simplest building block for proving provenance is to digitally sign the media object. 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.

Building a media authentication service requires solutions to multiple challenges. For example, in typical processes of redistribution and rendering, media content is routinely re-encoded by a content distribution network (CDN). A streaming system (e.g., Netflix) has pre-encoded streams, so one could try to embed authentication information into the media's metadata; however edits such as screen capture lose this metadata, and such derived content cannot be authenticated properly. Furthermore, to ensure an adequate quality of service (QoS) level, frames may be dropped. Thus, each type of transformation must be tracked throughout the transportation of the media.

We present AMP, a system that seeks a practical solution to the *a*uthentication of a *m*edia content's source based on *p*rovenance, while accounting for a wide variety of production and distribution scenarios at Internet scale. The paper provides a design for the AMP system. The design is framed by rising threats to the integrity of news sources, and thus to democracy. 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. A complementary approach involves (3) the detection of manipulation or synthesis via pattern recognition employing technical methods, such as machine-learned classifiers. Finally, there are opportunities to explore (4) event-certification methods for certifying that media as captured is linked to actual physical events, rooted in activities that are certified via a combination of methods to a time and place. The approach employed in the AMP system to securing

media is based on the joint use of (1) and (2), coupled with the certification of the identity and trust of the media provider. Strong authentication and fragile watermarking are individually promising for authenticating the provenance and veracity of media, per establishing certification and/or certifiable links to original sources of media as it was captured by reliable sources. We also seek a method that leverages the joint use of strong authentication and watermarking to provide valuable complementarity, per efficiencies and attack vectors.

This paper describes Microsoft's AMP initiative to develop technologies, a platform, and standards, for providing strong authentication and provenance information for media. AMP proposes to mitigate the negative societal impact of fake/synthetic media, based on certifiable provenance. The AMP effort brings together expertise in security and media, leveraging multiyear efforts in cryptography, watermarking, and recently released cloud security and ledger services.

The AMP system consists of four main modules including the Amp Service, the Media Provenance Ledger, the Manifest Database, and the AMP 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) [1], [2]. 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 have been uploaded, media players can then use the Amp Service to validate the authenticity of the corresponding media contents, even if it was 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.

Enabling large-scale media provenance will need the cooperation of multiple participants, including content producers, publishers, and technology providers. We envision AMP supporting a media provenance consortium with open governance rules, where all the governance operations are recorded in the AMP ledger, for auditability and transparency. We hope that the AMP project will be a starting point for broadly-adopted standards for media provenance verification.

II. AMP System Overview

This section provides an overview of the core AMP concepts and how they are combined to form an end-to-end media authentication and verification system. Figure 1 illustrates how the AMP components are integrated into a production, distribution and rendering pipeline. A content provider uses the AMP Tools to register the media as it is published, so that it can be authenticated by the AMP Service. Other organizations such as a CDN or internet service provider (ISP) can similarly record transformations they apply to the content

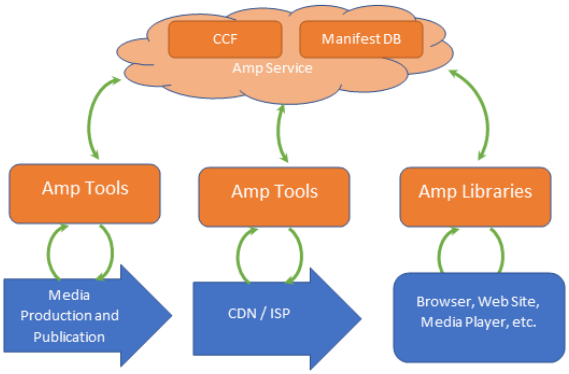

Fig. 1. Integration of AMP tools and services into a media production and distribution pipeline.

provider's original media content, using the AMP 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 a browser, a web site, or a media player use AMP manifests and libraries for verifying (i.e., authenticating) that a media item apparently from a content provider has been previously registered in the AMP Service by that provider.

A. AMP System Components

AMP Manifest. The manifest is the central data structure in AMP. It authenticates media objects (including various cryptographic hashes of their encodings) and binds them to their publisher-provided metadata. Manifests support simple media objects, streaming media, progressive download and adaptive bitrate streaming. A manifest can also records 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.

Media Provenance Ledger. 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) [3] or Raft [4]. CCF supports the registration of new manifests and issues signed manifest receipts. These receipts complement the producers 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.

Manifest Database. Eventually, we hope that manifests will be systematically distributed with the media object themselves, so that their provenance can be locally verified. To support a gradual transition, and withstand the distribution of media without their manifest (e.g., streamed from YouTube), AMP

maintains an indexed manifest database, so that clients can retrieve manifests given some media excerpts.

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

AMP Tools and Libraries. The system also provides 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 to check 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. This watermark carries a unique manifest identifier, which may be used to retrieve the original contents and metadata, and can be used to compare them with the transformed media.

Future Work. Several other system components can facilitate efficient interactions with the AMP system including a Provenance Service and a Provenance Browser Extension. We leave these two components as future work.

1) *Provenance Service.* This is a web front-end for additional client tools. The main purpose is to allow provenance discovery, verification and analysis of media that has been processed with AMP.
2) *Provenance Browser Extension.* This is a browser front-end for the client libraries in the form of a browser extension. It allows authentication "at a glance" and the ability to query for detailed information.

B. Prototype Implementation

Initially, AMP has been implemented to run on Linux (Ubuntu LTS 18.04). This decision was motivated, in part, because the CCF framework has been developed and tested on Ubuntu 18.04. The core AMP components were primarily implemented in C# using .NET Core 3.0. This allows us to develop and test these components on both Linux and Windows.

III. 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 IV), optionally distributed by media providers and distributors, and recorded in a complementary Manifest Database (Section VI). 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.

There are two types of manifests: static and streaming. A static manifest contains the cryptographic hash of its associated media object (e.g. a JPEG) or a collection of objects with different encodings (facsimiles). A streaming manifest contains an array of cryptographic hashes corresponding to "chunks" of the associated media. For example, a chunk might correspond to a few seconds of video or audio.

AMP manifests can be used to authenticate the original source material, or the 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 (AMID). It serves as a unique identifier and a commitment for the manifest. AMIDs are also digitally signed by content producers or distributors, and recorded on the ledger. AMP uses X.509 to create all digital signatures and SHA-256 for all cryptographic hashes.

A. Static Manifests

The details of a static manifest are provided in Table I. The publisher assigns an ObjectID to identify a particular media object. In addition, the ObjectID is encoded into the media object as a watermark and may also be inserted into the media's metadata.

The CodecInfo 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 re-compressions 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 value of the OriginAMID field includes one or more AMIDs 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 AMIDs can be recorded in the array. Note that OriginAMID[] is not authoritative on its own: it should only be trusted if the AMID that describes the transform is signed by a trusted authority.

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*.

In addition to fields we have described, we expect additional elements may need to be added, as we identify additional needs from new usage scenarios.

Field	Description
ObjectID	Publisher-assigned identifier for the media object.
PublisherURI	SA URI of a stable, publisher provided location service or a generic URL redirector service.
CodecInfo	String describing the media type (e.g., "JPEG", "MP4").
OriginAMID[]	One or more AMIDs that describe the source media used to create a derived work.
Copyright	Copyright string associated with the media object.
ObjectHash[]	Cryptographic hash of the associated simple media object (or collection of related media objects).

TABLE I
STATIC MANIFEST DESCRIPTION.

Field	Description
ObjectID	Publisher-assigned identifier for the media object.
PublisherURI	A URI of a stable, publisher provided location service or a generic URL redirector service.
CodecInfo	String describing the media type (e.g., "JPEG", "MP4").
OriginAMID[]	One or more AMIDs that describe the source media used to create a derived work.
Copyright	Copyright string associated with the media object.
ChunkInfo[][]	2D Array of data structures describing chunks of a collection or related media streams.

TABLE II
STREAMING MANIFEST DESCRIPTION.

Field	Description
ChunkHash	Cryptographic hash of a chunk.
ChunkStart	Offset pointing to the start of the chunk.
ChunkEnd	Offset pointing to the end of the chunk.

TABLE III
CHUNKINFO DATA STRUCTURE DESCRIPTION.

B. 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 problematic 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.

Table II provides a description of a streaming manifest, which is similar to the static manifest shown in Table I. While a static manifest contains one or more cryptographic hashes of an image or text document in the ObjectHash field, a streaming manifest contains an array of ChunkInfo data structures described in Table III. Each ChunkInfo element contains the cryptographic hash of a chunk, together with an indication of where the chunk starts and ends.

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 ChunkInfo entries in a single manifest. However, if a server is dynamically switching streams, then more than one manifests 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.

IV. MEDIA PROVENANCE LEDGER

AMP utilizes a distributed tamper-proof ledger to ensure consensus on the sequence of published authenticated media based on their cryptographically hashed manifests. Our implementation places several requirements on the ledger service. These requirements are addressed by the Confidential Consortium Framework (CCF) [5].

CCF is a framework for building permissioned confidential applications. AMP uses the framework to build a ledger-based application which is designed to securely store a cryptographic hash and copyright string for each 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 are registered at a given index in the ledger.

a) Permissioned 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., AMID), and the value being a signature computed by the publisher over a concatenation of the AMID and the copyright string (i.e., Copyright in Tables I and II). Once written, these key-value pairs are stored in a Merkle tree, and the Merkle tree is replicated and stored on persistent storage. To ensure that any tampering can be detected, CCF maintains a private key that the service protects and occasionally uses it to sign the Merkle root in the distributed ledger.

b) Receipts: One of the core features that AMP utilizes from CCF is its universally verifiable receipts. These receipts are periodically issued by the CCF service and cover all the committed transactions since the last receipt. They are implemented using digital signatures and auxiliary cryptographic hashes. 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 value of a receipt is that it is possible 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 [1].

c) Governance: CCF provides a flexible governance model. This allows for AMP to define the governance by writing scripts in scripting languages such as the Lua [6] or JavaScript [7] languages. 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.

d) 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 [4], and the variant of BFT implemented by CCF is a modified version of Practical Byzantine Fault Tolerance (PBFT) [3].

Raft leverages trusted execution environments (TEEs) and specifically Intel's SGX. Its trust model is that a single TEE compromise destroys both confidentiality and integrity. By using this trust model, CCF is able to utilize a variant of Raft which can handle malicious attacks as long as Intel's SGX is not compromised.

PBFT is a consensus algorithm that can make progress if less than $\frac{1}{3}$ of the nodes are actively malicious. PBFT's trust model is that a single TEE compromise destroys confidentiality but f+1 compromises (in a 3f+1 network) are required to destroy integrity. This distinction means that even if some of the CCF nodes, which are running in a SGX enclave, are compromised, the Media Provenance Ledger will not lose integrity. This added security comes at an increased performance and latency cost when committing data to the ledger.

Critically, both of these consensus protocols offer finality. This property states that once a transaction has been committed, it cannot be reverted. Furthermore, a CCF receipt provides an additional finality proof.

e) Distributed Execution: CCF utilizes TEEs to ensure that the operator of the Media Provenance Ledger is not able to perform malicious acts on the service. This is designed to provide the AMP consortium members with confidence that they can run the ledger service in a cloud datacenter, and that an operator (such as Azure) cannot compromise the service's confidentiality or integrity.

f) Implementation: CCF is primarily written in C++ although it allows applications to be written in multiple languages. The CCF framework has been developed and tested on Ubuntu 18.04 [2].

AMP will run CCF utilizing its CFT configuration. While SGX has been vulnerable to attacks in the past, Intel has been vigilant in providing mitigations and fixes to prevent these vulnerabilities. This option provides AMP with the best ledger performance while still ensuring integrity of the manifest data that is stored in the ledger.

V. Fragile Watermarking

We have developed fragile watermarking technology and a watermark payload (i.e., the data to be inserted). The purpose of watermarking is 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. Watermarking is an approach to embed metadata within the content, thus preserving such metadata even when the media is slightly edited via tools that may not preserve header-style metadata.

We propose the use of fragile watermarking techniques using a spread-spectrum approach [8]. Using techniques such as spread-spectrum watermarking, it is possible to add low-level pseudo-random 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. Such watermarking patterns will not meaningfully affect machine learning-based analytics performed on the content. The term "fragile" comes from the concept that the watermark will still be detectable after benign edits (typically minor recompression or light cropping), but will not survive major edits on media (for example, the technique of face replacements used commonly in present-day fake media [9], [10]).

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

The watermark payload string, which is inserted into the media item, is described in Table IV and contains the following fields: a media object ID (ObjectID), a publisher URI (PublisherURI), and a signature over these two fields (WatermarkPayloadSignature). AMP does not provide a centralized database containing the PublisherURI and ObjectID. Instead after decoding, the client extracts the payload and submits the ObjectID to the publisher using via PublisherURI. Both the ObjectID and the PublisherURI are specified by the publisher. The PublisherURI is typically a URI for the publisher's web service which is used to located the media by their unique ObjectID.

The watermarking insertion process transforms a media object by embedding a signed watermark before its publication. It should always succeed.

Field	Description
ObjectID	Publisher-assigned identifier for the media object which is the same as in Tables I and II.
PublisherURI	A URI of a stable, publisher provided location service or a generic URL redirector service which is the same as in Tables I and II.
WatermarkPayloadSignature	Signature value over the ObjectID and the PublisherID.

TABLE IV
WATERMARK PAYLOAD.

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 IV. The Watermark Verification Module uses the PublisherURI to obtain a signing certificate. Then, the Watermark Verification Module uses this signing certificate to check the WatermarkPayloadSignature over the ObjectID and PublisherURI. If this cryptographic step succeeds, it finally returns the ObjectID and the PublisherURI back to the client. Once the client has recovered the PublisherURI and the ObjectID, 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.

Watermarking Implementation

The audio watermarking code as been implemented in C. This implementation enables efficient porting to many different processing environments.

Future Work

We leave several items related to watermarking as future research. Watermark decoding is reasonably robust, but depending on the scope of media's modification, the decoded strings may contain some errors. Therefore, we expect to add provisions for higher robustness on the PublisherID field, as well as leverage error-correcting codes for a good balance between payload capacity and decoding robustness.

We plan to explore the use of soft media hashes within the bit string that is carried by the watermark, so that watermark patterns are not transferrable from one piece of media to another.

VI. 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 multimedia standards bodies. In the mean time, a client can use the Manifest Database to map a media object or chunk to a suitable manifest and receipt.

The AMP Manifest Database 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 media ObjectID 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 (ChunkHash).

Media players can quickly and easily extract or calculate the ObjectHash or a ChunkHash 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 Manifest Database the following steps will 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 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 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 ChunkHashes. 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 AMID 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 they 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.

VII. Tools and Governance

There are two parts to the authoring and management back-end. The first part supports the publishing flow. We have developed tools that create a signed manifest (AMP Manifest Creation Tool, AMP Signing Tool), watermark the media (AMP Watermark Tool) and record the manifest on a ledger (AMP Ledger Insertion Tool). The AMP Client Provenance Library can be used by the client application to chunk a video and compute the cryptographic hashes of these chunks. These tools and tool-chains can be used by an ISP, CDN, or another media editing tool to support "authenticated transformations" of an original work, as well as tools that allow authentication information to be added to legacy media (e.g., videos already hosted on YouTube).

The second part of the authoring back-end relates to governance. We use the Microsoft CCF (Confidential Consortium Framework) technology to maintain a ledger of published works and provide a governance model over it. CCF provides a flexible governance model, allowing for a group of members to vote on everything from adding and removing users to updating the CCF service code. If AMP is adopted to provide media provenance, we will collaborate with our media partners at that time to create a governance model. When additional partners join the partnership, we will use CCF to evolve the governance rules as required.

VIII. Example Media Publishing Flow

The purpose and operation of the various AMP components is demonstrated by tracing a typical flow of media through the system. The media publishing flow consists of two phases: publishing and playback. We present below how various AMP Service components can be used during the publishing and playback phases.

Publishing

Assume a content producer generates two media objects: picture.jpg, and video.mp4. The publisher:

1) Uses ffmpeg to convert video.mp4 into a set of re-compressions, video[n].mp4, at various quality levels (e.g., using DASH).
2) Generates a collection of unique ObjectIDs for the objects to be authenticated.
3) Uses the AMP Watermarking Tool to insert encoded versions of the ObjectIDs, their PublisherID and the WatermarkPayloadSignature into the watermark payload of the picture.jpg and all videos that are to be published.
4) Uses the AMP Manifest Creation Tool to create a manifest for the media objects.
5) Uses the AMP Signing Tool to signing the manifest with its publisher's key.
6) Registers the manifest's cryptographic hash and copyright string with the Media Provenance Ledger using the AMP Ledger Insertion Tool.
7) Uploads the manifests to the AMP Manifest Database using the AMP Ledger Insertion Tool.
8) Broadcasts (i.e., stages on a web site, etc.) picture.jpg and video[n].mp4.

Optional step for CDNs, ISPs, etc:

1) CDNs take video[n].mp4 and picture.jpg and create further derived copies using steps 4 through 8 except that these manifests refer back to the original AMP manifest.

Playback

A client application (e.g., browser, media player, etc.):

1) Links to the AMP Client Provenance Library or implements the functionality itself to cryptographically hash a video object's "chunks" or simple media object (e.g., JPEG, text).
2) Consults the AMP Service to obtain a suitable manifest or manifests.
3) Verifies the publisher's signature and the receipt generated by CCF to ensure that the manifest is valid. Successful verification ensures the authenticity of the media.
4) Displays the authentication information (simple or more complex information) if the media is authenticated.
5) Searches for a watermark in the media If a valid manifest is not found in the Manifest Database. Next, attempts to validate the media object based on the PublishedID and ObjectID if the WatermarkPayloadSignature is valid.

IX. Technology Showcases Demos

In this section, we describe two prototypes that we plan to implement to test and showcase the AMP technology.

Media Authentication Web Service. This web service allows a user to upload a video or enter the URL of a publicly

available web video (e.g., YouTube.) The service analyzes the video stream and displays detailed the provenance information.

Browser-Based Media Authentication. This showcase uses the same technology but integrates it into an browser extension (e.g., Chrome, Edge). The browser extension allows users to click through to obtain detailed provenance information, but will also display a simple green/yellow/red banner with publisher information when it is available, indicating that the video is fully authenticated, watermarked, or there is clear evidence of tampering.

X. Performance Evaluation

Media Provenance Ledger

We begin by measuring 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 (AMID) 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 expressed in several 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":
"88c3ba2b25cef698d9ca6775b7fd5c5e8bbc246098a55ad51b8078834c4add44
Copyright (c) Microsoft Corporation. All rights reserved."}}
```

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 every distributed between the east USA and west Europe Azure regions.
3) *Emerging hardware* - A cluster that is running in our own datacenter. All computers are under the same 40G switch, and the computers 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 V. We expect that there will be up to 1 billion entries added to the ledger every day, this results in an expected load of 11,575 operations per second. We can conclude from these results that our 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 the user's experience in media consumption.

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

TABLE V
Media Provenance Ledger throughput and latency.

AMP Service

Next, we estimate the maximum scale requirements for the AMP Service assuming the following parameters:

- 100 publishers
- 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.
- Each original video is transformed into 99 (100-1) variants by the CDN

Using these parameters, this translates into

- 3.7 million original videos/year
- 370 million original and transformed videos/year
- 220 million original chunks/year
- 22 billion 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 ChunkHashes.

XI. AMP Partnership and Standards Strategy

We believe that the proposed AMP media provenance certification and verification system can only be successful if it becomes a widely adopted industry standard. We hope to form a partnership with media organizations and additional technology providers in the near future. We plan to put this collaboration on a formal footing through the formation of an industry alliance similar to the Alliance for Open Media. Other companies can join, either as active contributors or supporters. Such a model can move quickly for ratification of a more detailed design, with the goal of developing reference code and performing sufficient testing to assess the efficiency and performance of the proposed provenance certification and

verification system. Once such a partnership develops more detailed designs, the specifications can be brought to formal media standards committees, such as the ITU or MPEG. That would allow additional improvements to the system design, as appropriate, and would strengthen the goal of fostering industry adoption. A key goal of such an effort should be to promote the development of an open, royalty-free standard, with focus on interoperability. We believe that an open standard will motivate faster adoption. We also believe that increasing trust in media will benefit the business models of all bona fide industry entities involved in the creation and distribution of media.

XII. Discussions

Currently, AMP does not include any components for detecting fake media. If successfully adopted, it will take a number of years before manifests for a large percentage of online media are stored in the ledger. However, we believe that fake media will rapidly improve and become more widely encountered before this time. Therefore, additional fake media detection algorithms will need to be incorporated into the media processing pipeline in the near future. A number of academic and industry efforts are currently underway to improve the detection of deep fakes. We see this work as orthogonal to the provenance solution proposed by AMP, and these detection methods can also be included in the future as part of the the AMP Service.

It is import to note that the purpose of AMP is 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.

XIII. Related Work

Previous related research to the AMP system and effort span three main areas: previously proposed provenance systems, other provenance partnerships, and deep fake detection and content generation.

Provenance Systems. Provenance systems for the prevention of deep fakes is a new and relatively understudied area. The provenance-based system that is most closely related to AMP was recently proposed by Hasan [11]. Like AMP, this system also employs blockchain. However, it is based on the Ethereum blockchain and smart contracts. Since AMP utilizes CCF, it is much more efficient, allowing the speedup of manifest insertion and queries by several orders of magnitude which is required for widespread deployment.

In addition to [11], several startups have proposed provenance-based systems including: Amber and Witness. Amber's technology [12], [13] is aimed at camera manufacturers and adds a cryptographic hash to the video at a user specified rate. Similar to AMP, these hashes are then stored on an Ethereum blockchain.

Similarly, Truepic [14] also provides a photo and verification service where the cryptographic signature is written to a blockchain.

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 [15]. This collaboration is also using a blockchain to provide a provenance solution for media.

The Content Authenticity Initiative is a second partnership with Adobe, The New York Times Company and Twitter [16].

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.

Deep Fake Detection. Deep fake detection is an alternate method to provenance solutions and rely on the algorithmic detection of synthetically generated media. A number of deep fake detection algorithms have been proposed in the literature.

In [18], Li et al. describe their realization that deep fake videos which had been created prior the paper's publication in 2018 often had eyes which failed to blink, which is natural for humans. Thus, they created an eye blink detector and used it as a proxy to detect deep fake videos.

McCloskey and Albright [19] noted that generative adversarial networks (GANs) fail to accurately reproduce colors that are captured naturally by photosensitive cells in a camera's sensor. Their approach to detecting deep fakes is to train a convolutional neural network (CNN) to detect this mismatch in the color.

Face warping artifacts can be introduced during the generation of deep fake videos. Li and Lyu trained a CNN to detect these artifacts to detect some types of deep fake attacks in [20]. Similarly, Yang et al. [21] also trained a CNN to detect inconsistencies in head poses.

In [22], Korshunov and Marcel explore trying to jointly use the audio and video, but their experiments indicated that adding the audio did not help.

In the FaceForensics++ system proposed by Rössler et al. [23], the Xception computer vision object recognition model which also employs CNNs were also used to various types of deep fakes. A leader board of deep fake detection algorithms on the FaceForensics++ dataset can be found at [24].

Content Generation. Generative adversarial networks were originally proposed by Goodfellow, et al. [25]. Several important works [26], [27] have investigated using GANs for large-scale, synthetic image generation. Recent research in GANs has enabled talking head models to be quickly adapted with just a few frames [28].

Popular face swap algorithms include Deepfakes [9] and FaceSwap [10]. Facial expressions can be transferred from one person to a person in a video in real-time using the Face2Face algorithm [29].

XIV. Conclusions

Increasingly sophisticated methods for synthesizing media, coupled with the wide reach of social media, has become a

threat to private and public institutions as well as to individuals. Fake media has the potential to significantly undermine trust in media and journalism, threatening the foundations of democracy. We will not be able to rely on algorithmic detection of deep fake media in the long-term. Provenance solutions will be required to properly authenticate the source and veracity of media.

To this end, we have proposed and constructed a prototype of the AMP system. AMP allows trusted content providers to form one or more consortiums that allow applications such as a media player or a browser to provide an indication to users that the content they are viewing has been verified to come from the purported source. Beyond the core security pipeline, human factors and design will play an important role. Inspired by the TLS lock icon, indicators can be provided by the browser or media player to alert users that the transmitted content can be traced back to its original source.

In order for a provenance solution to be successfully adopted, such a system must be formally adopted by a recognized standards body. We are seeking such coordination and adoption for the AMP system or a variant that provides similar functionality. We also believe that it is important to open source the code for a widely used provenance system. We plan to open source the AMP system in near future to facilitate its widespread adoption.

REFERENCES

[1] Microsoft, "Ccf: A framework for building confidential verifiable replicated services," https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf, 2019.

[2] ——, "Ccf documentation," https://microsoft.github.io/CCF/, 2019.

[3] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[4] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference (ATC)*, 2014.

[5] Microsoft, "Confidential consortium framework," https://github.com/Microsoft/CCF, 2019.

[6] Lua.org, "Lua - the programming language," https://theblog.adobe.com/content-authenticity-initiative/.

[7] ——, "ready to try javascript," https://www.javascript.com/.

[8] R. Malvar and D. Florencio, "Improved spread spectrum: A new modulation technique for robust watermarking," *IEEE Transactions on Signal Processing*, April 2003. [Online]. Available: https://www.microsoft.com/en-us/research/publication/improved-spread-spectrum-a-new-modulation-technique-for-robust-watermarking/

[9] "faceswap: Faceswap is a tool that utilizes deep learning to recognize and swap faces in pictures and videos," https://github.com/deepfakes/faceswap/.

[10] M. Kowalski, "Faceswap," https://github.com/MarekKowalski/FaceSwap/.

[11] H. R. Hasan and K. Salah, "Combating deepfake videos using blockchain and smart contracts," in *IEEE Access*, 2019, pp. 41 596–41 606.

[12] Amber, "Instilling trust into video," https://app.ambervideo.co/, 2019.

[13] L. H. Newman, "A new tool protects videos from deepfakes and tampering," https://www.wired.com/story/amber-authenticate-video-validation-blockchain-tampering-deepfakes/, 2019.

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

[15] "The news provenance project," https://www.newsprovenanceproject.com/, 2019.

[16] Adobe, "Introducing the content authenticity initiative," https://theblog.adobe.com/content-authenticity-initiative/, 2019.

[17] D. Kayyali, "Proofmode - verified visuals enabled!" https://blog.witness.org/2017/04/proofmode-helping-prove-human-rights-abuses-world/, 2017.

[18] Y. Li, M.-C. Chang, and S. Lyu, "In ictu oculi: Exposing ai generated fake face videos by detecting eye blinking," in *IEEE Workshop on Information Forensics and Security (WIFS)*, 2018.

[19] S. McCloskey and M. Albright, "Detecting gan-generated imagery using color cues," *arXiv preprint arXiv:1812.08247*, 2018.

[20] Y. Li and S. Lyu, "Exposing deepfake videos by detecting face warping artifacts," in *Workshop on Media Forensics*, 2019.

[21] X. Yang, Y. Li, and S. Lyu, "Exposing deep fakes using inconsistent head poses," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019, pp. 8261–8265.

[22] P. Korshunov and S. Marcel, "Deepfakes: a new threat to face recognition? assessment and detection," *arXiv preprint arXiv:1812.08685*, 2018.

[23] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics++: Learning to detect manipulated facial images," in *ICCV 2019*, 2019.

[24] T. V. C. Group, "Faceforensics benchmark," http://kaldir.vc.in.tum.de/faceforensics_benchmark//, 2019.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, , and A. C. Y. Bengio, "Generative adversarial nets," in *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[26] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *International Conference on Learning Representations(ICLR)*, 2018.

[27] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," in *International Conference on Learning Representations(ICLR)*, 2019.

[28] E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky, "Few-shot adversarial learning of realistic neural talking head models," *arXiv preprint arXiv:1905.08233v2*, 2019.

[29] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner, "Face2face: Real-time face capture and reenactment of rgb videos," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2016.