Thales Azure GenAI/ML Resource Kit

Protecting Sensitive Data

THALES

Contents

1 | Introduction

Generative AI's widespread appeal stems from its potential to transform industries. Businesses aim to leverage it for better customer experiences, streamlined operations, and faster processes. However, the success of these AI/ML applications hinges on data and its context, making the understanding and protection of sensitive enterprise data essential for proper and secure deployment.

Tuning or extending generative AI for specific business uses necessitates data, similar to other AI/ML workloads. A major concern for organizations is minimizing the risk of using their own data, which may contain sensitive personal information (PI/PII), for training. This personal data is often embedded within necessary contextual information for the model to function effectively.

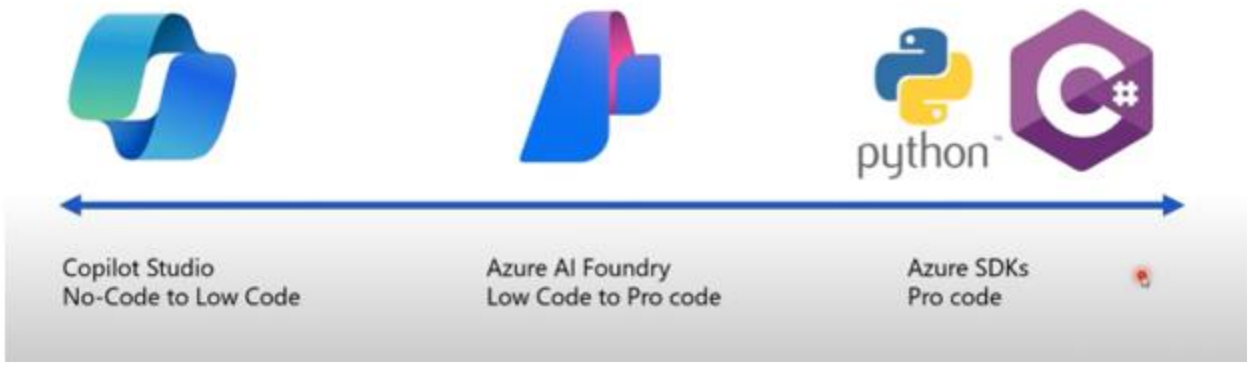
From a security perspective there are many potential security risks associated with the implementation of GenAi systems such as Data Poisoning Definition, Backdoor insertion Definition ,Model theft definition and Prompt injection definition attacks. ***The focus of this document will be on implementing Thales solutions with AZURE capabilities to ensure sensitive data such as PII, PCI, etc will be protected.***

Azure OpenAI Service provides access to OpenAI's powerful language models (like GPT-4o, GPT-3.5 Turbo) through REST APIs, SDKs, and the Azure AI Foundry portal. It allows you to integrate these models into your applications for various tasks, including:

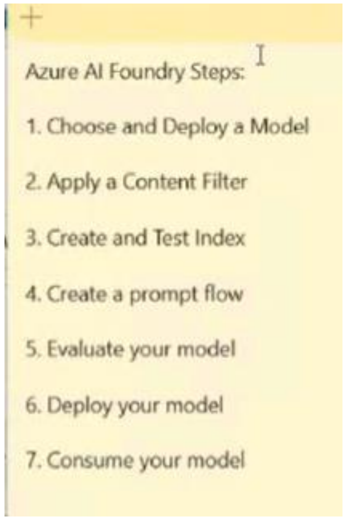
- Content Generation: Creating articles, blog posts, marketing copy, etc.
- Summarization: Condensing large amounts of text into key points.
- Chatbots: Building conversational AI interfaces.
- Code Generation: Assisting developers with writing code.
- Translation: Converting text between languages.
- Image Understanding (with multimodal models): Analyzing and responding to image content.

Key benefits include the enterprise-grade security, scalability, and reliability of Microsoft Azure, along with responsible AI content filtering

Listed below is a diagram highlighting the Azure GenAI tooling.



This is the overall workflow process when using the AI Foundry:


Azure AI Foundry Steps:

1. Choose and Deploy a Model
2. Apply a Content Filter
3. Create and Test Index
4. Create a prompt flow
5. Evaluate your model
6. Deploy your model
7. Consume your model

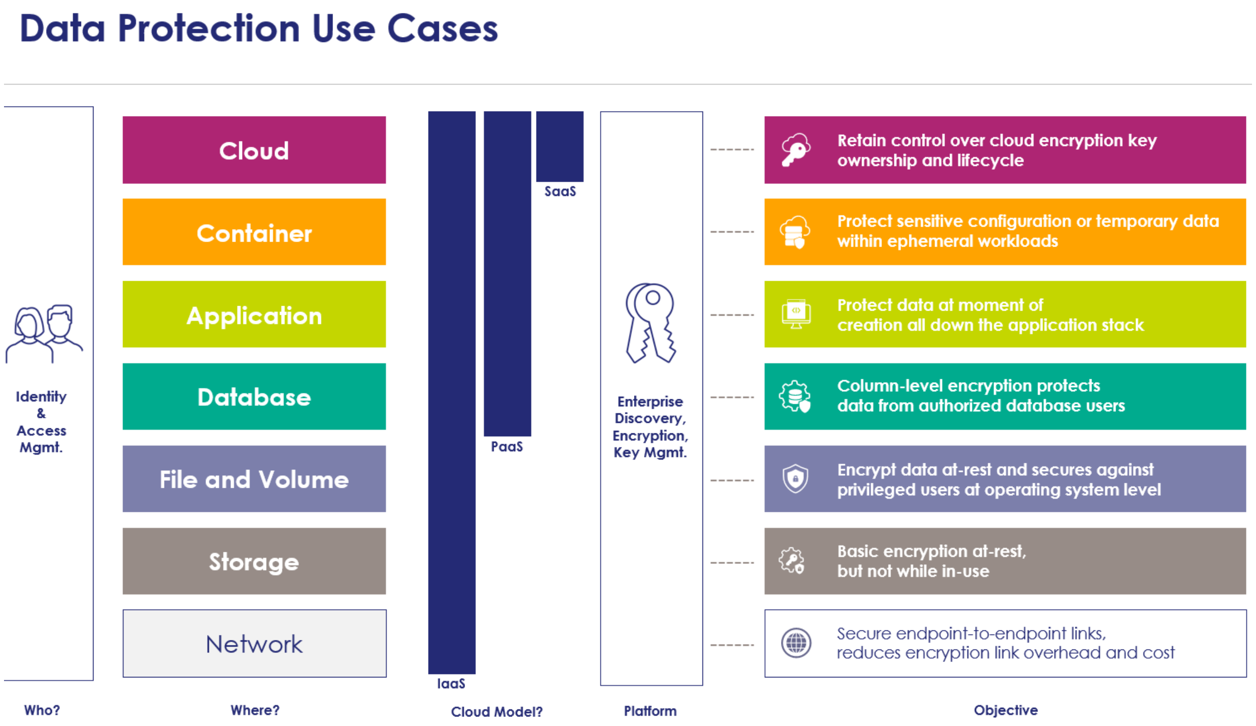
The focus of this document will be on Azure AI Foundry which is a unified platform designed for building, deploying, and managing AI applications at scale. It brings together various Azure AI services, including Azure OpenAI Service, along with tools for:

- **Model Exploration:** Discovering and evaluating a wide range of foundation, open, task-specific, and industry models from different providers (including OpenAI).
- **Customization:** Fine-tuning models, prompt engineering, and Retrieval Augmented Generation (RAG).
- **Agent Building:** Creating AI agents with reasoning and autonomous task execution capabilities.
- **Deployment and Management:** Streamlining the deployment process and offering tools for monitoring and optimization.
- **Collaboration:** Providing a collaborative environment for AI development teams.
- **Responsible AI:** Incorporating safety filters and evaluation metrics.

During the training phases there are situations when the data needs to be in an Azure Blob storage to be processed. Thales provides customers the ability to populate the Azure Key Vault with key material created in an external key manager (CipherTrust Manager) using customer managed keys to achieve better control and compliance. See Appendix on Bring Your Own Encryption & Hold Your Own Key for more information.

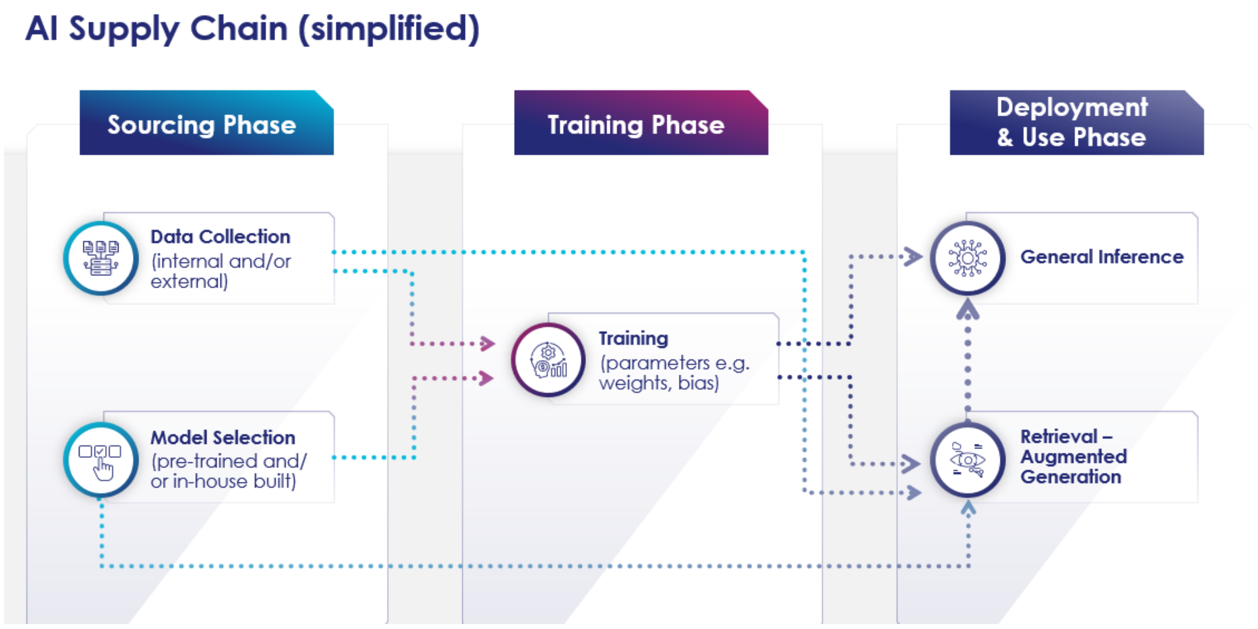
One thing to keep in mind is that using a strategy of Bring Your Own Key (BYOK) or Hold Your Own Key (HYOK) may be sufficient for many use cases, depending on the classification of your data, it may not be appropriate for use cases where the data is highly sensitive. When implementing AZURE native encryption, it is important to know what threats you are protected from with this kind of strategy. The diagram below highlights the different type of encryption and the threat you are protected from. AZURE "Native" encryption is somewhat similar to Full Disk Encryption (FDE) that customers deploy for their on-premises disk. The difference with Azure Key Vault is that instead of having one key for the entire disk AZURE provides customers the ability to have their own key for just their "service or portion of the disk" (whatever the key is protecting in the CSP). The other difference is that customers have the ability to disable the key to prevent access to the service which makes the entire service unavailable. It is important to keep in mind ***you are NOT protected once the disk is spinning and a bad actor logs onto the data center to infiltrate your systems and data***.

As seen from the diagram above when using the Azure AI Foundry capabilities some of the locations of data reside on Azure Blobs in the customer's control. Looking at the encryption stack below it is necessary to deploy a different kind of encryption, bring your own encryption (BYOE) to prevent this kind of threat. For more information on how this can be accomplished see the section titled Bring Your Own Encryption in the Appendix.


Data Protection Use Cases
Identity & Access Mgmt.
Cloud
Container
Application
Database
File and Volume
Storage
Network
SaaS
PaaS
IaaS
Enterprise Discovery, Encryption, Key Mgmt.
Retain control over cloud encryption key ownership and lifecycle
Protect sensitive configuration or temporary data within ephemeral workloads
Protect data at moment of creation all down the application stack
Column-level encryption protects data from authorized database users
Encrypt data at-rest and secures against privileged users at operating system level
Basic encryption at-rest, but not while in-use
Secure endpoint-to-endpoint links, reduces encryption link overhead and cost
Who?
Where?
Cloud Model?
Platform
Objective

The Generative AI workflow can be broadly categorized into three distinct phases. First, the **sourcing phase** involves critical decisions regarding data collection, encompassing the gathering and preparation of relevant datasets, and model selection, where the appropriate pre-trained or custom model is chosen based on the desired application. Next, the **training phase** focuses on fine-tuning the selected model using the sourced data, optimizing its parameters to generate desired outputs. Finally, the **deployment phase** entails making the trained model accessible for practical use, encompassing both general inference, where the model generates outputs based on user prompts, and Retrieval Augmented Generation (RAG), which enhances responses by grounding them in external knowledge sources for improved accuracy and contextuality.

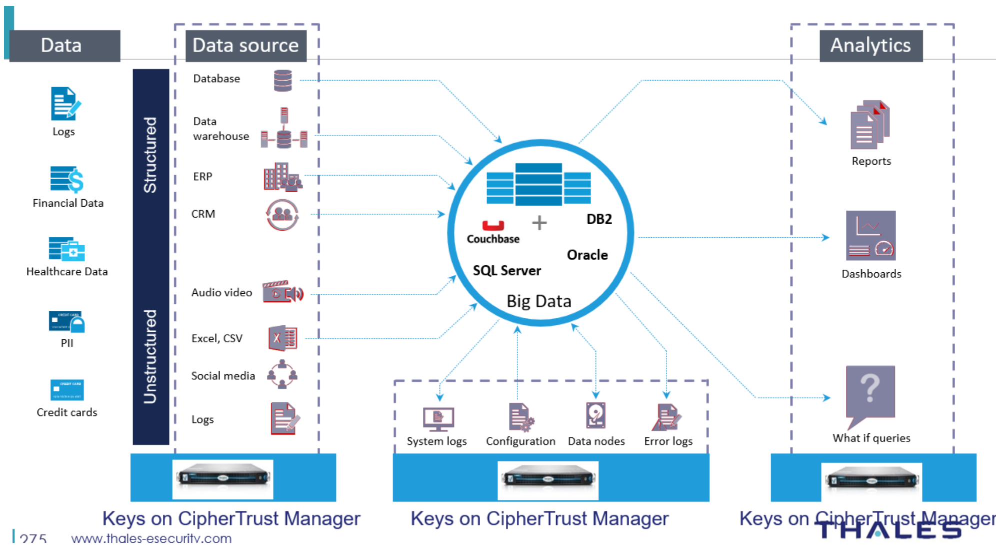
THALES


AI Supply Chain (simplified)
Sourcing Phase
Training Phase
Deployment & Use Phase
Data Collection (internal and/or external)
Model Selection (pre-trained and/ or in-house built)
Training (parameters e.g. weights, bias)
General Inference
Retrieval – Augmented Generation

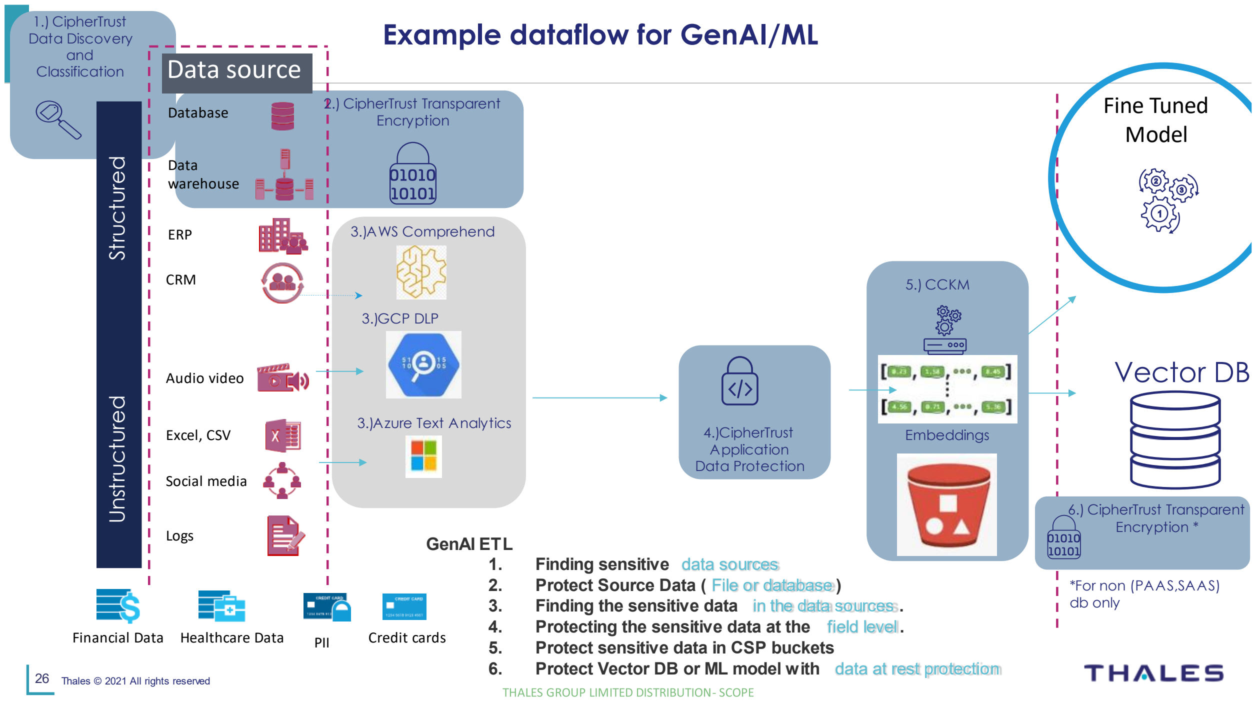
2 | Sourcing Phase.

The first phase is dedicated to sourcing, we want to define which data we will use for the training phase. Data appropriate for GenAI can come from several locations both structured and unstructured***. It is important to protect this data both at a file level and eventually the field or column level as well***. It is estimated that about 80% of organization data is in unstructured data sources which is something to be aware of as finding sensitive data can be a cumbersome and time-consuming process. Organizations should plan on using appropriate data discover and classification tools to identify where the sensitive data is and then once found protect it. Both Thales and AZURE provide this capability which will be explained later in this document. See Appendix section "**Finding sensitive data sources and protecting it"** for more information on this topic.

This workflow is very similar to what customers are already implementing to build data warehouses and data marts. The Extract Transform and Load (ETL) process is to extract data from source systems, consolidate, clean and protect the data in a manner that can provide value to the business. The diagram below shows a traditional ETL environment.



As can be seen above there are many locations during this process where sensitive data may reside. Depending on the classification of the data much of this information should be protected with a strategy like (BYOE) encryption that is stronger than full disk encryption (FDE) as it is **vulnerable to anyone who has access to the operating system.** There are two non-mutually exclusive strategies to protect sensitive data during this process using BYOE protecting the entire file or database and or protecting the column in a database or field in a file.   Please see BYOE section for more information. The same applies for the GenAI ETL process which is shown below along with the areas where Thales can provide additional levels of protection.


Example dataflow for GenAI/ML
1.) CipherTrust Data Discovery and Classification
Data source
Structured
Unstructured
Database
Data warehouse
ERP
CRM
Audio video
Excel, CSV
Social media
Logs
2.) CipherTrust Transparent Encryption
01010 10101
3.)AWS Comprehend
3.)GCP DLP
3.)Azure Text Analytics
4.)CipherTrust Application Data Protection
5.) CCKM
Embeddings
Fine Tuned Model
Vector DB
6.) CipherTrust Transparent Encryption *
*For non (PAAS,SAAS) db only
Financial Data
Healthcare Data
PII
Credit cards
26
Thales © 2021 All rights reserved
THALES GROUP LIMITED DISTRIBUTION- SCOPE
THALES

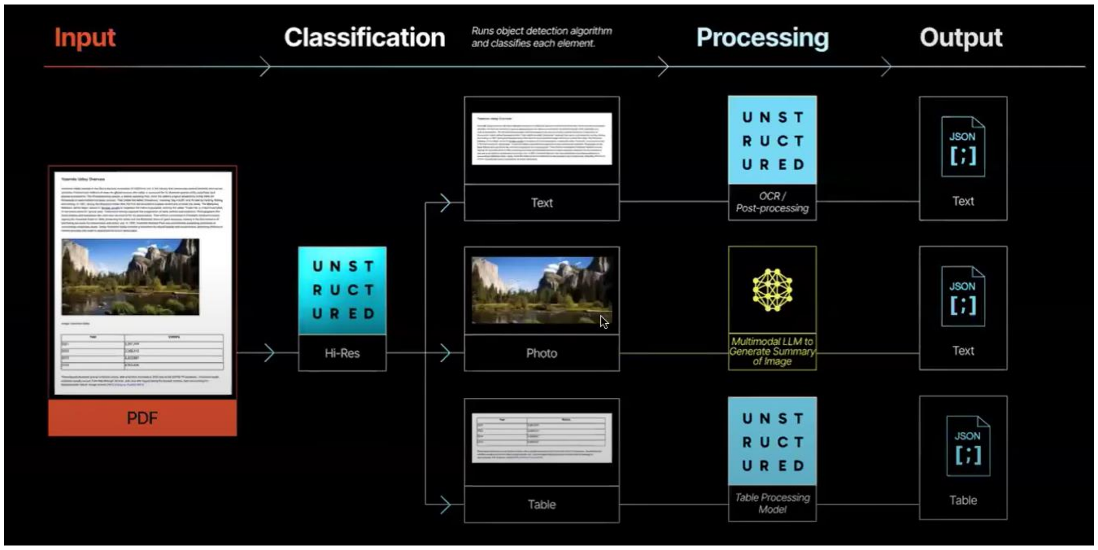
GenAI ETL

1. **Finding sensitive** data sources
2. **Protect Source Data (** File or database **)**
3. **Finding the sensitive data** in the data sources**.**
4. **Protecting the sensitive data at the** field level**.**
5. **Protect sensitive data in CSP buckets**
6. **Protect Vector DB or ML model with** data at rest protection

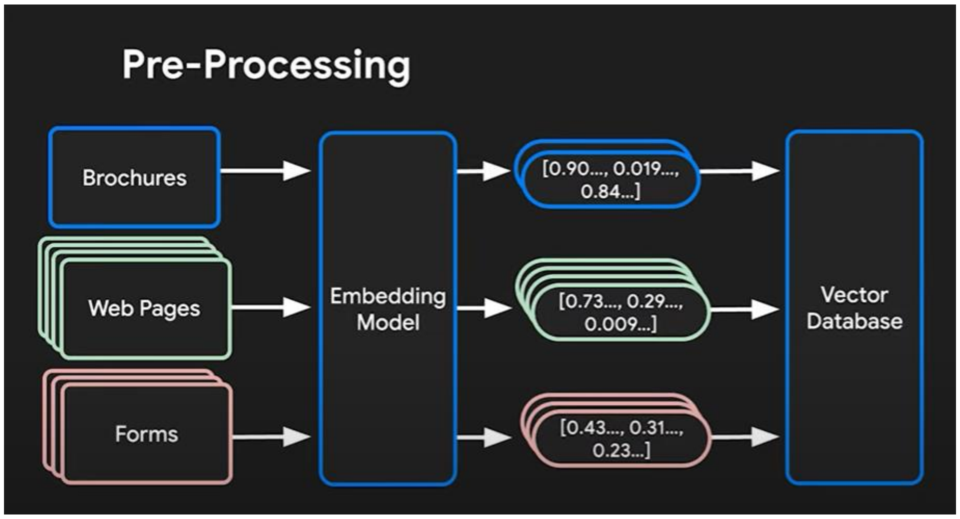
As noted earlier with GenAI and ML environments it is estimated that about 80% of the data to be used is unstructured which typically has **not** been verified to be PII, PCI etc safe. The diagram above shows how the GenAi ETL process has some unique steps that are necessary to prepare the data for training or the RAG but conceptually the process to protect the data is the same except for the fact that a higher % of the sources will be unstructured data.

1. **Finding sensitive data sources**
2. **Protect Source Data (File or database)**
3. **Finding the sensitive data in the data sources.**
4. **Protecting the sensitive data at the field level.**
5. **Protect sensitive data in CSP buckets**
6. **Protect Vector DB or ML model with data at rest protection**

A common example of unstructured data is pdf file that needs to be loaded into a fine tunned model or a vector database. As you can see below the first step is to extract the text from the document. Once this is done text scanning for sensitive data should be incorporated to the workflow converting any sensitive data to encrypted format and written back out in its original sequence replacing the sensitive **field** with the encrypted value. Once this has been accomplished then the data will be ready for output either for training the model or for the Vectordb to be used for the RAG.



If the original sensitive data has not been protected the resulted model will inherit the same data classification as the original data set that was used to train/fine-tune it. This model can now be put under the various sophisticated jail-break, prompt injection attacks and can be forced to reveal the sensitive data it contains. Once the data has been protected the next step is to create embeddings which is done in the training phase and for prompts. The diagram below shows the overall process from sourcing the content to creating embeddings to be loaded into a vectordb and to fine tune a model.



There are various methos (encryption, fine grain authorization) to protect the sensitive data outside of the GenAI model. For example, on how this can be accomplished see Appendix “**Protecting sensitive fields in in file”**

3 | Training Phase

Then we have the second phase, the training phase, which is optional depending on your use case. AZURE defines the training process in 4 steps. Steps 1 & 2 have already been done in the prior phase. As you can see below step 2 is uploading the data into the cloud storage which should be protected at a field level before the Azure TextAnalytics pipeline creates the embeddings. The bucket can also be protected with keys in the Azure Vault and sourced by the external key manager. See sections later in this document on BYOE, BYOK & HYOK.

The training phase of the Generative AI (GenAI) process is where the model learns to understand and generate new data that resembles the data it was trained on. During step 1 the data should be formatted in a specific json format. There are a couple of formats that can be used depending on the use case.   See this link for more details.

https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/fine-tuning?context=%2Fazure%2Fai-studio%2Fcontext%2Fcontext&tabs=azure-openai&pivots=programming-language-studio

This format is fine for basic evaluation or prompt-response datasets. It is commonly used for LLM fine-tuning or simple response evaluation.

```
[
  {
    "query": "What is the capital of France?",
    "response": "The capital of France is Paris."
  }
]
```

As you can see this format contains more content.

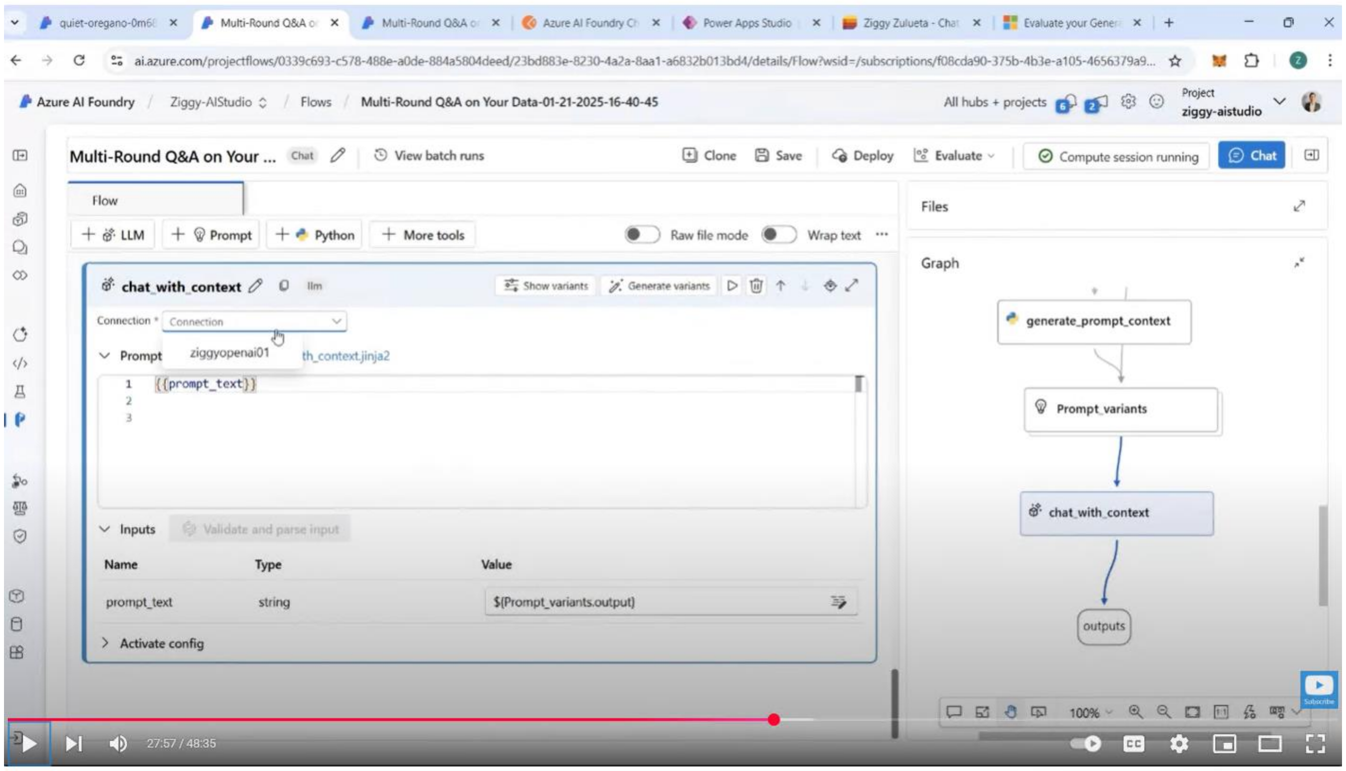
```
[
  {
    "query": "What is the capital of France?",
    "response": "The capital of France is Paris.",
    "context": "France is a country in Europe. Its capital city is Paris.",
    "ground_truth": "Paris"
  }
]
```

- **context:** Useful when evaluating **RAG (Retrieval-Augmented Generation)** systems or testing whether the model can **reason using specific background content.**
- **ground_truth:** Used for **evaluation/benchmarking.** You compare the response to the ground_truth to compute metrics like **exact match, BLEU,** or **F1 score.**
- This structure is essential for **evaluation pipelines,** such as using Azure AI Foundry's **EvalML** or **AzureML evaluations** for precision testing.

So, which should you use?

Use Case	**Format**
Basic LLM prompt/response logging or fine-tuning	query + response
Evaluation with known correct answers	Add ground_truth
RAG testing or context-aware QA	Add context
Full benchmark testing (e.g., with EvalML, LangChain Eval, etc.)	All four fields

AZURE AI Foundry has a workflow product that automates the process of training a model. Listed below is an example of the workflow.



Name	Type	Value
prompt_text	string	${Prompt_variants.output}

One of the steps for the workflow is to define what bucket contains the data for training. As you can see from the screen below fine tuning a model requires uploading data to a bucket which by default includes AZURE native encryption of the data in the Azure Blob Storage which again may not be enough protection depending on your data sensitivity.

This bucket can be protected with two different strategies using the Azure Key Vault or BYOE. The process to protect the sensitive data is the same as described in the sourcing phase.   Please see Appendix BYOE & Bring Your Own Keys and Hold Your Own Keys section for more information on how to deepen your security for AZURE blob storage.

Weather you decide to protect the data at a field level or not with the process above you also may consider strong levels of protection for the data stored in the AZURE buckets as there may be other data you do not want revealed to unintended audiences.

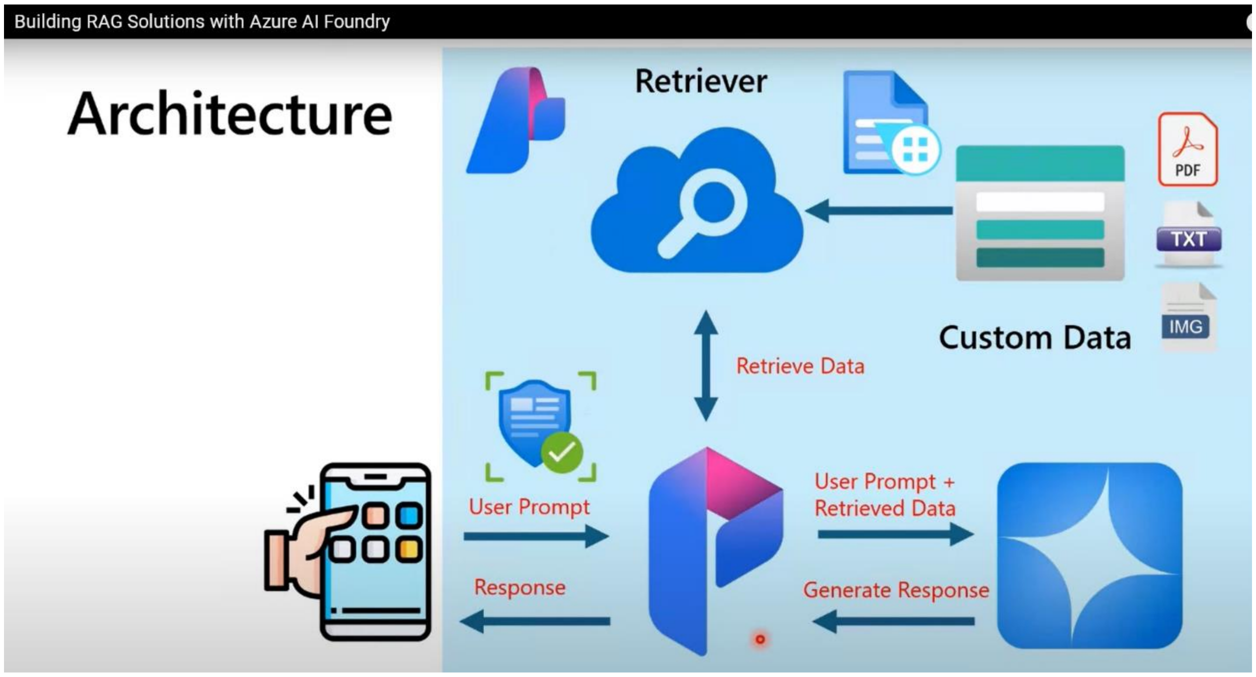
4 | Deployment & Use Phase.

And the last phase is the inference phase, where the decision makers will select a General Machine learning/General Inference technique or a RAG technique.

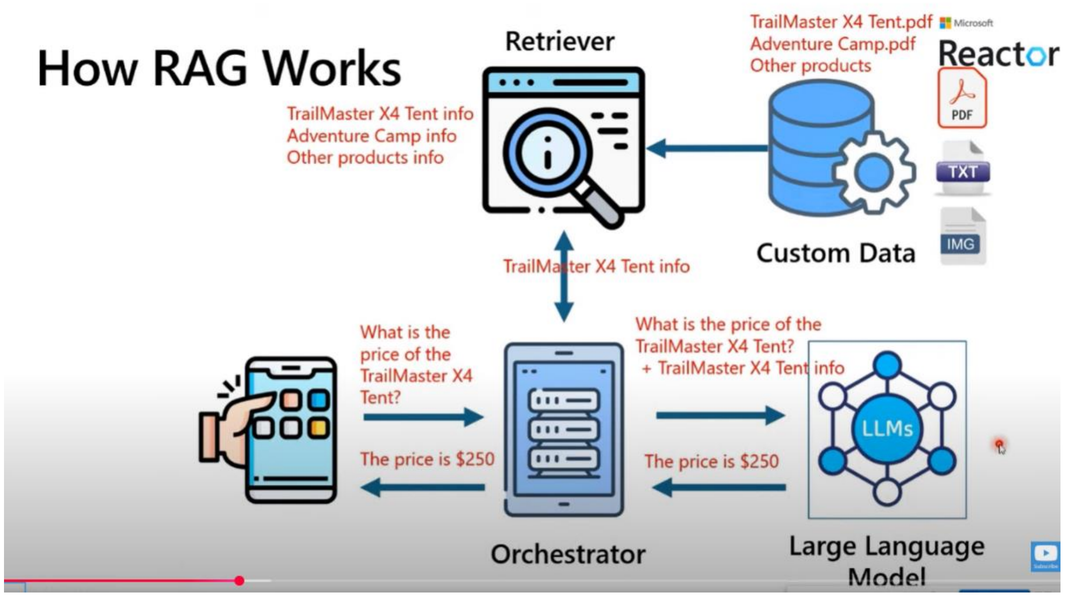
Retrieval Augmented Generation (RAG) addresses the limitations of Large Language Models (LLMs) by providing them with access to relevant, up-to-date, and domain-specific data. By grounding LLM responses in proprietary, private, or dynamic information, RAG significantly enhances accuracy and contextuality, overcoming the inherent static nature and knowledge gaps of LLMs. This is crucial for business applications requiring precise, context-aware outputs.

RAG also enables InfoSec team to apply the necessary security tools which are technically not possible within the model itself.

Azure AI Foundry provides powerful APIs, SDKs, and services like Azure TextAnalytics RAG Engine and Azure TextAnalytics Pipelines that enable you to build your own automated data ingestion workflows tailored to your specific needs and data sources. These tools offer flexibility and control over the entire process. The diagram below shows the entire RAG process.



In the diagram below the retriever is the Vector database. The Vector database can be populated in a few ways including using the AI Foundry UI. By default the Azure AI Search Vector DB is used.



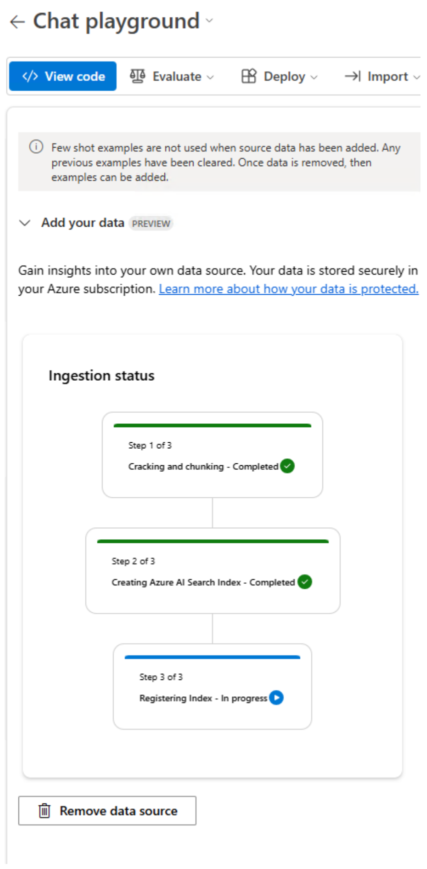
Here is a listing of all the Vector Database options from Microsoft.

THALES

-

- Store your embeddings and perform vector (similarity) search using your choice of Azure service:
  - Azure AI Search
  - Azure SQL Database
  - Azure Cosmos DB for MongoDB vCore
  - Azure SQL Database
  - Azure Cosmos DB for NoSQL
  - Azure Cosmos DB for PostgreSQL
  - Azure Cache for Redis

The Orchestrator is the prompt flow capability that AI Foundry uses to access the Vector database for relevant content and then add that to the original prompt to be sent to the LLM. Here is screenshot showing the different stages that need to take place to load the data into the Vector DB.



Few shot examples are not used when source data has been added. Any previous examples have been cleared. Once data is removed, then examples can be added.

Gain insights into your own data source. Your data is stored securely in your Azure subscription. Learn more about how your data is protected.

Ingestion status

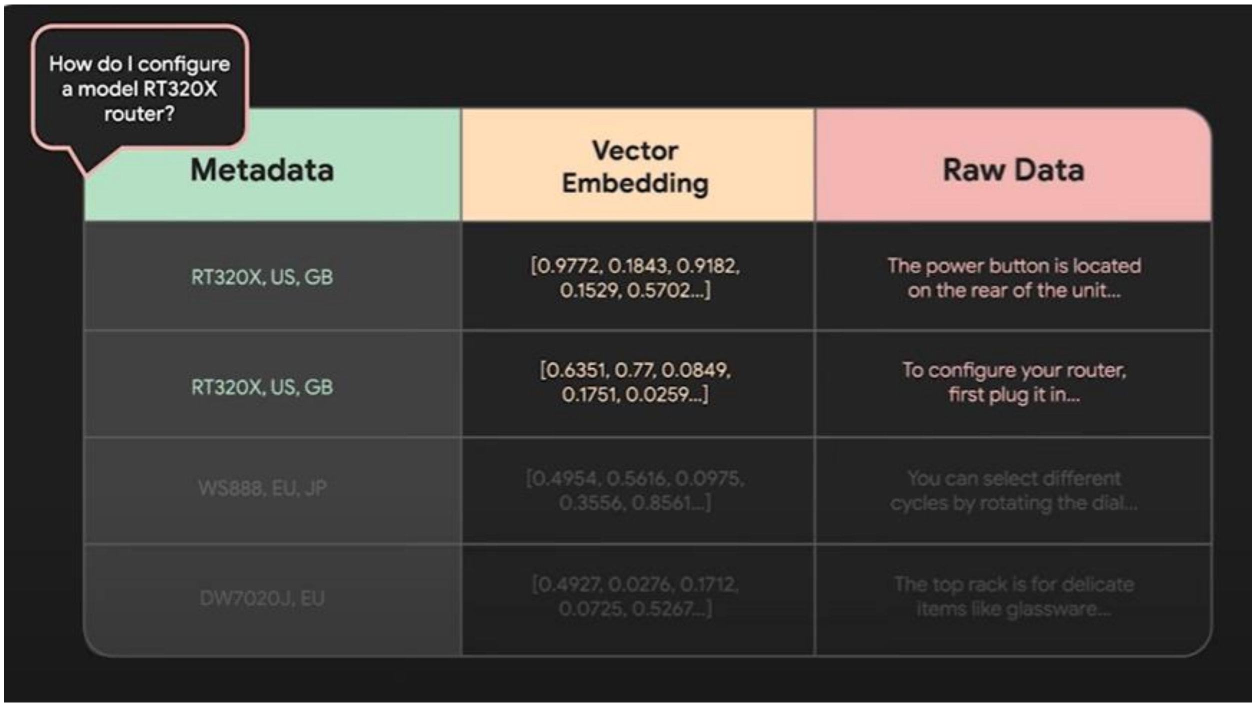
Step 1 of 3
Cracking and chunking - Completed

Step 2 of 3
Creating Azure AI Search Index - Completed

Step 3 of 3
Registering Index - In progress

Sensitive Data in RAG Metadata

When creating your embeddings it is possible to store custom metadata along with the metadata. You can store metadata such as author, team, department etc. to meet your enterprise prompt management needs. If some of this information is sensitive, it may need to be encrypted to protect it from unintended usage.


How do I configure a model RT320X router?

Metadata	Vector Embedding	Raw Data
RT320X, US, GB	[0.9772, 0.1843, 0.9182, 0.1529, 0.5702...]	The power button is located on the rear of the unit...
RT320X, US, GB	[0.6351, 0.77, 0.0849, 0.1751, 0.0259...]	To configure your router, first plug it in...
WS888, EU, JP	[0.4954, 0.5616, 0.0975, 0.3556, 0.8561...]	You can select different cycles by rotating the dial...
DW7020J, EU	[0.4927, 0.0276, 0.1712, 0.0725, 0.5267...]	The top rack is for delicate items like glassware...

The primary Azure AI Foundry product within Azure TextAnalytics that populates and serves as the vector database for Azure AI Foundry applications is **Azure AI Search**

Azure TextAnalytics Vector Search: This is the managed vector database service in AZURE that allows you to store and efficiently search through vector embeddings. You populate it with the vector representations of your data (text, images, audio, etc.) generated using embedding models (often also hosted on Vertex AI, like textembedding-gecko).

Azure TextAnalytics Vector Search is the dedicated, managed service within the Azure TextAnalytics ecosystem optimized for this purpose in GenAI applications. It offers features specifically tailored for vector search workloads, including efficient indexing and querying algorithms.

Prompt Management/Engineering

Effective prompt design for language models isn't governed by strict rules, but rather by strategic techniques that can influence the model's output. However, thorough testing and evaluation are essential for achieving optimal results.

Prompt engineering is about *how* to write good prompts and prompt management is about *how* to organize and deploy those prompts effectively in both use cases sensitive data could be contained from the original data entered by the user or data that has been obtained from the Vector database or SQL queries for user specific content. Listed below are a couple of links that provide more information.
https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/prompt-engineering?context=%2Fazure%2Fai-foundry%2Fcontext%2Fcontext&tabs=chat

Microsoft uses something called “Prompt Flow” to manage the prompting in AI Foundry. Prompt flow is a development tool designed to streamline the entire development cycle of AI applications powered by Large Language Models (LLMs). Prompt flow provides a comprehensive solution that simplifies the process of prototyping, experimenting, iterating, and deploying your AI applications.

Prompt flow is available independently as an open-source project on GitHub, with its own SDK and VS Code extension. Prompt flow is also available and recommended to use as a feature within both Azure AI Foundry and Azure Machine Learning studio.

Prompt flow offers a well-defined process that facilitates the seamless development of AI applications. By using it, you can effectively progress through the stages of developing, testing, tuning, and deploying flows, ultimately resulting in the creation of fully fledged AI applications

The lifecycle consists of the following stages:

- Initialization: Identify the business use case, collect sample data, learn to build a basic prompt, and develop a flow that extends its capabilities.
- Experimentation: Run the flow against sample data, evaluate the prompt's performance, and iterate on the flow if necessary. Continuously experiment until satisfied with the results.
- Evaluation and refinement: Assess the flow's performance by running it against a larger dataset, evaluate the prompt's effectiveness, and refine as needed. Proceed to the next stage if the results meet the desired criteria.
- Production: Optimize the flow for efficiency and effectiveness, deploy it, monitor performance in a production environment, and gather usage data and feedback. Use this information to improve the flow and contribute to earlier stages for further iterations.

The process of prompt engineering is iterative and relies on testing to improve model performance. When crafting prompts, clearly define goals and anticipated results, and systematically test different approaches to identify areas for optimization.
A prompt's effectiveness hinges on two key aspects: its content and its structure. See link below for more information:
https://learn.microsoft.com/en-us/azure/ai-foundry/concepts/prompt-flow

Here is an example of how a possible prompt interaction would work with a GenAI agent.

Input:

*[Agent] Hi, my name is **Jason**, can I have your name?*

*[Customer] My name is **Valeria***

[Agent] In case we need to contact you, what is your email address?

*[Customer] My email is **v.racer@example.org***

[Agent] Thank you. How can I help you?

[Customer] I’m having a problem with my bill.

De-identified Output:

*[Agent] Hi, my name is **g4xQF**, can I have your name?*

*[Customer] My name is **3aVls3k***

[Agent] In case we need to contact you, what is your email address?

*[Customer] My email is **E.zu9yn@XwgFwY3.TO7***

[Agent] Thank you. How can I help you?

[Customer] I'm having a problem with my bill.

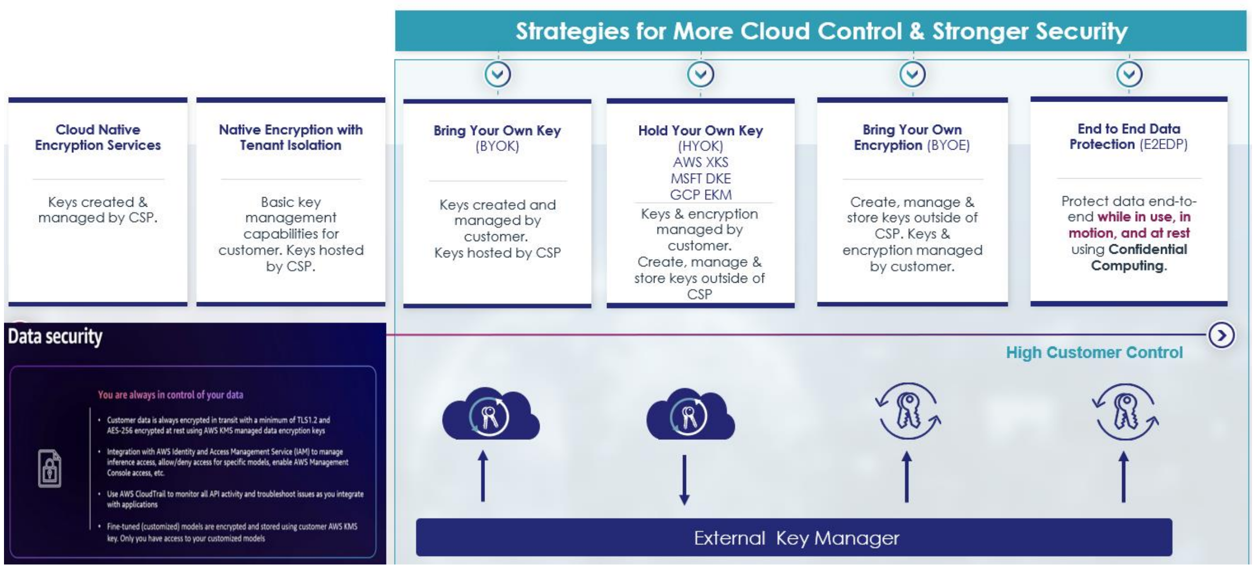
See the Appendix section (**Prompt Protection**) is an example of how to use Thales encryption capabilities to protect sensitive data either going into the prompt or from the model response. There may be scenarios where you need to re-identify a person because of the type of application or the right to removal in case of compliance. This can be done with a tagging architecture. For more information see examples provided in the git repository provided in the appendix.

5 | Appendix.

Bring Your Own Keys and Hold Your Own Keys

Native Encryption Services: Can be deceptively simple to use, but very difficult to live with…The problem with native encryption & key mgmt. is that it is UNIQUE to the provider. These services are 100% managed by the CSPs. Also, customers have limited control and visibility over their cloud-encrypted data. There is no way to know who is accessing what and why; depending on the industry you are in, the sensitivity of data and the Cloud Service Provider, you may need to complement cloud security with additional controls for compliance. The dark blue section on the lower left is a description of AWS Data Security for GenAI outlining the native KMS encryption.

Protection Strategies for Data in the Cloud


Strategies for More Cloud Control & Stronger Security
Cloud Native Encryption Services
Keys created & managed by CSP.
Native Encryption with Tenant Isolation
Basic key management capabilities for customer. Keys hosted by CSP.
Bring Your Own Key (BYOK)
Keys created and managed by customer. Keys hosted by CSP
Hold Your Own Key (HYOK) AWS XKS MSFT DKE GCP EKM
Keys & encryption managed by customer. Create, manage & store keys outside of CSP
Bring Your Own Encryption (BYOE)
Create, manage & store keys outside of CSP. Keys & encryption managed by customer.
End to End Data Protection (E2EDP)
Protect data end-to-end while in use, in motion, and at rest using Confidential Computing.
High Customer Control
External Key Manager
Data security

You are always in control of your data

- Customer data is always encrypted in transit with a minimum of TLS1.2 and AES-256 encrypted at rest using AWS KMS managed data encryption keys
- Integration with AWS Identity and Access Management Service (IAM) to manage inference access, allow/deny access for specific models, enable AWS Management Console access, etc.
- Use AWS CloudTrail to monitor all API activity and troubleshoot issues as you integrate with applications
- Fine-tuned (customized) models are encrypted and stored using customer AWS KMS key. Only you have access to your customized models

Bring Your Own Key (BYOK): For customers that need greater control and visibility over the cloud-encrypted data, the providers offer variations of flexible key management such as Bring Your Own Key (BYOK) It is unique to the customer, depending on how they generate and manage their keys. Now, you have some control over your encrypted data because you can bring your own key material or master keys. Customers have the ability to generate and import the encryption keys or key material for their cloud-native encryption services. But still, there are gaps in visibility, control and usage of keys. Nobody know what happens after you bring your own key, if the cloud admins can access them, what happens in case of a breach or government subpoenas, etc.. Require additional tools for multi-cloud and hybrid data security & compliance.

Hold Your Own Key (HYOK) refers to a security model where organizations maintain full control over their encryption keys, even when using cloud services provided by a third party like **Azure AI Foundry Platform (AZURE)**, **Amazon Web Services (AWS)**, or **Microsoft Azure**. In this model, the organization uses its own key management infrastructure, often located on-premises or in a third-party environment, to generate, store, and manage encryption keys. The cloud provider does not have access to these keys, ensuring that even if the cloud infrastructure is compromised, the encrypted data remains secure and inaccessible.

Benefits of External Key Manger

For both Bring Your Own Keys and Hold Your Own Keys you get can obtain the following benefits from the Key Manager.

1. Key Lifecycle Management
    a. Detail: Native CSP key management services has limited ability to automate the lifecycle of keys especially across multiple subscriptions
    Impact: Customers have to implement expensive manual key management processes to meet internal key security requirements

2. Attaining Compliance
    a. Detail: Insufficient authorization control or DR services to ensure keys are not accidentally or intentionally deleted
       Impact: There is too high of risk that data will be lost
    b.
3. Encryption Key Visibility
    a. Detail: Internal and external regulations require that encryption keys do not permanently reside in the cloud
       Impact: Enterprise cannot move data to the cloud where data regulations require more control of keys that are locally stored
    b.
4. Assign permissions to keys
    a. Only allow certain accounts access and operations such as encrypt,decrypt. Prevent deletes unless have a quarum. Have dates expiration dates with automatic key roation schedules, Keys have states (active,destroyed,deactivated) as approved by nist. Should be able to search on keys by attributes like algorithums, key length etc.

For more information see link below:

https://www.thalestct.com/wp-content/uploads/2022/09/Best-Practices-Cloud-Data-Protection-and-Key-Management-TCT-WP.pdf

https://cpl.thalesgroup.com/blog/cloud-security/shared-responsibility-model-cloud

Bring Your Own Encryption (BYOE)

There are a couple of use cases to consider when using a BYOE strategy to protect sensitive data for GenAI platforms. The first is protecting the entire file or all files in a particular directory or S3 bucket and only allowing a certain process or user to access the content. The second is to protect the fields in a file that contain sensitive data. For working example on how to protect sensitive fields in a file see Appendix section on Examples.

Protecting the entire file.

Thales provides a capability called CipherTrust Transparent Encryption(CTE) to protect the entire file in a directory or S3 bucket. See links for more information on CTE.

https://cpl.thalesgroup.com/encryption/transparent-encryption

https://www.youtube.com/watch?v=VzRxrhUZfc0

https://www.thalestct.com/wp-content/uploads/2022/09/aws-s3-with-cte-sb.pdf

https://www.thalestct.com/wp-content/uploads/2022/09/avoiding-amazon-s3-leaks-wp.pdf

Protecting fields in a file.

Thales provides several different application encryption and tokenization capabilities.

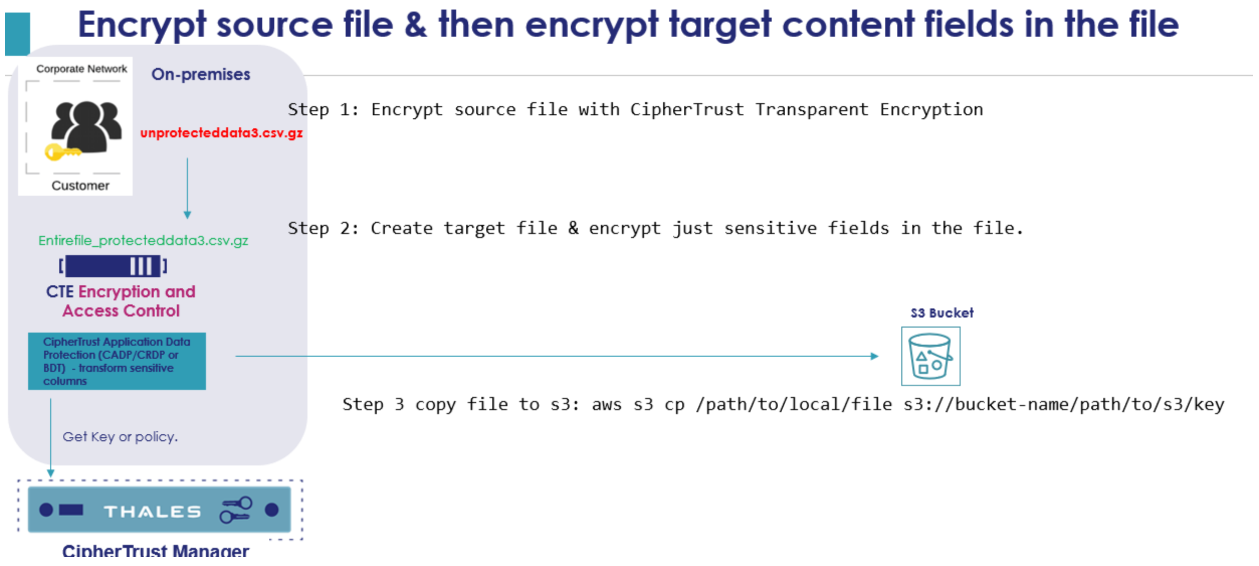
See links below for more information.

https://cpl.thalesgroup.com/encryption/ciphertrust-application-data-protection

https://cpl.thalesgroup.com/encryption/tokenization

Note for some use cases it might be necessary to protect the file with both types of encryption depending on what stage of the process it is during the ETL process. Listed below are a couple of options on how this can be accomplished.

Option 1. Encryption on premises.


Encrypt source file & then encrypt target content fields in the file

Step 1: Encrypt source file with CipherTrust Transparent Encryption

Step 2: Create target file & encrypt just sensitive fields in the file.

Step 3 copy file to s3: aws s3 cp /path/to/local/file s3://bucket-name/path/to/s3/key

Corporate Network
On-premises
Customer
unprotecteddata3.csv.gz
Entirefile_protecteddata3.csv.gz
CTE Encryption and Access Control
CipherTrust Application Data Protection (CADP/CRDP or BDT) - transform sensitive columns
Get Key or policy.
S3 Bucket
THALES
CipherTrust Manager

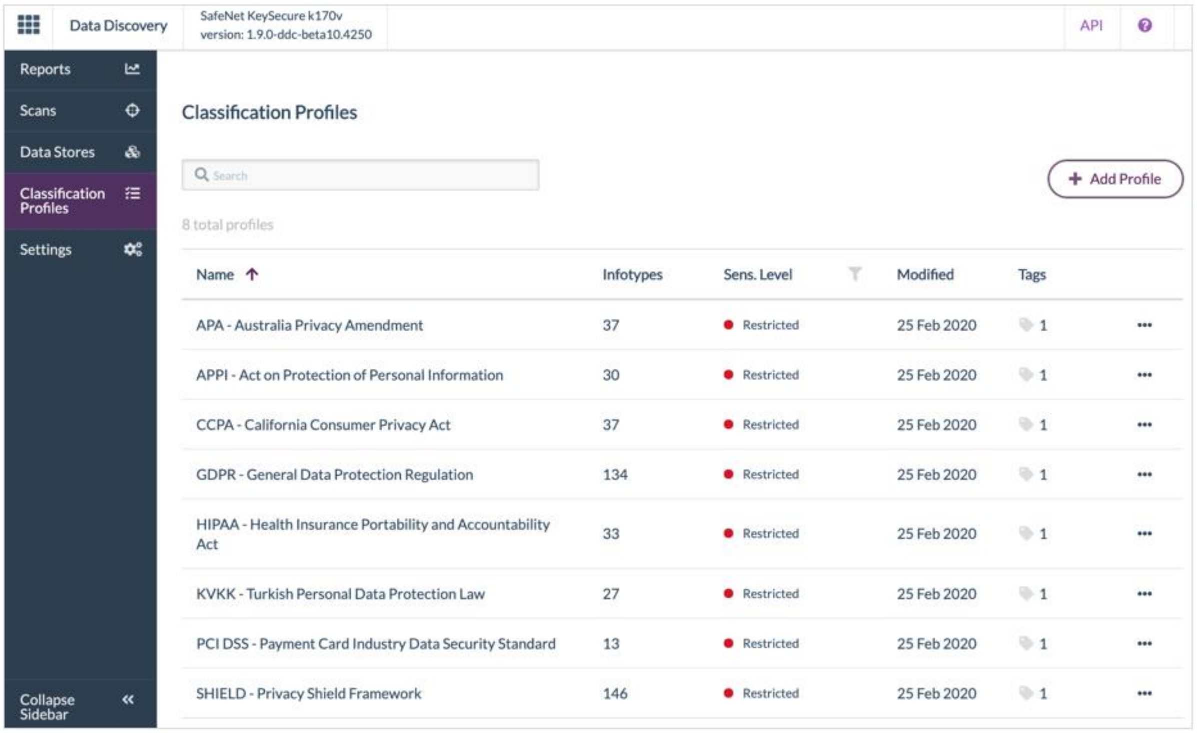
For working example see **Protecting sensitive fields in in file** in the Appendix Examples section.

Finding sensitive data sources and protecting it.

This content in this section focuses on encryption of sensitive data for AI and ML environments. It will explain how data needs to be protected at a field level in a file before the content is made available to a vector database or used for training an AI model. Since the content is specifically used by the models encrypting it at a field level is the most secure and appropriate for this use case.

When protecting sensitive data for general business use protecting sensitive data in a file at a field level may or may not be appropriate since decrypting on the fly and incorporating RBAC may not be available to the applications using the files. There are other methods such as Thales CipherTrust Manger Transparent Encryption (CTE) that may be more appropriate for these kinds of use cases.

As noted earlier it is critical that organizations find data sources that have sensitive information to avoid breaches. Thales provides a product that will find the datasources that contain sensitive data and also provide a report that shows the sensitivity and other metrics such as the information type (credit card, email, etc). Listed below is a screenshot showing the classification profiles and sensitivity levels.



Classification Profiles

8 total profiles

Name	Infotypes	Sens. Level	Modified	Tags
APA - Australia Privacy Amendment	37	● Restricted	25 Feb 2020	1
APPI - Act on Protection of Personal Information	30	● Restricted	25 Feb 2020	1
CCPA - California Consumer Privacy Act	37	● Restricted	25 Feb 2020	1
GDPR - General Data Protection Regulation	134	● Restricted	25 Feb 2020	1
HIPAA - Health Insurance Portability and Accountability Act	33	● Restricted	25 Feb 2020	1
KVKK - Turkish Personal Data Protection Law	27	● Restricted	25 Feb 2020	1
PCI DSS - Payment Card Industry Data Security Standard	13	● Restricted	25 Feb 2020	1
SHIELD - Privacy Shield Framework	146	● Restricted	25 Feb 2020	1

Once Thales Data Discovery and Classification (DDC) has found the sensitive data at a **file level** it needs to be protected at **field level**. In order to find the actual attribute in the file AZURE provides a very powerful API called DLP that can find PII data in a file and then once found can be protected with Thales encryption. See Examples section below for detailed example.

https://cpl.thalesgroup.com/encryption/data-discovery-and-classification

https://cloud.google.com/sensitive-data-protection/docs/reference/rpc/google.privacy.dlp.v2

AZURE Text Analytics vs Microsoft Purview

The relationship between the Azure Text Analytics API and Microsoft Purview is not a direct one where Purview solely uses the Text Analytics API for all its text-related functionalities. Instead, **Microsoft Purview likely leverages various Azure services and its own built-in capabilities for text analysis**, and the Text Analytics API could be one of them for specific tasks.
Here's a breakdown to clarify:
Azure Text Analytics API (now part of Azure AI Language)

- This is a collection of cloud-based APIs that use natural language processing (NLP) for text analysis.
- It offers features like:
    - **Sentiment Analysis:** Determining the positive, negative, or neutral sentiment of text.
    - **Key Phrase Extraction:** Identifying the main topics or talking points in a piece of text.
    - **Language Detection:** Automatically detecting the language of the input text.
    - **Named Entity Recognition (NER):** Identifying and categorizing entities like people, organizations, locations, etc.
    - **Personally Identifiable Information (PII) Detection:** Identifying and redacting sensitive personal information.
    - **Text Summarization:** Condensing long texts into shorter summaries.
    - **Custom Text Classification and Entity Recognition:** Building custom models for specific classification or entity recognition needs.

Microsoft Purview

- This is a unified data governance service that helps you manage and govern your data across on-premises, multicloud, and SaaS environments.
- Its core functionalities include:
    - **Data Discovery and Classification:** Automatically scanning and classifying data assets based on sensitivity and type.
    - **Data Lineage:** Tracking the movement and transformation of data across systems.
    - **Data Catalog:** Creating a searchable inventory of data assets with rich metadata.
    - **Data Governance:** Defining and enforcing data policies and standards.
    - **Data Security:** Identifying and protecting sensitive data.
    - **Risk and Compliance:** Helping organizations meet regulatory requirements.

Examples

Protecting sensitive fields in in file

As mentioned earlier it is estimated about 80% of data in an organization is unstructured files such as pdf text and csv files. It is important to convert the sensitive data in those files before the content is used in a vector database or used to fine train a model. Here is an example of how this can be accomplished using the Thales Application Encryption along with the AZURE Text Analytics API to find sensitive data in a file.

```
import java.io.*;
import java.util.Properties;

/**
 * ThalesAzureProtectRevealBatchProcessor is a batch processing application
 * designed to protect (encrypt) or reveal (decrypt) data within files,
 * specifically targeting Azure-related content, using Thales CipherTrust
 * Data Protection (CADP) solutions.
 * It reads configuration from a properties file, processes input files,
 * and writes the protected/revealed output to a specified directory.
 */
```

```
public class ThalesAzureProtectRevealBatchProcessor {

        // A static Properties object to hold configuration loaded from thales-config.properties
        private static Properties properties;

        /**
         * Static initializer block to load properties from "thales-config.properties"
         * when the class is loaded. This ensures that configuration is available
         * before any other methods are called.
         * It throws a RuntimeException if the properties file cannot be found or loaded.
         */
        static {
                try (InputStream input =
ThalesAzureProtectRevealBatchProcessor.class.getClassLoader()
                                .getResourceAsStream("thales-config.properties")) {
                        properties = new Properties();
                        if (input == null) {
                                // If the input stream is null, the properties file was not found
                                throw new RuntimeException("Unable to find udfConfig.properties");
                        }
                        // Load properties from the input stream
                        properties.load(input);

                } catch (Exception ex) {
                        // Catch any exceptions during file loading and wrap them in a
RuntimeException
                        throw new RuntimeException("Error loading properties file", ex);
                }
        }

        /**
         * The main method is the entry point of the application.
         * It orchestrates the entire process of reading configuration,
         * initializing helpers, processing files, and reporting statistics.
         *
         * @param args Command-line arguments:
         * args[0]: mode (e.g., "protect" or "reveal")
         * args[1]: Azure Cognitive Services Endpoint
         * args[2]: Azure Cognitive Services API Key
         * args[3]: Input directory path
         * args[4]: Output directory path
         * args[5]: File extension to process (e.g., "pdf", "txt")
         * @throws IOException If an I/O error occurs during file operations.
         */
        public static void main(String[] args) throws IOException {

                // Record the start time in nanoseconds for performance measurement
                long startTime = System.nanoTime();

                // Flag to skip header (not explicitly used in the provided code, but often useful
in batch processing)
                boolean skiphdr = false;
                // Cloud Service Provider, currently hardcoded to "azure"
                String csp = "azure";
                // Counter for number of records processed in the current file
                int nbrofrecords = 0;
                // Total number of records processed across all files
                int totalnbrofrecords = 0;

                // Retrieve various configuration properties from the loaded properties object
                String metadata = properties.getProperty("METADATA");
                String crdpip = properties.getProperty("CRDPIP");
                String keymanagerhost = properties.getProperty("KEYMGRHOST");
                String crdptkn = properties.getProperty("CRDPTKN");
                String policyType = properties.getProperty("POLICYTYPE");
                String defaultpolicy = properties.getProperty("DEFAULTPOLICY");
                String showmetadata = properties.getProperty("SHOWMETADATA");
                String revealuser = properties.getProperty("REVEALUSER");
                System.out.println(" user  " + revealuser); // Print the reveal user

                // Convert showmetadata property to a boolean
                boolean showmeta = showmetadata.equalsIgnoreCase("true");

                // Initialize ThalesCADPProtectRevealHelper. This helper class is responsible
                // for interacting with the Thales CipherTrust Manager for cryptographic
operations.
                ThalesCADPProtectRevealHelper tprh = new
ThalesCADPProtectRevealHelper(keymanagerhost, crdptkn, null,
                                policyType, showmeta);

                // The commented-out line below shows an alternative helper for REST-based
protection/reveal.
                // ThalesProtectRevealHelper tprh = new ThalesRestProtectRevealHelper(crdpip,
metadata, policyType, showmeta);

                // Set additional properties for the Thales CADP helper
                tprh.revealUser = revealuser;
                tprh.policyType = policyType;
                tprh.policyName = properties.getProperty("POLICYNAME");
                tprh.defaultPolicy = defaultpolicy;

                // Initialize a ContentProcessor based on the Cloud Service Provider.
                // This processor handles the specific content type (e.g., Azure documents).
                ContentProcessor cp = null;
                if (csp.equalsIgnoreCase("azure"))
                        cp = new AzureContentProcessor(properties);

                // Declare File objects for input and output directories
                File inputDir = null;
                File outputDir = null;

                // Parse command-line arguments
                String mode = args[0]; // "protect" or "reveal"
                AzureContentProcessor.cognitiveservices_endpoint = args[1]; // Azure Cognitive
Services Endpoint
                AzureContentProcessor.cognitiveservices_apiKey = args[2];   // Azure Cognitive
Services API Key
                inputDir = new File(args[3]); // Input directory path
                outputDir = new File(args[4]); // Output directory path
                String fileextension = args[5]; // File extension to process

                int nbroffiles = 0; // Counter for the number of files processed

                // Get a list of files from the input directory that match the specified file
extension
                File[] inputFiles = inputDir.listFiles((dir, name) ->
name.toLowerCase().endsWith(fileextension));

                File outputFile = null; // Declare outputFile here to be accessible within the
loop

                // Check if any input files were found
                if (inputFiles != null) {
                        // Iterate through each input file
                        for (File inputFile : inputFiles) {

                                // Determine the output file name based on the mode ("protected" or
"revealed" prefix)
                                if (mode.equalsIgnoreCase("protect"))
                                        outputFile = new File(outputDir, "protected" +
inputFile.getName());
                                else
                                        outputFile = new File(outputDir, "revealed" +
inputFile.getName());

                                // Process the file using the appropriate ContentProcessor
(AzureContentProcessor in this case)
                                if (csp.equalsIgnoreCase("azure"))
                                        nbrofrecords = cp.processFile(inputFile, outputFile, null,
tprh, mode, skiphdr);

                                nbroffiles++; // Increment the count of processed files
                        }
                        // Accumulate the total number of records processed
                        totalnbrofrecords = totalnbrofrecords + nbrofrecords;
                } else {
                        // If no files were found, print a message
                        System.out.println("No " + fileextension + " files found in the directory: 
" + inputDir.getAbsolutePath());
                }

                // Record the end time in nanoseconds
                long endTime = System.nanoTime();

                // Calculate the elapsed time in nanoseconds
                long elapsedTimeNano = endTime - startTime;

                // Convert nanoseconds to seconds for readability
                double elapsedTimeSeconds = (double) elapsedTimeNano / 1_000_000_000.0;

                // Print the application execution time
                System.out.printf("Application execution time: %.6f seconds%n", 
elapsedTimeSeconds);

                // Print final statistics
                System.out.println("Number of files = " + nbroffiles);
                System.out.println("Total nbr of records = " + totalnbrofrecords);
                // Assuming cp (ContentProcessor) has these public fields for statistics
                System.out.println("Total skipped entities = " + cp.total_skipped_entities);
                System.out.println("Total entities found = " + cp.total_entities_found);
        }
}
```

To view the entire solution see the following github repository: ???

Prompt Protection

This java example uses a couple of helper classes, ThalesProtectRevealHelper to control settings for the CiphterTrust Manager and associated profile information and ContentProcessor to control settings for the content to be protected. To view the entire solution see the following github repository: ???

```
import okhttp3.*; // Import OkHttp for making HTTP requests

import org.json.JSONArray; // Import JSONArray for handling JSON arrays
import org.json.JSONObject; // Import JSONObject for handling JSON objects

import java.io.IOException; // Import IOException for handling I/O errors
import java.io.InputStream; // Import InputStream for reading from files
import java.util.Iterator; // Import Iterator for iterating over JSON keys
import java.util.Properties; // Import Properties for loading configuration from a file

/**
 * ThalesAzureOpenAIClientPromptExample demonstrates how to interact with Azure OpenAI
 * services, specifically for sending prompts and processing responses, while
 * integrating with Thales CipherTrust Data Protection (CADP) for protecting
 * or revealing sensitive information (PII) within the prompt and response.
 * It loads configuration from a properties file and uses OkHttp for API calls.
 */
public class ThalesAzureOpenAIClientPromptExample {
        // A static Properties object to hold configuration loaded from thales-config.properties
        private static Properties properties;

        /**
         * Static initializer block to load properties from "thales-config.properties"
         * when the class is loaded. This ensures that configuration is available
         * before any other methods are called.
         * It throws a RuntimeException if the properties file cannot be found or loaded.
         */
        static {
                try (InputStream input =
ThalesAzureOpenAIClientPromptExample.class.getClassLoader()
                                .getResourceAsStream("thales-config.properties")) {
                        properties = new Properties();
                        if (input == null) {
                                // If the input stream is null, the properties file was not found
                                throw new RuntimeException("Unable to find udfConfig.properties");
                        }
                        // Load properties from the input stream
                        properties.load(input);

                } catch (Exception ex) {
                        // Catch any exceptions during file loading and wrap them in a
RuntimeException
                        throw new RuntimeException("Error loading properties file", ex);
                }
        }

        /**
         * The main method is the entry point of the application.
         * It sets up the Thales CADP helper, prepares the prompt by processing
         * sensitive data, sends the prompt to Azure OpenAI, and then processes
         * the AI's response, including checking for PII in the output.
         *
         * @param args Command-line arguments:
         * args[0]: Azure Cognitive Services Endpoint
         * args[1]: Azure Cognitive Services API Key
         * args[2]: OpenAI API Key
         * args[3]: OpenAI Endpoint URL
         * @throws IOException If an I/O error occurs during HTTP communication.
         */
        public static void main(String[] args) throws IOException {

                // Retrieve various configuration properties from the loaded properties object
                String keymanagerhost = properties.getProperty("KEYMGRHOST");
                String crdptkn = properties.getProperty("CRDPTKN");
                String policyType = properties.getProperty("POLICYTYPE");
                String showmetadata = properties.getProperty("SHOWMETADATA");
                String revealuser = properties.getProperty("REVEALUSER");
                System.out.println(" user  " + revealuser); // Print the reveal user

                // Convert showmetadata property to a boolean
                boolean showmeta = showmetadata.equalsIgnoreCase("true");

                // Initialize AzureContentProcessor. This class is assumed to handle
                // the detection and processing (protection/revelation) of PII within text.
                AzureContentProcessor cp = new AzureContentProcessor(properties);
                // Set Azure Cognitive Services endpoint and API key from command-line arguments
                cp.cognitiveservices_endpoint = args[0];
                cp.cognitiveservices_apiKey = args[1];
                // Set OpenAI API key and endpoint from command-line arguments
                String OPENAI_API_KEY = args[2];
                String OPENAI_ENDPOINT = args[3];

                // Initialize ThalesProtectRevealHelper (specifically
ThalesCADPProtectRevealHelper).
                // This helper interacts with the Thales CipherTrust Manager for cryptographic
operations.
                ThalesProtectRevealHelper tprh = new ThalesCADPProtectRevealHelper(keymanagerhost,
crdptkn, null, policyType,
                                showmeta);
                // Set additional properties for the Thales CADP helper
                tprh.revealUser = revealuser;
                tprh.policyType = policyType;
                tprh.policyName = properties.getProperty("POLICYNAME");

                // Initialize OkHttpClient for making HTTP requests to the OpenAI API
                OkHttpClient client = new OkHttpClient();

                // Define the input content string that contains a prompt and potentially PII
                String inputcontent = "        String inputText = Explain 'rubber duck debugging'
in one line to Kathi Wolf     Burney Circle  Mertzmouth     OH      24114   328.639.0623x3743
        becker.drury@hotmail.com      7/7/1992       5.32755E+15    15      203-04-0501";

                // Process the input content to protect/reveal PII before sending it to OpenAI.
                // The result `newPrompt` will have the sensitive data handled according to the
policy.
                String newPrompt = cp.processTextChunk(inputcontent, tprh);
                System.out.println(newPrompt); // Print the processed prompt

        // Construct the JSON request body for the OpenAI API call
        JSONObject json = new JSONObject();

        // Add "messages" array to the JSON, containing a single system message with the
processed prompt
        json.put("messages", new org.json.JSONArray().put(new JSONObject()
            .put("role", "system") // Role of the message sender (system, user, assistant)
            .put("content", newPrompt))); // The actual message content
        json.put("max_tokens", 200); // Set the maximum number of tokens for the AI's response

        // Create the request body with the JSON payload and specify content type as
application/json
        RequestBody body = RequestBody.create(json.toString(),
MediaType.get("application/json"));

        // Build the HTTP POST request to the OpenAI endpoint
        Request request = new Request.Builder()
            .url(OPENAI_ENDPOINT) // Set the URL for the request
            .addHeader("api-key", OPENAI_API_KEY) // Add the API key as a header
            .post(body) // Set the request method to POST and attach the body
            .build(); // Build the request

        // Execute the HTTP request and handle the response
        try (Response response = client.newCall(request).execute()) {
            // Check if the HTTP response was successful (2xx status code)
            if (!response.isSuccessful()) {
                System.out.println("Request failed: " + response); // Print failure details
                return; // Exit if the request failed
            }
            // Read the response body as a string
            String responseBody = response.body().string();
            System.out.println("Full Response: " + responseBody); // Print the full raw response

            // Parse the JSON response received from OpenAI
            JSONObject jsonResponse = new JSONObject(responseBody);
            System.out.println(jsonResponse); // Print the parsed JSON object
            // Get the "choices" array from the response, which contains the AI's generated text
            JSONArray choices = jsonResponse.getJSONArray("choices");

            // Check if there are any choices (generated responses)
            if (choices.length() > 0) {
                // Get the first choice (assuming only one is needed for this example)
                JSONObject choice = choices.getJSONObject(0);

                // Extract and print the content (the AI's generated text)
                JSONObject message = choice.getJSONObject("message");
                String content = message.getString("content");
                System.out.println("\nExtracted Content:\n" + content);
                System.out.println("\nNow checking for PII in output :\n" + content);

                // Process the AI's output content to reveal/protect any PII that might have been
generated.
                // An example email "sam@aol.com" is appended to demonstrate PII detection in
output.
               String cleanoutput = cp.processTextChunk(content + " sam@aol.com", tprh);

               System.out.println("clean output = " + cleanoutput); // Print the cleaned output

                // Extract and print content_filter_results, which indicate if the content was
flagged
                // for harmfulness, sexual content, self-harm, or violence.
                JSONObject contentFilterResults = choice.getJSONObject("content_filter_results");
                System.out.println("\nContent Filter Results:");

                // Iterate over the keys (filter categories) in content_filter_results
                Iterator<String> keys = contentFilterResults.keys();
                while (keys.hasNext()) {
                    String key = keys.next(); // Get the filter category name
                    JSONObject filterDetails = contentFilterResults.getJSONObject(key); // Get
details for this category
                    boolean filtered = filterDetails.getBoolean("filtered"); // Check if content
was filtered
                    // Get severity if available, otherwise "N/A"
                    String severity = filterDetails.has("severity") ?
filterDetails.getString("severity") : "N/A";
                    // Print the filter results for each category
                    System.out.println("- " + key + ": filtered=" + filtered + ", severity=" +
severity);
                }
            } else {
                System.out.println("No choices found in response."); // Message if no AI response
was generated
            }
        }
    }
}
```

Output:

```
- PII Entity: Kathi Wolf, Category: Person, Confidence: 0.85
 - PII Entity: Burney Circle, Category: Address, Confidence: 0.77
 - PII Entity: OH     24114, Category: Address, Confidence: 0.66
 - PII Entity: becker.drury@hotmail.com, Category: Email, Confidence: 0.80
 - PII Entity: 7/7/1992, Category: DateTime, Confidence: 0.90
 - PII Entity: 203-04-0501, Category: USSocialSecurityNumber, Confidence: 0.65
       String inputText = Explain 'rubber duck debugging' in one line to RU5DY2hhcg==-
17:10010001Hm0W bzIA   Burney Circle  Mertzmouth     OH     24114   328.639.0623x3743
       RU5DY2hhcg==-31:10010001m4Uly.x7r1S@8639OwP.uZq     7/7/1992      5.32755E+15    15
       203-04-0501
Full Response:
```

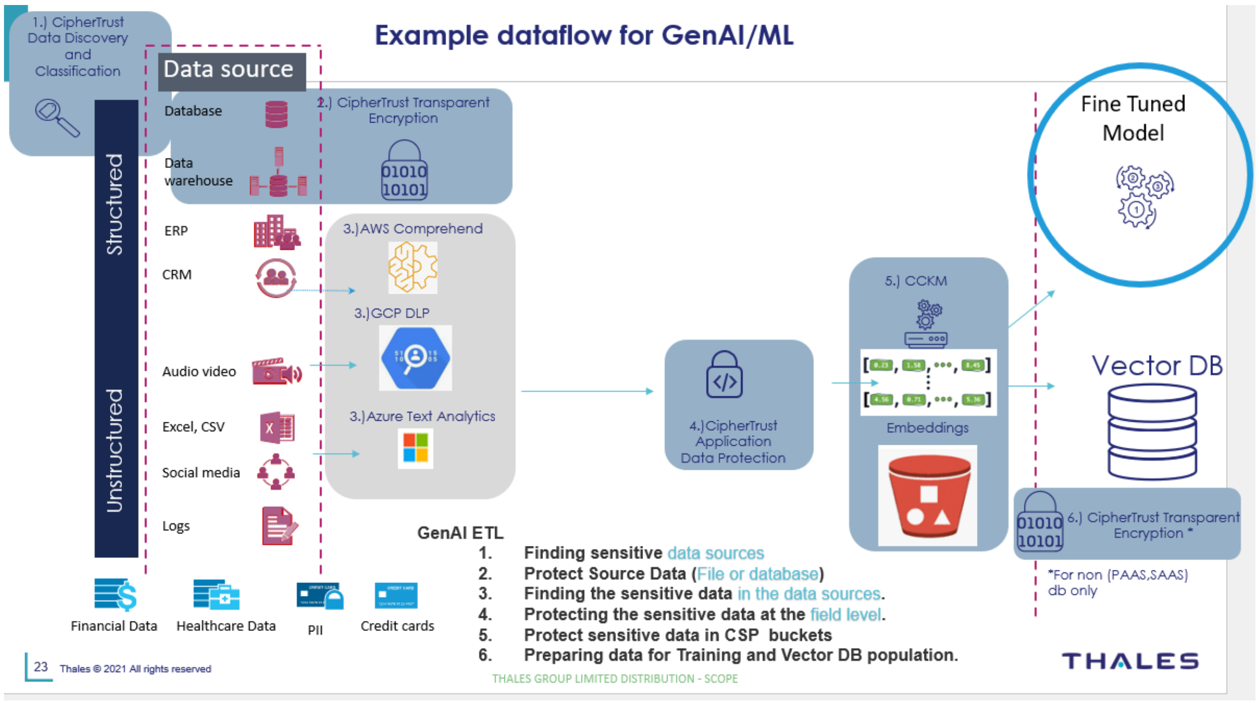
```
{"choices":[{"content_filter_results":{"hate":{"filtered":false,"severity":"safe"},"self_harm":{"
filtered":false,"severity":"safe"},"sexual":{"filtered":false,"severity":"safe"},"violence":{"fil
tered":false,"severity":"safe"}},"finish_reason":"stop","index":0,"logprobs":null,"message":{"con
tent":"Rubber duck debugging is a method where a programmer explains their code line-by-line to a
rubber duck in order to identify
errors.","role":"assistant"}}],"created":1752079998,"id":"chatcmpl-
BrSKkSmhpRfVbpi403MpHz5wpNH5Q","model":"gpt-4-turbo-2024-04-
09","object":"chat.completion","prompt_filter_results":[{"prompt_index":0,"content_filter_results
":{}}],"system_fingerprint":"fp_5603ee5e2e","usage":{"completion_tokens":26,"prompt_tokens":118,"
total_tokens":144}}

{"created":1752079998,"prompt_filter_results":[{"content_filter_results":{},"prompt_index":0}],"u
sage":{"completion_tokens":26,"prompt_tokens":118,"total_tokens":144},"model":"gpt-4-turbo-2024-
04-09","id":"chatcmpl-
BrSKkSmhpRfVbpi403MpHz5wpNH5Q","choices":[{"content_filter_results":{"self_harm":{"severity":"saf
e","filtered":false},"hate":{"severity":"safe","filtered":false},"sexual":{"severity":"safe","fil
tered":false},"violence":{"severity":"safe","filtered":false}},"finish_reason":"stop","index":0,"
message":{"role":"assistant","content":"Rubber duck debugging is a method where a programmer
explains their code line-by-line to a rubber duck in order to identify
errors."},"logprobs":null}],"system_fingerprint":"fp_5603ee5e2e","object":"chat.completion"}
```

```
Extracted Content:
Rubber duck debugging is a method where a programmer explains their code line-by-line to a rubber
duck in order to identify errors.

Now checking for PII in output :
Rubber duck debugging is a method where a programmer explains their code line-by-line to a rubber
duck in order to identify errors.
```

```
 - PII Entity: sam@aol.com, Category: Email, Confidence: 0.80
clean output = Rubber duck debugging is a method where a programmer explains their code line-by-
line to a rubber duck in order to identify errors. RU5DY2hhcg==-18:1001000wt9@FBl.52X

Content Filter Results:
- self_harm: filtered=false, severity=safe
- hate: filtered=false, severity=safe
- sexual: filtered=false, severity=safe
- violence: filtered=false, severity=safe
```

Overall workflow

The diagram below shows the entire process


Example dataflow for GenAI/ML
1.) CipherTrust Data Discovery and Classification
Data source
Structured
Unstructured
Database
Data warehouse
ERP
CRM
Audio video
Excel, CSV
Social media
Logs
2.) CipherTrust Transparent Encryption
3.) AWS Comprehend
3.)GCP DLP
3.)Azure Text Analytics
4.)CipherTrust Application Data Protection
5.) CCKM
Embeddings
Fine Tuned Model
Vector DB
6.) CipherTrust Transparent Encryption *
*For non (PAAS,SAAS) db only
Financial Data
Healthcare Data
PII
Credit cards
GenAI ETL
23 Thales © 2021 All rights reserved
THALES
THALES GROUP LIMITED DISTRIBUTION - SCOPE

GenAI ETL

1. **Finding sensitive data sources**
2. **Protect Source Data (File or database)**
3. **Finding the sensitive data in the data sources.**
4. **Protecting the sensitive data at the field level.**
5. **Protect sensitive data in CSP buckets**
6. **Preparing data for Training and Vector DB population.**

```
Here are some samples that can be used to implement the above.

1.) Run Thales DDC or other datasoruce discovery tool to find datasources that have sensitive
data.
2.) Use Thales CTE to protect those data sources.
3.) Run Thales DDC Report to export a list of datasources with sensitive data.
4.) If any of the sources are of file types excel,word,pdf run sample code
ExtractFilesFromDDCReport.java to generate a list of the datsources that need to be converted to
text.
5.) Run sample code ConvertToTextBatchProcessor.java to convert files found in step for to text
version.
6.) Depending on customer preference for CSP (AWS,GCP,Azure) run appropriate sample code
ThalesCSPProtectRevealBatchProcessor.java to find sensitive fields in datasources and create a
new output with sensitive data protected.
7.) Now load protected data into appropriate AWS,GCP, Azure training data upload.
```

Thales Protect/Reveal Options

The solutions provide in this document for field level protection are based on two application development offerings from Thales CRDP and CADP for Java.

1.)CRDP

CipherTrust RESTful Data Protection (CRDP) is a RESTful webservice that protects sensitive data using encryption, tokenization, or data generalization. CRDP is designed from the ground up to seamlessly fit with existing cloud and on-premise applications. CRDP is deployed as a container and performs a wide range of data protection functions on the sensitive data including data masking, pseudonymization, and redaction. CRDP is easy to deploy, configure, manage, and migrate.
https://thalesdocs.com/ctp/con/crdp/latest/index.html

2.) CipherTrust Application Data Protection for Java (CADP for Java)

Is a Java Cryptography Extension (JCE) provider that enables users to integrate the Key Manager's capabilities into their Java applications. CADP for Java is available in two variants:

- **Centrally Managed APIs**
- **Traditional APIs**

The examples provided in this document are based on Centrally Managed APIs.

This is our new set of APIs available from CADP for Java 8.18.1 release. CADP for Java offers simplified APIs (Protect/Reveal) to perform cryptography operations. CADP for Java integrates seamlessly with **Application Data Protection** on CipherTrust Manager, enabling organizations to centrally configure, manage, and enforce data-centric cryptographic policies in a reusable, human-readable format, providing flexibility and ease of management.
Protection policy (ciphers, keys, IV, tweak, and so on) defines how to protect the sensitive data. Whereas, Access policy determines who can view sensitive data and how (plaintext, ciphertext, custom masking format or redacted).
Centrally Managed APIs offer several advantages over traditional APIs, including:

- Developers do not need to understand cryptographic parameters (cipher, key, IV, Nonce, Tweak, and so on) to protect data as the Data Security Admins are responsible to handle configurations and policies.
- Each deployed application with CADP for Java is visible on CipherTrust Manager (providing a Single Pane of Glass view).
- Data Security Admins gain Crypto Agility, enabling real time changes to cipher, keys, and parameters.

https://thalesdocs.com/ctp/con/cadp/cadp-java/latest/admin/index.html#centrally-managed-apis

The code provided in this document provides examples using both the CRDP and CADP for Java Centralized API's. As you can see from the code there are two helper classes that make it easy to switch between either api.

```
// Initialize ThalesProtectRevealHelper. Two options are commented out, showing different
// implementations (CADP vs. REST). The REST implementation is currently active.
// ThalesProtectRevealHelper tprh = new ThalesCADPProtectRevealHelper(keymanagerhost, crdptkn,
null, policyType,showmeta);

// Using ThalesRestProtectRevealHelper for content protection and revelation.
ThalesProtectRevealHelper tprh = new ThalesRestProtectRevealHelper(crdpip, metadata,
policyType,showmeta);
```

Note: For each option there will be different properties required please refer to the code samples for detailed explanations.

Thales Protect/Reveal Sample Code

https://github.com/ThalesGroup/CipherTrust_Application_Protection/blob/master/crypto/java/ProtectRevealSample.java

https://github.com/ThalesGroup/CipherTrust_Application_Protection/blob/master/crypto/java/ProtectRevealBulkDataSample.java

Thales ReProtect or Rekey Sample Code

https://github.com/ThalesGroup/CipherTrust_Application_Protection/blob/master/crypto/java/ReprotectSample.java