MAV PDF Conversion & Watermarking Architecture

Status: Ready for Review

Contributors: Jayman Dalal, Zexing Song

Context

The high-level goals for the Microsoft Audit Vault are:

- Ensure confidential evidence is properly managed and shared securely in an interface accessible by external auditors.
- To provide a portal that External Auditors can access and view, but with restricted and monitored ability to download evidence.
- Allow auditors to request evidence and provide feedback on evidence submitted by the compliance PM.
- Compliance PM can streamline communication and collaboration with external auditors
- To ensure legal compliance with the FBA examination and long-term evidence retention (minimum 10 years).

See FBA Tool Functional Spec.docx

This design document focuses on the backend infrastructure for core capability of the service around the document lifecycle management of the evidence – uploading, storing, conversion, watermarking and securing, viewing and downloading of the evidence documents.

Requirements

Functional

1. Any document (pdf, word, png, jpg) uploaded by Compliance PM marked as "Evidence" should be converted (if required) and stored as a PDF.
2. The PDF document must be watermarked with fixed disclaimer text while stored at rest.
3. The PDF must have a dynamic/configurable custom text added as watermark before downloading and should be tamper proof (text & contents cannot be copied from any page of the PDF).
4. The source document files can be up to 200MB in size.
5. The final watermarked PDF file available for viewing in a PDF viewer of the web client of the External Auditor or Compliance PM

Non-Functional Requirements

1. Original document must be kept securely after conversion to intermediate PDF.
2. PDF conversion should be performant, robust and stateless.
3. The PDF document should be stored at rest with the highest level of encryption - AES 256, as well as use HTTP in transit.
4. The final watermarked PDF should be stored with limited privileges and have limited TTL to view from the viewer. If required, the token can be refreshed to extend the TTL.
5. Only authenticated + authorized users should be able to view the final watermarked PDF.
6. Service must prioritize consistency and durability over availability. Every effort must be made to increase the reliability to 99.99+.
7. Low Latency - Time to have Evidence PDF ready to view in client < 5s

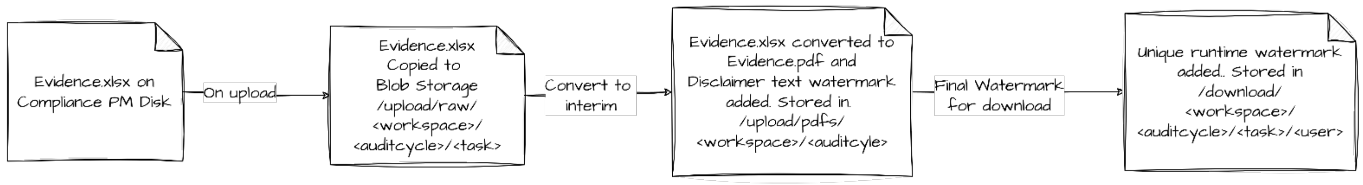
Scale

Given requirements below, we are expecting to process about approx 5K documents/year (~2-5 documents/hour upload and ~10-20 documents/hour for viewing at peak audit cycle) thru this system or about 100GB of data/year. Assuming the requirements grow at 20% YOY over next 10 years, we are looking about ~3TB of storage. Core volume requirements:

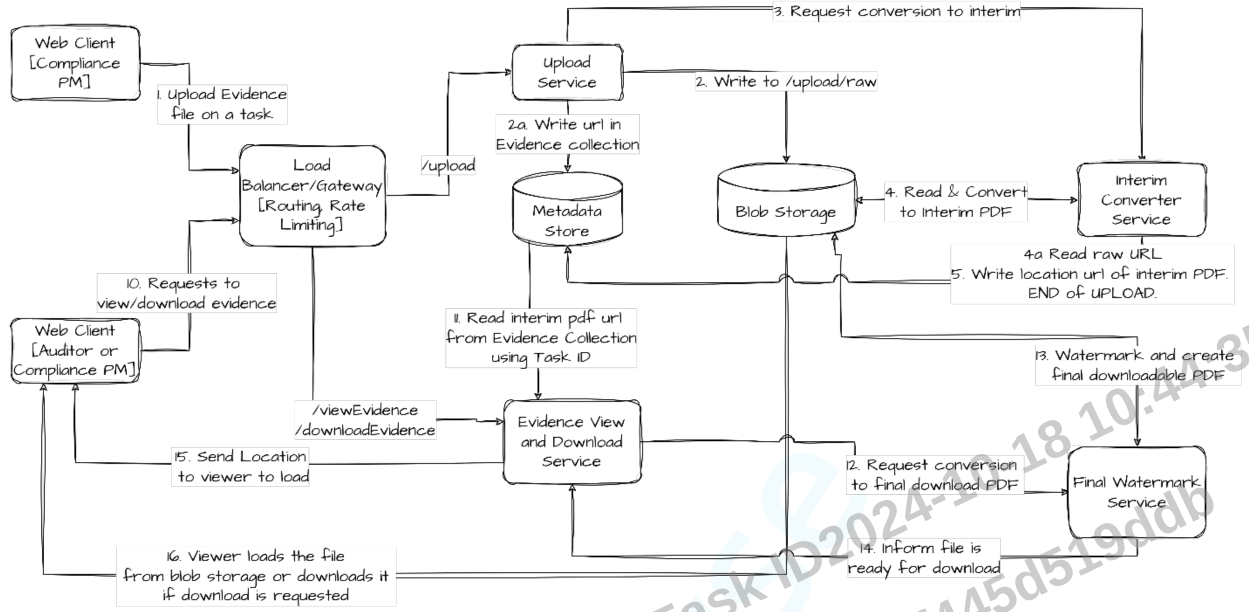
- Approximately 4 audit cycles per year
- 2-4 domains per cycle
- 100 tasks per domain
- 4-5 evidences per task
- 200MB per evidence file max

High Level Design

Evidence Document State Transition


Evidence.xlsx on Compliance PM Disk
On upload
Evidence.xlsx Copied to Blob Storage /upload/raw/ <workspace>/ <auditcycle>/<task>
Convert to interim
Evidence.xlsx converted to Evidence.pdf and Disclaimer text watermark added. Stored in /upload/pdfs/ <workspace>/<auditcyle>
Final Watermark for download
Unique runtime watermark added. Stored in /download/ <workspace>/ <auditcycle>/<task>/<user>

Data Flow


Web Client [Compliance PM]
1. Upload Evidence file on a task
Load Balancer/Gateway [Routing, Rate Limiting]
/upload
Upload Service
3. Request conversion to interim
2. Write to /upload/raw
2a. Write url in Evidence collection
Metadata Store
Blob Storage
4. Read & Convert to Interim PDF
Interim Converter Service
4a Read raw URL
5. Write location url of interim PDF. END of UPLOAD.
10. Requests to view/download evidence
Web Client [Auditor or Compliance PM]
11. Read interim pdf url from Evidence Collection using Task ID
13. Watermark and create final downloadable PDF
/viewEvidence /downloadEvidence
Evidence View and Download Service
15. Send Location to viewer to load
12. Request conversion to final download PDF
Final Watermark Service
14. Inform file is ready for download
16. Viewer loads the file from blob storage or downloads it if download is requested

Core Entities

EvidenceDocument

- evidenceDocumentId
- uploadedBy
- uploadTime
- taskId
- sourceDocumentName
- rawStorageUrl
- interimStorageUrl
- rawFileStatus: UploadInProgress | uploadCompleted | Errored
- interimFileStatus: ConversionStarted | ConversionCompleted | Errored

SecureEvidenceDocumentFile

- secureEvidenceDocumentId
- evidenceDocumentId
- userId – the user who is requesting the view on the document
- requestTime
- finalStorageUrl

EvidenceDocumentAudit

- evidenceDocumentAuditId
- requestedByUserId
- requestTime
- documentId – (must be one of evidenceDocumentId or secureEvidenceDocumentId)
- auditType: Upload | View | Download | TokenRefresh

APIs

POST /requestUpload

Header: JWT Token for user

Params

- file: IFormFile
- taskId

Returns

- Status: 202 Accepted
- evidenceDocumentId

Notes:

- To support large files, we would need to configure ASP.NET Core. See Upload files in ASP.NET Core | Microsoft Learn and Uploading Large Files in ASP.NET Core - Code Maze (code-maze.com).
- EvidenceDocument should now have an entry for this document that is about to be uploaded now. rawFileStatus=UploadInProgress
- EvidenceDocumentAudit should now have a new entry added to track this request from user with status=Upload and documentId= evidenceDocumentId

GET /uploadStatus

Header: JWT Token for user

Params

- evidenceDocumentId

Returns

- Status: 200 OK
- Status: Completed | Errored | Unknown

Notes

- EvidenceDocument for the matching evidenceDocumentId should have rawFileStatus=UploadCompleted or UploadErrored

GET /viewEvidence

Header: JWT Token for user

Params

- evidenceDocumentId

Returns

- Status: 200 OK
- secureEvidenceDocumentFileId
- sasToken+blobURL for final watermarked file

Notes

- SecureEvidenceDocumentFile will have an entry to track the generation of the final interim file using from this user.
- EvidenceDocumentAudit should now have a new entry added to track this request from user with status=View and documentId= secureEvidenceDocumentFileId

GET /refreshSasToken

Header: JWT Token for user

Params

- secureEvidenceDocumentFileId

Returns

- Status: 200 OK

Notes

- EvidenceDocumentAudit should now have a new entry added to track this request from user with status=RefreshToken and documentId= secureEvidenceDocumentFileId
- Open Issue: Do we want to provide capabilities to revoke the access to this file at some point so we can block the token refresh?

GET /downloadEvidence

Header: JWT Token for user

Params

- secureEvidenceDocumentFileId

Returns

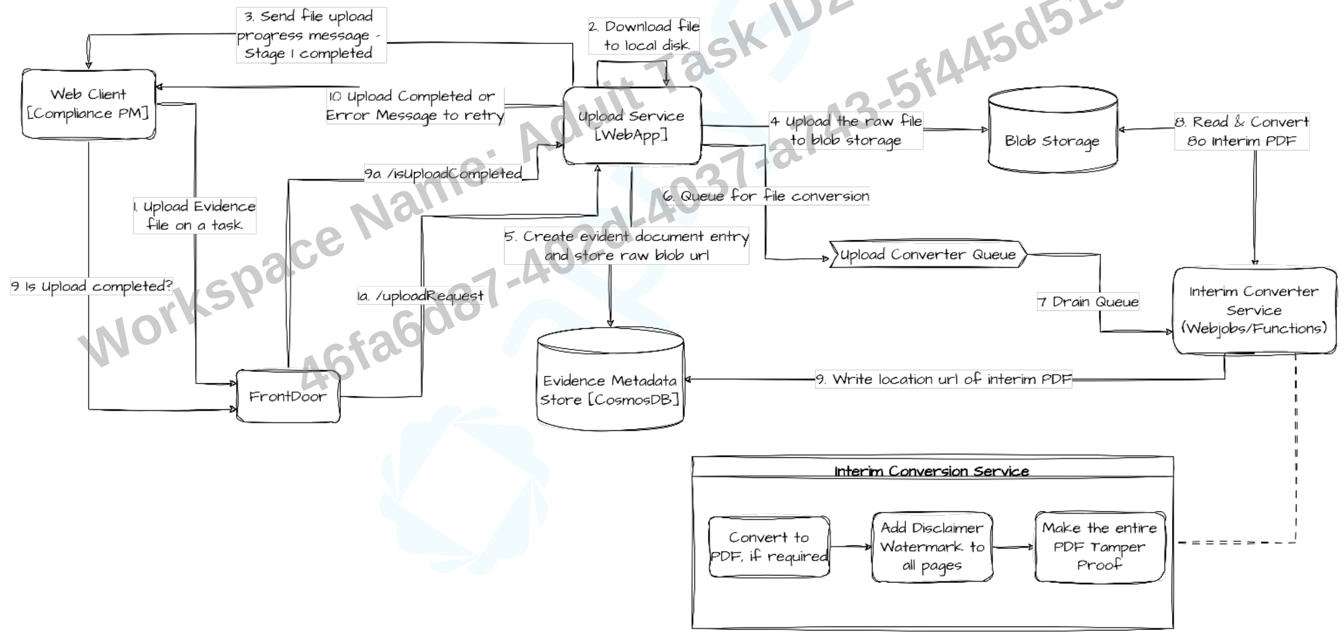
- Status: 200 OK
- Pdf file to download and save to the PC

Notes

- `Assumption: The evidence should have been viewed before the download has been requested. We should throw error if that were to happen.`
- `EvidenceDocumentAudit should now have a new entry added to track this request from user with status=Download and documentId= secureEvidenceDocumentFileId`

Detailed Design

Upload Path


3. Send file upload progress message - Stage 1 completed
2. Download file to local disk
Web Client [Compliance PM]
10 Upload Completed or Error Message to retry
Upload Service [WebApp]
4 Upload the raw file to blob storage
Blob Storage
8. Read & Convert 8o Interim PDF
9a /isUploadCompleted
1. Upload Evidence file on a task
6. Queue for file conversion
5. Create evident document entry and store raw blob url
Upload Converter Queue
9 Is Upload completed?
1a. /uploadRequest
7 Drain Queue
Interim Converter Service (Webjobs/Functions)
FrontDoor
Evidence Metadata Store [CosmosDB]
9. Write location url of interim PDF
Interim Conversion Service
Convert to PDF, if required
Add Disclaimer Watermark to all pages
Make the entire PDF Tamper Proof

Config

- Storage should be configured to have Microsoft Defender scanning enabled by default. See [Protect your storage accounts with the Microsoft Defender for Storage plan - Microsoft Defender for Cloud | Microsoft Learn].

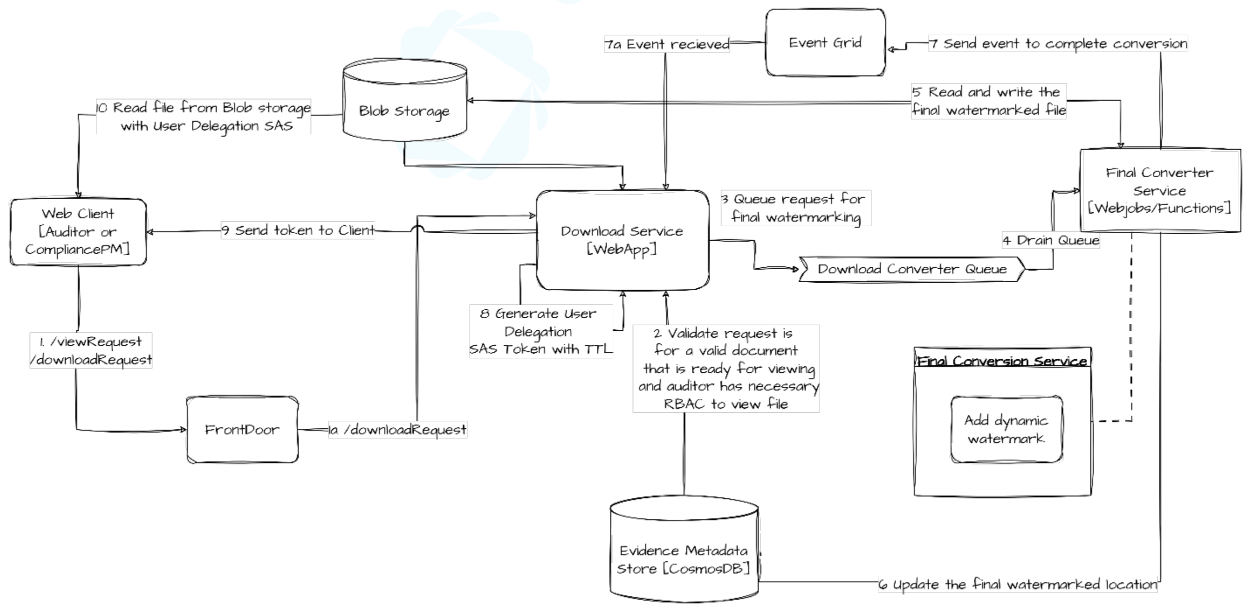
Steps

1. Authenticated Compliance PM makes a request to upload (/uploadRequest) an evidence (say Evidence.docx) using web-client. The request bears the OAuth EntraID token of the

authenticated user. Request comes to the Frontdoor and is routed to the Upload service. This is the EvidenceUploadController in AuditVault webapp.

2. EvidenceUploadController validates the token on the request with EntraID. It downloads the file and copies to local disk storage.
3. It returns message to client that stage 1 upload is complete
4. It copies the file from local disk to blob storage location /upload/raw/workspace/auditcyle/task.
5. It creates an entry in Evidence Metadata collection in CosmosDB and updates the location of blob url there
    1. EvidenceDocument should now have an entry for this document that is about to be uploaded now. rawFileStatus=UploadInProgress
    2. EvidenceDocumentAudit should now have a new entry added to track this request from user with status=Upload and documentId= evidenceDocumentId
6. It queues up the document for conversion.
    1. At this point the file upload is persisted successfully and status check for Is upload completed can return success.
7. Queue is drained by the Interim Converter Azure Function. The entire document JSON has the needed information (storage url location, etc) to do the conversion.
8. The Azure Function does the conversion and watermarking and writes the file back to the blob store to correct container location.
9. Meanwhile the client keeps polling (for now this is cheap) every couple seconds on /isUploadCompleted.
10. If we have completed upto Step 6, then success is returned, otherwise we return an error and client has to try prompting to upload again.
    1. EvidenceDocument for the matching evidenceDocumentId should have rawFileStatus=UploadCompleted or UploadErrored

Evidence View and Download Path


Event Grid
7a Event recieved
7 Send event to complete conversion
Blob Storage
10 Read file from Blob storage with User Delegation SAS
5 Read and write the final watermarked file
Final Converter Service [Webjobs/Functions]
Web Client [Auditor or CompliancePM]
9 Send token to Client
Download Service [WebApp]
3 Queue request for final watermarking
4 Drain Queue
Download Converter Queue
8 Generate User Delegation SAS Token with TTL
2 Validate request is for a valid document that is ready for viewing and auditor has necessary RBAC to view file
1. /viewRequest /downloadRequest
FrontDoor
1a /downloadRequest
Final Conversion Service
Add dynamic watermark
Evidence Metadata Store [CosmosDB]
6 Update the final watermarked location

Steps

1. Authenticated and authorized Auditor makes a request to view the evidence on the task. The request bears the OAuth EntraID token of the authenticated user. Request comes to the Frontdoor and is routed to the Download service.
2. Validate that the request is for a valid Task and Evidence Document and the auditor has been authorized to view the file.
3. Add a request to the queue for downloading. The payload contains the Blob url and some user guid related information that we can use to watermark.
    1. SecureEvidenceDocumentFile will have an entry to track the generation of the final interim file using from this user.
    2. EvidenceDocumentAudit should now have a new entry added to track this request from user with status=View or Download and documentId= secureEvidenceDocumentFileId
4. The Final Converter Service is Azure Function that drains the queue periodically on a durable task. It opens the payload and extracts the blob url where the interim file is to convert the file.
5. The file is stamped with final watermark content (user id guid, timestamp, etc) and uploaded to the /download/<workspace>/<auditcycle>/<task>/<user> folder.
    1. If the SecureEvidenceDocumentFile file has already been generated before for this exact user on the same task, then just reuse the same file and skip regeneration.
6. An event is sent via EventGrid to the Download Service with the location to final url and which user it is for.
7. Using the user token on the original request Generate User Delegation SAS token (Create a user delegation SAS - Azure Storage | Microsoft Learn) locally in the webapp. (TBD, we may have to request the a refresh on the token from frontend, in case the request has expired).
    1. Open question: How long do we want to issue the TTL for delegation sas token. Maybe start with 30mins and then refresh as needed.
8. Send the token to the client.
9. The auditor client opens the pdf file viewer and downloads the file from the blob storage with the token.
    1. If the TTL on the SAS Token expires by the time we get the file ready for viewing, we might have to add an endpoint to refresh the token.
    2. All the auditor viewing activities should be logged to EvidenceDocumentAudit, ensuring that we are compliant with not logging any customer content, etc.

Open Questions

1. What if user requests to view the same file again say within some time period like 30mins? Do we generate new document?
2. Engg design for frontend client for evidence pdf viewing?
    1. Some sample code provided below.

PDF Library Evaluation

We have been analyzing two libraries for doing PDF conversion, watermarking and making the PDF tamer proof:

1. https://help.syncfusion.com/document-processing/pdf/pdf-library/net/overview
2. Server/Desktop docx, ppt, excel to PDF Conversion Library | Apryse Documentation

Dimensions for Evaluation

- Performance of conversion to PDF from Office and Image files.
- Equality of how the PDF file looks post conversion.
- Performance of watermarking.
- Position of watermarking using text and look/feel of watermark'ed document.
- Ensuring tamper-proofness of the watermarking process.
- Making the entire file tamper-proof thru Rasterization or File Permissions.
- Other cool features that the SDK supports – Redaction, etc.
- Offline licensing validation.
- Technical support quality
- Pricing

Analysis Summary

- Apryse is the choice SDK. It has a more comprehensive feature set, very performant and has the necessary secure output that we need.
- Apryse also sets us up for success in future for MAV as we evolve it into a full fledged document workflow with support for redaction, client viewer, etc.
- Apryse is 2-3X more expensive per year [~$10-15K] at the volume we are looking to have over the next year compared to SyncFusion [$4-5K].

Detailed Analysis

Based on the demand for evidence manipulation, Apryse performance testing has been conducted based on the following categories:

- MS Office Format (Word, Excel, PowerPoint) and Image format (PNG, JPG) convert to PDF format
- ~~Watermark Process~~ (Instant Processed, No need Test)
- ~~Password Access Control (Temper-Proof)~~ (Instant Processed, No need Test)
- PDF Rasterization (Temper-Proof)
  - CRUD Method
  - Memory Stream Method

In MVP-0, Microsoft Audit Vault supports the customers updating 6 types of data: PDF, JPG, PNG, DOCX, XLSX and PPTX. In Apryse, it supports converting all MS Office products to PDF format via the API: "*pdftron.PDF.Convert.OfficeToPDF*" and image to PDF format via the API: *pdftron.PDF.Convert.ToPdf*".

The chart below is the Conversion performance testing for Apryse Commercial license Library:

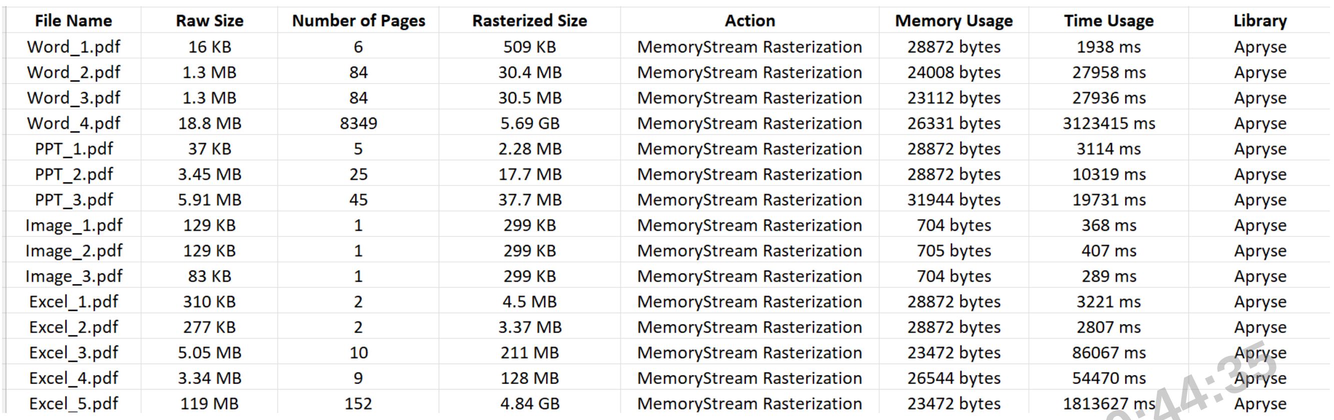
File Name	Raw Size	Generated Size	Action	Memory Usage	Time Usage	Library
Word_1.docx	15 KB	16 KB	Convert To PDF	3728 bytes	114 ms	Apryse
Word_2.docx	1.1 MB	1.3 MB	Convert To PDF	34920 bytes	3399 ms	Apryse
Word_3.docx	1.14 MB	1.3 MB	Convert To PDF	34920 bytes	3277 ms	Apryse
Word_4.docx	3.26 MB	18.8 MB	Convert To PDF	34920 bytes	7566 ms	Apryse
PPT_1.pptx	55.8 KB	37 KB	Convert To PDF	2728 bytes	105 ms	Apryse
PPT_2.pptx	3.77 MB	3.45 MB	Convert To PDF	34920 bytes	1723 ms	Apryse
PPT_3.pptx	8.27 MB	5.91 MB	Convert To PDF	31848 bytes	2464 ms	Apryse
Image_1.png	197 KB	129 KB	Convert To PDF	3768 bytes	138 ms	Apryse
Image_2.jpg	197 KB	129 KB	Convert To PDF	1768 bytes	150 ms	Apryse
Image_3.png	97 KB	83 KB	Convert To PDF	3768 bytes	131 ms	Apryse
Excel_1.xlsx	281 KB	310 KB	Convert To PDF	3728 bytes	334 ms	Apryse
Excel_2.xlsx	278 KB	277 KB	Convert To PDF	3728 bytes	252 ms	Apryse
Excel_3.xlsx	1.02 MB	5.05 MB	Convert To PDF	31848 bytes	13312 ms	Apryse
Excel_4.xlsx	1.08 MB	3.34 MB	Convert To PDF	34920 bytes	8133 ms	Apryse
Excel_5.xlsx	13.7 MB	119 MB	Convert To PDF	29520 bytes	327326 ms	Apryse
Excel_6.xlsx	82.8 MB	428 MB	Convert To PDF	150454 bytes	929222 ms	Apryse
PDF_1.pdf	16 KB	N/A	Convert To PDF	N/A	N/A	Apryse
PDF_2.pdf	283 KB	N/A	Convert To PDF	N/A	N/A	Apryse
PDF_3.pdf	434 KB	N/A	Convert To PDF	N/A	N/A	Apryse
PDF_4.pdf	771 KB	N/A	Convert To PDF	N/A	N/A	Apryse

In enabling the Temper-Proof for evidence, Rasterization (Image-based PDF) is one of the options powered by Apryse library. Two general coding methods are available now: either CRUD or Memory Stream. In CRUD way, we create a temporary image for each pdf page and save it into Blob and fetch the temp file injected into a pdf template, then delete the temporary image immediately from Blob. The other way is using memory stream such that no CRUD action is needed in Blob level, all the temp files will be processed in memory (RAM). According to Azure Instance Microsoft Audit Vault subscripted, the Instance is 1.75 GB RAM, it will be sufficient to hand up to 160 MB evidence updated by customer based on the load testing.

The chart below is the CRUD Rasterization performance testing for Apryse Commercial license Library:

File Name	Raw Size	Number of Pages	Rasterized Size	Action	Memory Usage	Time Usage	Library
Word_1.pdf	16 KB	6	509 KB	Normal Rasterization	28264 bytes	1961 ms	Apryse
Word_2.pdf	1.3 MB	84	30.4 MB	Normal Rasterization	23344 bytes	30436 ms	Apryse
Word_3.pdf	1.3 MB	84	30.5 MB	Normal Rasterization	23408 bytes	30506 ms	Apryse
Word_4.pdf	18.8 MB	8349	5.69 GB	Normal Rasterization	23376 bytes	3560650 ms	Apryse
PPT_1.pdf	37 KB	5	2.28 MB	Normal Rasterization	28264 bytes	3237 ms	Apryse
PPT_2.pdf	3.45 MB	25	17.7 MB	Normal Rasterization	28272 bytes	11568 ms	Apryse
PPT_3.pdf	5.91 MB	45	37.7 MB	Normal Rasterization	23344 bytes	23159 ms	Apryse
Image_1.pdf	129 KB	1	299 KB	Normal Rasterization	520 bytes	436 ms	Apryse
Image_2.pdf	129 KB	1	299 KB	Normal Rasterization	520 bytes	434 ms	Apryse
Image_3.pdf	83 KB	1	299 KB	Normal Rasterization	520 bytes	435 ms	Apryse
Excel_1.pdf	310 KB	2	4.5 MB	Normal Rasterization	28240 bytes	3436 ms	Apryse
Excel_2.pdf	277 KB	2	3.37 MB	Normal Rasterization	28240 bytes	2936 ms	Apryse
Excel_3.pdf	5.05 MB	10	211 MB	Normal Rasterization	23408 bytes	94962 ms	Apryse
Excel_4.pdf	3.34 MB	9	128 MB	Normal Rasterization	23400 bytes	60394 ms	Apryse
Excel_5.pdf	119 MB	152	4.84 GB	Normal Rasterization	26480 bytes	2001756 ms	Apryse

The chart below is the Memory Stream Rasterization performance testing for Apryse Commercial license Library:



File Name	Raw Size	Number of Pages	Rasterized Size	Action	Memory Usage	Time Usage	Library
Word_1.pdf	16 KB	6	509 KB	MemoryStream Rasterization	28872 bytes	1938 ms	Apryse
Word_2.pdf	1.3 MB	84	30.4 MB	MemoryStream Rasterization	24008 bytes	27958 ms	Apryse
Word_3.pdf	1.3 MB	84	30.5 MB	MemoryStream Rasterization	23112 bytes	27936 ms	Apryse
Word_4.pdf	18.8 MB	8349	5.69 GB	MemoryStream Rasterization	26331 bytes	3123415 ms	Apryse
PPT_1.pdf	37 KB	5	2.28 MB	MemoryStream Rasterization	28872 bytes	3114 ms	Apryse
PPT_2.pdf	3.45 MB	25	17.7 MB	MemoryStream Rasterization	28872 bytes	10319 ms	Apryse
PPT_3.pdf	5.91 MB	45	37.7 MB	MemoryStream Rasterization	31944 bytes	19731 ms	Apryse
Image_1.pdf	129 KB	1	299 KB	MemoryStream Rasterization	704 bytes	368 ms	Apryse
Image_2.pdf	129 KB	1	299 KB	MemoryStream Rasterization	705 bytes	407 ms	Apryse
Image_3.pdf	83 KB	1	299 KB	MemoryStream Rasterization	704 bytes	289 ms	Apryse
Excel_1.pdf	310 KB	2	4.5 MB	MemoryStream Rasterization	28872 bytes	3221 ms	Apryse
Excel_2.pdf	277 KB	2	3.37 MB	MemoryStream Rasterization	28872 bytes	2807 ms	Apryse
Excel_3.pdf	5.05 MB	10	211 MB	MemoryStream Rasterization	23472 bytes	86067 ms	Apryse
Excel_4.pdf	3.34 MB	9	128 MB	MemoryStream Rasterization	26544 bytes	54470 ms	Apryse
Excel_5.pdf	119 MB	152	4.84 GB	MemoryStream Rasterization	23472 bytes	1813627 ms	Apryse

Next Steps

#	Task	Owner	Status
1	POC: Evaluate how Microsoft Conversion Service works? Questions: 1. Does it support Word, Excel, PNG and JPG files? 2. Does the service persist the file after conversion or it deletes it? 3. How long does a typical 30MB file takes?	Zexing	Abandoned
2	POC: Evaluate libraries to convert to PDF from Word, Excel, PNG and JPG. Question: 3. Two libs to evaluate (maybe there are more): 1. https://help.syncfusion.com/document-processing/pdf/pdf-library/net/overview 2. Server/Desktop docx, ppt, excel to PDF Conversion Library | Apryse Documentation 4. What source formats are supported? 5. Also does the library support watermarking? 6. How long does a conversion typically take? Profile CPU/Memory on conversion.	Zexing	Complete
3	POC: Upload Path POC. Steps 1. Provision Azure Queue, Azure Function, Azure Blob Storage. 2. Add a simple controller EvidenceUploadController as describe in upload flow 3. Build a simple orchestration to upload a file and then simulate that it goes thru the conversion.	Zexing	In progress

4	Costing: Investigate costing for different libraries for PDF conversion and watermarking. Question: 1. What is the commercial license model? 2. Do you support conversion from word, excel, png and jpg to PDF? 3. Do you support watermarking? 4. Do you support making the PDF tamer proof?	Jayman	Complete
5	Bring up ViewEvidence Path		
6	Integrate Evidence viewing with rest of the MAV and testing		

Sample Code

Disclaimer: This is sample code that has not been tested. The goal is to demonstrate how the functionality will work e2e between client and backend. Please do the necessary checks.

Upload Evidence

Backend (C# ASP.NET Core)

```
using System.Threading.Tasks;
using Azure.Storage.Blobs;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace Microsoft.AuditVault.Core.Services;

public class UploadEvidenceService(string connectionString)
{
    private readonly BlobServiceClient _blobServiceClient = new
BlobServiceClient(connectionString);

    public async Task<string> UploadFileAsync(IFormFile file, string containerName)
    {
        var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
        await containerClient.CreateIfNotExistsAsync().ConfigureAwait(false);
```

```
            var blobClient = containerClient.GetBlobClient(file.FileName);

            using (var stream = file.OpenReadStream())
            {
                await blobClient.UploadAsync(stream, true).ConfigureAwait(false);
            }
            return blobClient.Uri.ToString();
        }
    }

    [ApiController]
    [Route("api/[controller]")]
    public class UploadEvidenceController(UploadEvidenceService uploadEvidenceService) : ControllerBase
    {
        private readonly UploadEvidenceService _uploadEvidenceService = uploadEvidenceService;

        [HttpPost("upload")]
        public async Task<IActionResult> UploadFile([FromForm] IFormFile file)
        {
            if (file == null || file.Length == 0)
                return BadRequest("File is empty");

            var containerName = "your-container-name";
            var fileUrl = await _uploadEvidenceService.UploadFileAsync(file, containerName);

            return Ok(new { fileUrl });
        }
    }
```

Frontend (React)

```
import React, { useState } from 'react';
import axios from 'axios';

const FileUpload = () => {
  const [file, setFile] = useState(null);
  const [message, setMessage] = useState('');

  const handleFileChange = (e) => {
    setFile(e.target.files[0]);
  };

  const handleFileUpload = async () => {
    if (!file) {
      setMessage('Please select a file to upload');
      return;
    }

    const formData = new FormData();
    formData.append('file', file);

    try {
      const response = await axios.post('/api/blob/upload', formData, {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      });
      setMessage(`File uploaded successfully: ${response.data.fileUrl}`);
    } catch (error) {
      console.error('Error uploading file', error);
      setMessage('Error uploading file');
    }
  };

  return (
    <div>
    <input type= "file" onChange = { handleFileChange } />
      <button onClick={ handleFileUpload }> Upload </button>
  { message && <p>{ message } </p> }
  </div>
    );
};

export default FileUpload;
```

Viewing and Downloading Evidence

Backend (C# ASP.NET Core)

```
using System;
using System.IO;
using System.Threading.Tasks;
using Azure.Identity;
using Azure.Storage.Blobs;
using Azure.Storage.Sas;
using Microsoft.AspNetCore.Mvc;

namespace Microsoft.AuditVault.Core.Services;

public class EvidenceService(string storageAccountUri)
{
    private readonly BlobServiceClient _blobServiceClient = new BlobServiceClient(new
Uri(storageAccountUri), new DefaultAzureCredential());

    public string GenerateUserDelegationSasToken(string containerName, string blobName)
    {
        var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
        var blobClient = containerClient.GetBlobClient(blobName);

        var userDelegationKey = _blobServiceClient.GetUserDelegationKey(DateTimeOffset.UtcNow,
DateTimeOffset.UtcNow.AddHours(1));

        var sasBuilder = new BlobSasBuilder
        {
            BlobContainerName = containerName,
            BlobName = blobName,
            Resource = "b",
            ExpiresOn = DateTimeOffset.UtcNow.AddHours(1)
        };
        sasBuilder.SetPermissions(BlobSasPermissions.Read);

        var sasToken = sasBuilder.ToSasQueryParameters(userDelegationKey,
_blobServiceClient.AccountName).ToString();

        return $"{blobClient.Uri}?{sasToken}";
    }

    public async Task<Stream> GetEvidenceStreamForDownloadAsync(string containerName, string
blobName)
    {
        var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
        var blobClient = containerClient.GetBlobClient(blobName);
        var sasToken = GenerateUserDelegationSasToken(containerName, blobName);
        var stream = await blobClient.OpenReadAsync();

        return stream;
    }
}

[ApiController]
[Route("api/[controller]")]
public class EvidenceController(EvidenceService evidenceService) : ControllerBase
{
    private readonly EvidenceService _evidenceService = evidenceService;


    [HttpGet("sas-token-url")]
    public IActionResult GetEvidenceSasTokenUrl(string containerName, string blobName)
    {
        var sasToken = _evidenceService.GenerateUserDelegationSasToken(containerName, blobName);
        return Ok(new { sasToken });
    }

    [HttpGet("download-pdf")]
    public async Task<IActionResult> DownloadEvidenceAsync(string containerName, string blobName)
    {
        var stream = await _evidenceService.GetEvidenceStreamForDownloadAsync(containerName,
blobName);
        return File(stream, "application/pdf", "file.pdf");
    }
}
```

Frontend (React)

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const PdfViewer = ({ containerName, blobName }) => {
  const [pdfUrl, setPdfUrl] = useState('');

  useEffect(() => {
    const fetchSasToken = async () => {
      try {
        const response = await axios.get(`/api/blob/sas-
token?containerName=${containerName}&blobName=${blobName}`);
        setPdfUrl(response.data.sasToken);
      } catch (error) {
        console.error('Error fetching SAS token', error);
      }
    };

    fetchSasToken();
  }, [containerName, blobName]);

  return (
    <div>
    {
      pdfUrl?(
                   <iframe src = { pdfUrl } width = "100%" height = "600px" title = "PDF Viewer" >
</iframe>
      ): (
          <p>Loading...</p>
               )
}
</div>
       );
   };

export default PdfViewer;
```

```
const DownloadButton = () => {
  const handleDownload = async () => {
    try {
      const response = await fetch('https://your-backend-url/api/download-pdf');
      const blob = await response.blob();
      const url = window.URL.createObjectURL(blob);
      const a = document.createElement('a');
      a.href = url;
      a.download = 'file.pdf';
      document.body.appendChild(a);
      a.click();
      a.remove();
    } catch (error) {
      console.error('Error downloading the file:', error);
    }
  };

  return (
    <button onClick={handleDownload}>Download PDF</button>
  );
};
```

```
export default DownloadButton;
```