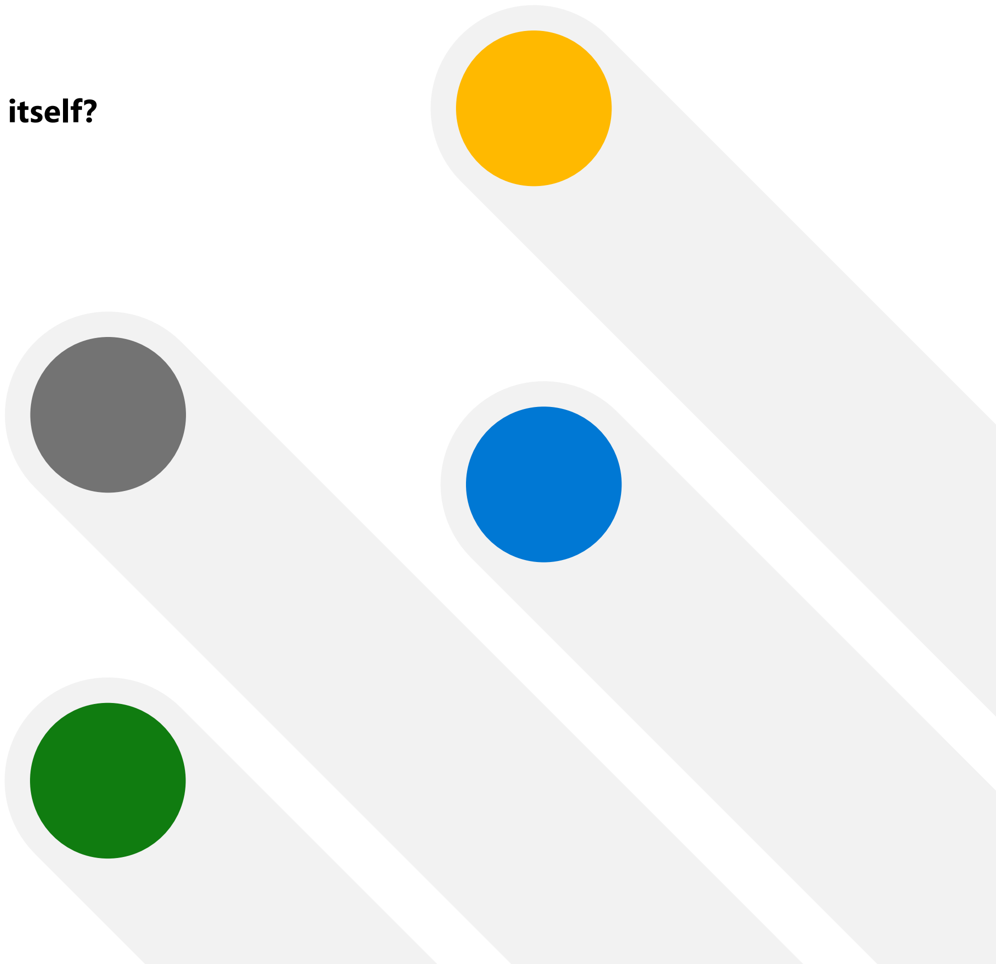
Microsoft Security

Double Key Encryption – Planning and Deployment Considerations

Michael Wirth
Ralf Wigand

Version: 2.1



1. Introduction

Protecting information from getting stolen or leaked is one of the fundamental goals since the very beginning. The times where putting printouts into a safe are over already for a long time, and today IT has taken over that task. By using strict permissions and various encryption technologies, IT has done in most cases a good job. But today's modern work style challenges these solutions by simultaneous editing, the need for limited sharing capabilities, and last, but not least, by outsourcing information into the hands of others, like cloud providers. Microsoft Purview Information Protection enables rights management in a more flexible way than it is possible with simple permission or encryption solutions, since the rights that are granted stick with the information, and both are protected by encryption technology. This allows some services to still work, like indexing, malware protection and others. Microsoft Purview Information Protection will be introduced in chapter 2.1.

Although cloud providers take massive invests into securing customers data not only from malicious attackers, but also from themselves, there are situations where some data must be secured in a way that even the provider can't get access to them. And here is where Double Key Encryption, or short DKE, enters the stage. DKE builds on top of Microsoft Purview Information Protection by adding a second key to the game, and that key is not managed by the provider but by the customer. This will be part of chapter 2.2.

At this point there is the moment for a warning: Just adding a second key is not sufficient per se to protect information, even if that second key is managed by the customer. If someone (hacker, insider, provider) can get access to that second key or take over an identity which allows access to the key, the whole thing about "double key" is obsolete. While safeguarding the second key's store itself is the obvious concern, protecting user and administrator identities from theft or forgery is of fundamental al importance as well.

When the same identity provider is used for accessing the document and for accessing the second key as well, there is only limited benefit in having two keys anymore. We will come back to this in multiple situations further down in this document, but we will also discuss solutions which can prevent or mitigate this kind of attack. A solid threat model must be in place first, focusing on what kind of attack and what kind of attacker the information should be protected against.

Since dealing with encryption keys (and now even with multiple of them) is not that easy and surely not a beginner's task, Microsoft offers a reference implementation of the DKE web service as open source on GitHub. It is a great way to learn and understand how the various components are connected and how data flows between services. The reference implementation will be explained in chapter 2.3.

But that is only the beginning of the journey. There are more aspects that should be taken into consideration when implementing the productive solution, and some of them will be covered in chapter 3, like External Key Management, isolating Cloud services from DKE data, and more.

When it comes to possible threats or security goals, chapter 4 gives some aspects, and it also covers operational considerations for the DKE service itself. After that, some possible flavors of how to embed DKE into a security solution are shown in chapter 5, each of them naming the goals that can be achieved by the model, but also which ones cannot be solved with the discussed model.

2. What is Double Key Encryption?

Double Key Encryption (DKE) is a feature in Microsoft Purview Information Protection (and the underlying Azure Rights Management Services (or Azure RMS in short), which are also part of the Purview product). It consists of client functionality (which is included in Microsoft 365 desktop applications like Word, Excel, PowerPoint, or Outlook) and a server component, further described as the DKE service. This DKE service is a web service application operated and controlled by the customer either as an appliance offered by partners or as a self-hosted web application (which in turn could be run in a customer's own data center on-premises or with a Cloud Service Provider).

2.1 Microsoft Purview Information Protection

Note: If you are already familiar with the mechanics of Azure RMS and know about the role and purpose of "Publishing Licenses" and "Use Licenses" you can easily skip this part and continue with "Double Key Encryption under the hood" in chapter 2.2 on page 9. Nevertheless, reading this part is encouraged since the following chapters will rely on some of these terms about DKE specifically.

Fundamentally, Microsoft Purview Information Protection[1] is a client-side encryption technology. In contrast to other solutions which are based on traditional PKI components, it automatically handles all the key management operations on behalf of end users. It does so by relying on the user's identity which is handled by the underlying directory service, Microsoft Entra ID, by using a user's or a group's SMTP address, as stored in the directory attributes "mail" and "proxyAddresses" as the significant identifier. When no mail address is set at all, Azure RMS falls back to reading the attribute "userPrincipalName" to establish the user's identity. In fact, the author of a document or e-mail message does not even have to know anything about the recipients other than their e-mail address (and inside an organization not even this, if the administrator defines sensitivity labels and associated permissions centrally).

[1] Further detail can be found here: [How Azure RMS works - Azure Information Protection | Microsoft Learn]

The second important aspect beyond the encryption itself is the usage restriction options ("rights") which originally gave the name to the Rights Management Services component: The author of a document can specify which rights or permissions the recipients of the file should have, for example printing, editing, saving, or set an expiry date after which the document can no longer be opened at all.

Finally, both encryption and rights travel with the file all the time.

For a better understanding of the following DKE discussion, let's look at the regular Microsoft Purview Information Protection flow first. The relevant components are:

- Microsoft Entra ID, a cloud-based directory and identity service, which provides a tenant[2] as the primary unit of isolation between multiple customers. It is the tenant's directory data like user and group mail addresses which control Purview's overall authorization process.
- Within the scope of this tenant there is an instance of Purview Information Protection, a cloud service that amongst other things provides sensitivity labels and encryption settings for the users of this tenant and
- Azure Rights Management, another cloud service which holds cryptographic key material for the tenant and performs the actual processing of cryptographic artefacts.
- The client computers may be located anywhere, i.e., in the cloud like virtual desktops, on-premises or in a mobile scenario.
- Finally, later paragraphs will introduce the Double Key Encryption web service (its actual location will be subject to discussion about threats and protection measures later).

[2] https://learn.microsoft.com/en-us/microsoft-365/solutions/tenant-management-overview

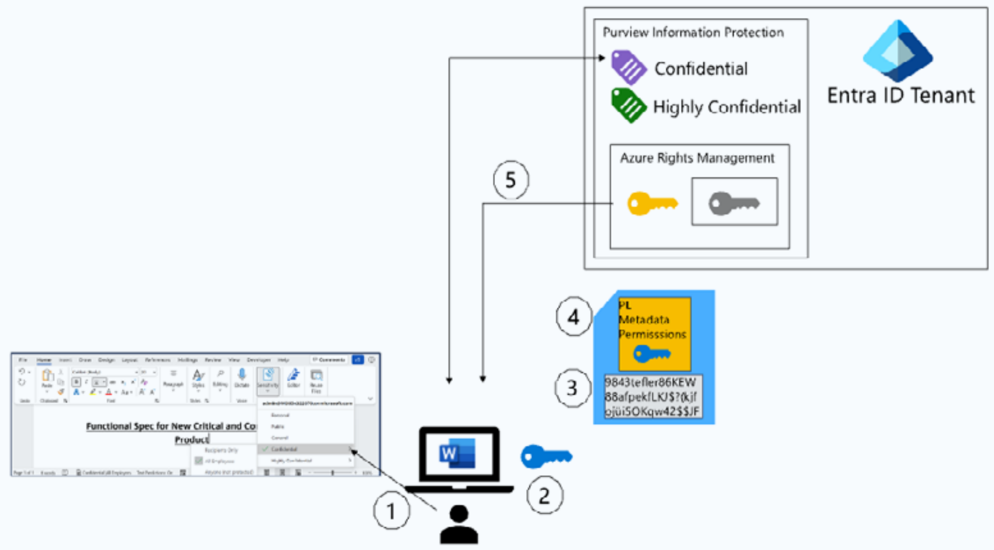


Figure 1: Microsoft Purview Information Protection flow for encryption.

Figure 1 shows a user interacting with a locally installed Office 365 application. The user authenticates inside the customer tenant against Microsoft Entra ID, the Microsoft cloud identity service.

The steps in Figure 1 are:

1. User creates sensitive content within the application and selects the sensitivity label “Confidential”.
2. The application learns that this label is applying encryption and creates a symmetric AES Content Key (in application memory). This Content Key is unique per document, or more precisely, per documentID, which is metadata that is put into the document when it is protected initially. Simply copying a given file obviously does not change its metadata, and documentID remains intact together with the rest of the file.
3. The file stream gets encrypted by this Content Key.
4. The Content Key is put into the Publishing License (PL), together with other metadata and the RMS permissions as defined in the template for “Confidential”. Note that steps 2-4 happen within the process memory of the desktop application itself.
5. The client reads (and caches) the public key of the RMS service in the tenant (depicted in yellow in Figure 1) and wraps the relevant parts of the PL with this key. The PL is put into the file together with the ciphertext. Note that there are also some metadata elements in a PL which are always in the clear, e.g., the URL of the RMS service.

At this point, the decryption of the original text would require access to the Content Key (the blue key in Figure 1), which itself needs to be decrypted by using the tenant's private key. This tenant's private key is managed by Azure Rights Management Services. Azure RMS will only decrypt the Content Key if the requesting party can provide a valid identity that matches the identifiers in the metadata of the Publishing License. The communication between the consuming user and the RMS is not only encrypted in transit (using a TLS connection), but also protected by the so-called End User License (the artefact which contains the Content Key) by encrypting it with a public key belonging to the user as described in the following step-by-step explanation of the decryption process:

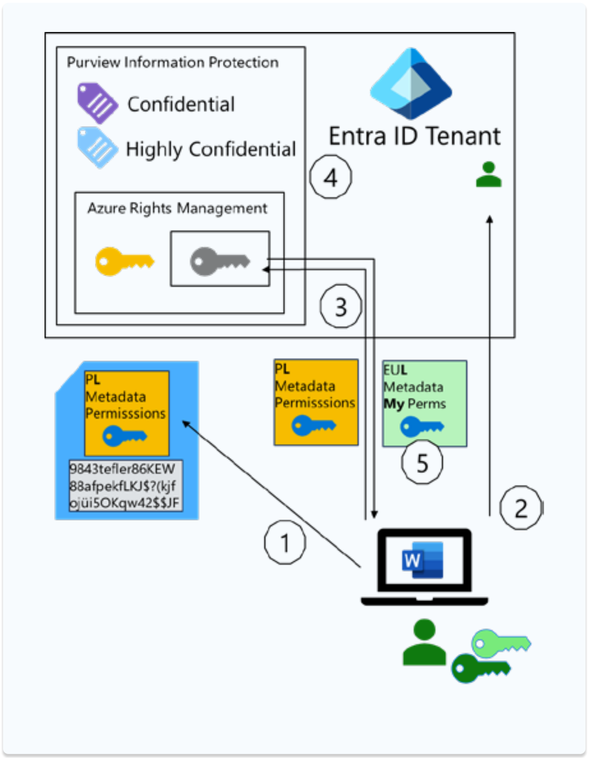


Figure 2: Microsoft Purview Information Protection flow for decryption.

Figure 2 shows the user accessing a protected document (the interaction with the local O365 app was left out for better reading). The user authenticates again inside the customer tenant against Microsoft Entra ID, the Microsoft cloud identity service.

The steps in Figure 2 are:

1. A consuming user tries to open the document within the application. As the application is RMS-aware, the PL can provide the URL of the RMS service whose public key protected the file's PL. The URL is not encrypted as mentioned in step 5 for Figure 1.

2. The consuming user authenticates to Azure RMS using Microsoft Entra ID. If the user successfully authenticates to this RMS instance for the very first time, an RSA key pair, called the GIC (Global Identity Certificate) gets created and stored in the local Windows user profile (the green key pair in Figure 2).

3. The client sends the PL (not the entire file content) to the RMS service, together with a proof of identity (including the user GIC's public key).

4. It is now up to the RMS service to decide if the user identity, based on the e-mail address in the GIC and other Microsoft Entra ID directory data like group memberships, is authorized to access the content and what the user's effective permission will be as specified by the publishing author in the PL. These effective permissions will be calculated in additive mode, i.e., if group A has been granted READ and group B has been granted PRINT, the resulting effective permissions for a user who is a member in both groups will be both READ and PRINT.
5. These effective permissions will be encoded into a new XML structure, the so-called Use License or End User License (EUL), together with the decrypted Content Key. This EUL is then protected with the GIC public key so that only the user in possession of the corresponding GIC private key can open the EUL.
6. The application receives the EUL, decrypts it by using the locally stored private key of the GIC, decrypt the original text by using the (now decrypted) Content Key and enforces the effective permissions of the consuming user as granted by the publishing author.

Note: RMS is not limited to end user devices: It can be implemented on cloud service instances acting as client towards Microsoft Purview Information Protection as well. With this, a cloud service can perform certain operations on the document like Indexing, Search, rendering files in Office Online, Co-authoring, eDiscovery, CoPilot and many more that need to process clear text.

2.2 Double Key Encryption under the hood

As the name already implies, DKE works on the principle of adding another round of encryption to the "Content Key", the blue key symbol in Figure 1, which all RMS-capable applications use to encrypt their file data stream as already described in 2.1. To achieve this, a DKE service holds another private/public key pair, the DKE key. To be precise, DKE can support one key pair per DKE-label, as the key name itself is part of the DKE URL naming scheme and each label may refer to exactly one URL.

After the file has been encrypted with the Content Key, but before the Publishing License (PL) gets protected by the tenants RMS key), the Content Key is additionally protected with the public portion of the DKE key and therefore "doubling" the encryption of the actual Content Key. The location of the DKE service is stored as a URL (as specified by the Purview administrator) within a Purview sensitivity label itself, alongside the other label parameters. After this, the "regular" RMS process continues with step 5 in Figure 1 by encrypting the (now already DKE-encrypted) Content Key together with the PL.

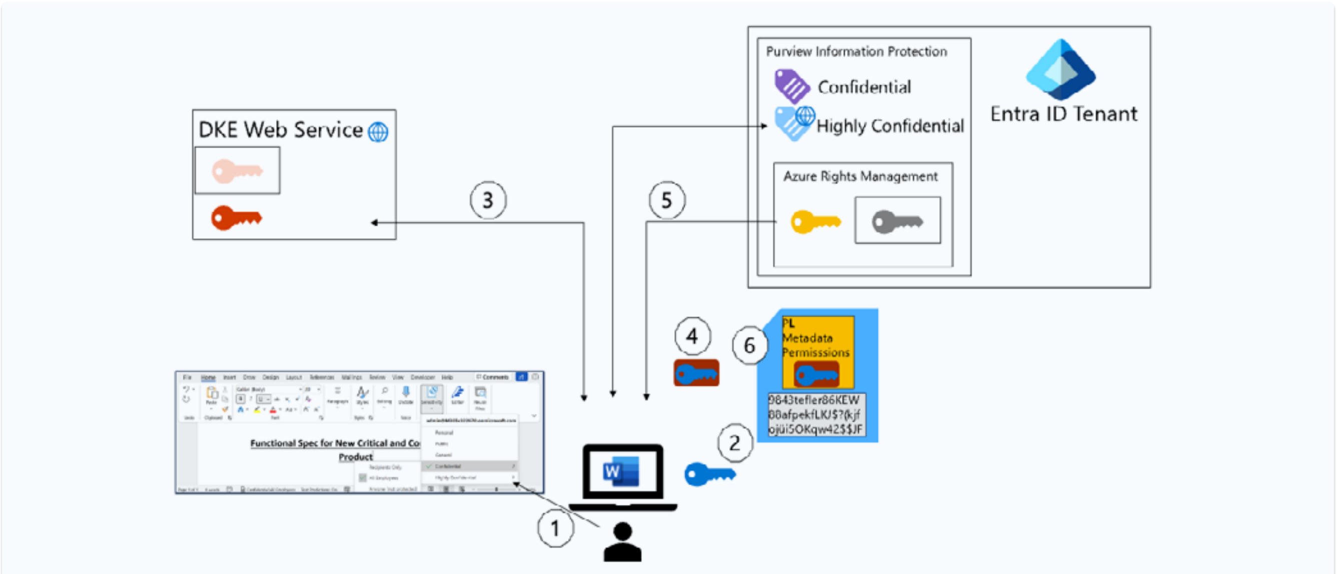


Figure 3: DKE flow for encryption.

Figure 3 shows a user interacting with a local installed O365 application. The symbol for the "Highly Confidential" label indicates that the label contains information (URL) where to find the DKE service.

The steps in Figure 3 are:

1. User creates sensitive content within the application and this time selects a DKE-enabled label (marked with a globe symbol in this and all following diagrams to represent a URL pointing to the service).
2. The Content Key gets created and the file stream is encrypted (as in step 3 for Figure 1).
3. The client application connects to the URL specified in the DKE label to retrieve the DKE public key.
4. The Content Key is wrapped using this public DKE key (the algorithm used is RSA-OAEP).

   The (now already encrypted) Content Key is put into the Publishing License (PL), together with other metadata and the RMS permissions as defined in the template. Note that steps 2-4 happen within the process memory of the desktop application itself.
5. From here on, the steps to create and protect the Publishing License (PL) are identical to the regular Microsoft Purview Information Protection flow of operations as described in Figure 1 steps 4 and 5: The client reads (and caches) the public key of the RMS in the tenant (depicted in yellow in Figure 3) and wraps the relevant parts of the PL with this key. The PL is put into the file together with the ciphertext.
6. The relevant parts of the PL are protected with the tenant's public key and saved to the file container.

For the respective decryption operation, the client application needs to call DKE's decryption web service endpoint after the Use License (EUL) was successfully acquired (using Microsoft Purview Information Protection's regular flow as shown in Figure 2). The wrapped content key contained in the Use License is sent to the DKE service, the user's authentication and authorization are checked, and - if accepted - DKE unwraps the Content Key and sends it back to the client over the established TLS-protected connection.

Figure 4 shows a user accessing a protected document (the interaction with the local Office 365 app was again left out for better reading). The user authenticates inside the customer tenant against Microsoft Entra ID, the Microsoft cloud identity service. Please note that the DKE service might require additional authentication that is not indicated in this figure but will be the topic in discussions further down.

The steps in Figure 4 are:

1. A consuming user tries to open the document within the application. As the application is RMS-aware, the PL can provide the URL of the RMS service whose public key protected the file's PL.
2. The consuming user authenticates to Azure RMS using Microsoft Entra ID. If the user successfully authenticates to this RMS instance for the very first time (as already described), an RSA key pair, called the GIC (Global Identity Certificate) gets created and stored in the local Windows user profile (the green key pair in Figure 4).
3. The consumer sends the PL to the Azure RMS service and, if authorized successfully, receives the End Use License EUL.

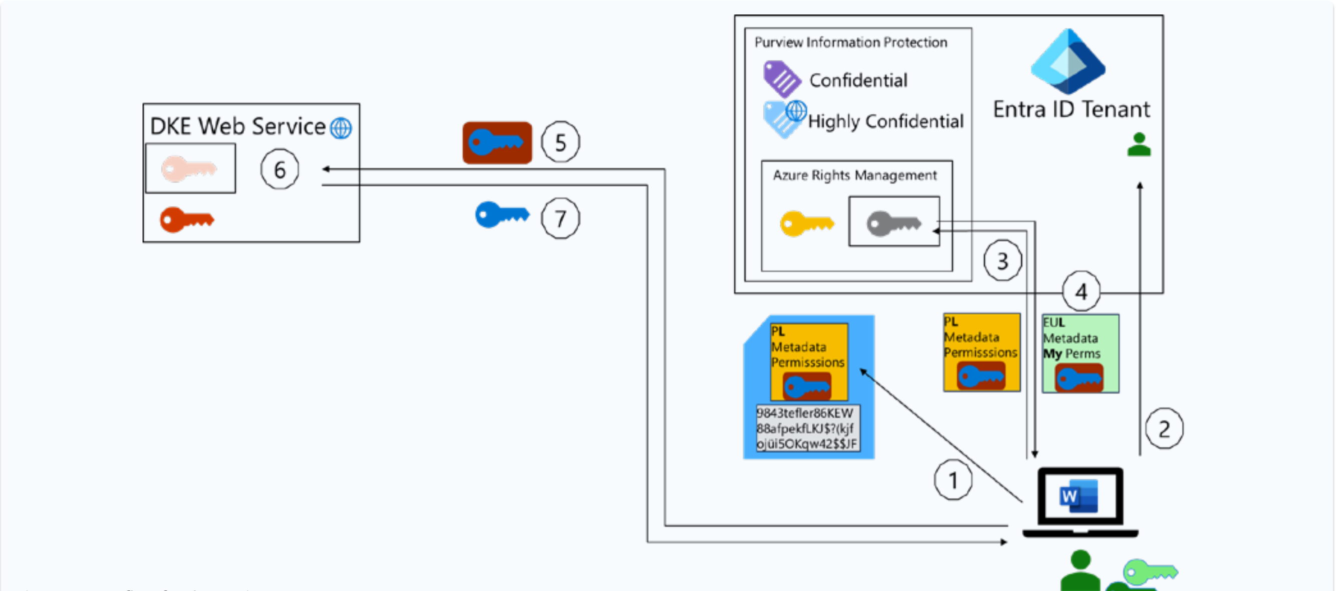


Figure 4: DKE flow for decryption.

4. While the rest of the EUL is readable to the user (who is in possession of the GIC's private key portion), the Content Key is still wrapped with the DKE public key.
5. The wrapped Content Key is sent to the DKE service (its URL is known since it is part of the EUL metadata).
6. The caller's identity is validated (for details see next chapter), and if the authorization requirement (as configured in DKE's configuration) is satisfied, the DKE service, using its private DKE key portion, unwraps the Content Key.
7. Finally, the Content Key is sent back to the client. The original content can be decrypted and rendered by the application, enforcing the effective permissions of the consuming user as granted by the publishing author.

2.3 Reference Implementation for the DKE web service

To get yourself familiar with all the involved keys, encryption and decryption steps, Microsoft is providing a reference implementation for the DKE web service. As Microsoft expects that customers who consider using DKE are looking for a high degree of security and trust based on transparency, the reference implementation was published as Open Source on GitHub[3]. This implementation is built on top of .NET Core, which itself is Open Source and makes it possible to run the service not only on Windows but also on Linux.

This decision came with a drawback, though: The reference implementation will only include a file-based approach for the storage of the DKE keys. While this might be secure enough for a lot of cases, there are other scenarios asking for protection against additional threat vectors, like for instance, storing the key inside an HSM, a Hardware Security Module (a physical computing device that safeguards and manages digital keys for strong authentication and cryptography operations). But integrating a HSM as a key store in the reference code would require a Cryptographic Service Provider in Windows, which typically has a dependency on the actual HSM vendor or would require a PKCS#11 based approach under Linux. Consequently, Microsoft refrained from selecting a specific HSM solution, especially as it turned out that some HSM partners had announced to offer DKE service implementations as part of their own product offerings.

Another aspect of the reference implementation besides transparency by Open Source was the need for ease of deployment. It should be possible to get a DKE instance up and running without significant infrastructure investment. So, Microsoft's DKE documentation proposes to use Azure App Services (a serverless PaaS service for hosting web applications), while the DKE application itself is registered in Microsoft Entra ID as an Enterprise application using

[3] GitHub - Azure-Samples/DoubleKeyEncryptionService: Download, install, and set up the Double Key Encryption service for Microsoft 365.

Open ID Connect (OIDC) as a protocol and consuming JSON Web Tokens (JWT) issued by a Microsoft Entra ID tenant. The "trust relationship" towards Microsoft Entra ID is established by putting the tenant information into a local configuration file. With this, the DKE service is using the same ID provider (Microsoft Entra ID) as Purview Information Protection itself and thereby provides an easy path to integrate with existing Zero Trust technologies like password-less authentication and conditional access which are both available as part of Microsoft Entra ID. On the other hand, it should be mentioned that, as the service uses the same ID provider (Microsoft Entra ID in this case) for both Purview and DKE service authentication, the setup does not utilize a cloud-independent identity infrastructure. Successful theft or forging of a user's access token could give an attacker access to both Purview and DKE.

3. General considerations

3.1 External Key Management

One common misconception is that simply putting the Key Management Solution into the on-premises data center would protect against access by the Cloud Service Provider.

However, when executing cryptographic operations in the cloud (in contrast to operations on the user's device), one would still require granting the identity of some cloud service the permission to access the cryptographic API surface. Even if storing the key only on-premises in files (like in the reference implementation discussed in chapter 2.3) – or within an HSM – and adding an external Identity Provider to the infrastructure, the cloud service would then still need to securely store its own credentials for authentication. This would again have to happen within the service's own scope or domain in the cloud. Strictly speaking, a cloud service acts as a client towards a key management system here (using its own service identity and credentials).

Using client encryption from the end user's device, helps avoid this catch-22 scenario: here, it is the end user whose identity is validated by a web service in front of the key store (and this is the pattern for DKE as well when a decryption operation is requested).

3.2 Single Sign On

It should be obvious that enforcing strong authentication and a compliant, healthy device are highly desirable when it comes to interacting with a cryptographic key management system like RMS or DKE. At the same time, ease of use, i.e., a seamless SSO experience is crucial for end user acceptance. Excessive prompting during the opening of an Office document is not only annoying to the user, but it can also be a technical challenge for the application itself to react to potential error conditions during the authentication flow. Therefore, providing seamless access to the DKE web service is one fundamental aspect of the overall data security and threat model. The capabilities of the Office client applications with regards to SSO and multiple user identities need to be kept in mind as well.

3.3 Cloud Service Access to DKE

Microsoft cloud services cannot interact with DKE-protected content for two reasons:

- First, support for DKE from a caller's perspective is a function implemented by the Microsoft Information Protection Software Development Kit[4] (MIP SDK). This SDK is a client-side component and is therefore not used by services like SharePoint Online or Exchange Online.
- Second, the cloud services' identity is not in the list of authorized principals of the DKE service (and the customer is in control of this configuration).

The consequences are significant. As Cloud Services cannot read the content of the documents,

› these documents are not indexed and cannot be searched.

› the content cannot be discovered (like in eDiscovery or Data Subject Rights requests).

› the documents' content cannot be rendered in Office Online.

› these files cannot be edited collaboratively.

› they always need to be downloaded to the client to be opened and edited.

› server-side anti-malware will not work.

› traditional Data Loss Prevention will also not work.

This might be an inconvenience that customers are willing to accept (given the improved data security), but it also brings significant risks requiring customers to choose wisely before applying DKE to a document. Sometimes there is only a small percentage of files, the "crown jewels", which need DKE encryption, and only a part of the users require a DKE-enhanced sensitivity label.

[4] Microsoft Information Protection SDK documentation | Microsoft Learn

3.4 Feature Availability

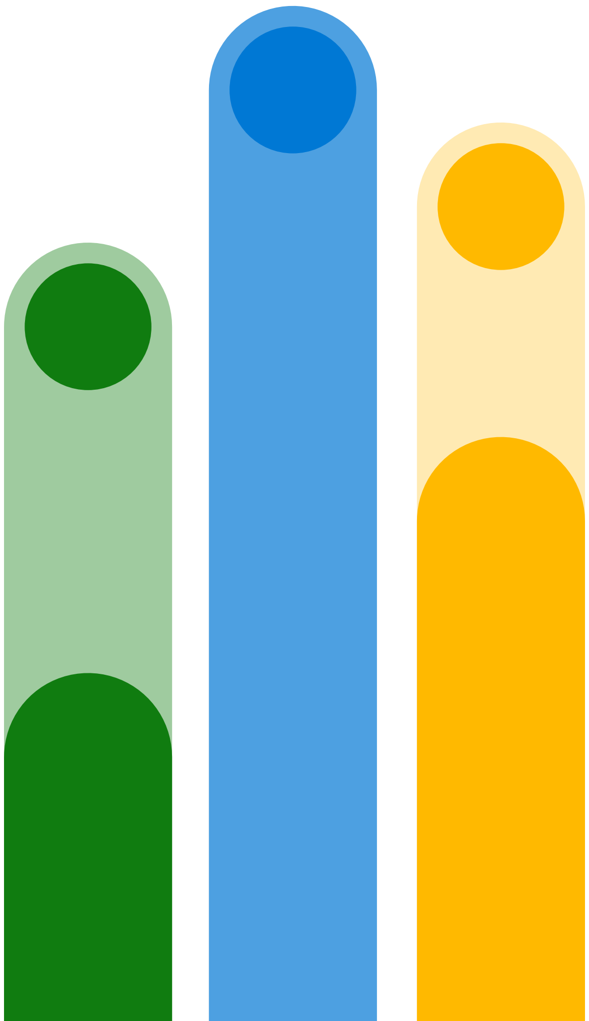
Today, DKE is available within Office 365 Apps for Windows as well as in the MIP SDK. Support for other client OS platforms is planned, starting with Office Apps on iOS, Mac and Android (expected in Summer 2024).

Starting April 2024 support for the Office plug-in within the Azure Information Protection client package has ended, so it is no longer possible to use Office perpetual installations with DKE.

The identity model for Office 365 Apps is built around modern authentication protocols with Microsoft Entra ID. Any additional authentication component protecting DKE (as mentioned above) should integrate seamlessly with the user's desktop session or be established even "lower in the stack", e.g., at the network layer. In most cases Office 365 Apps won't be aware of such additional authentication, and therefore won't be able to prompt the user to authenticate or reauthenticate. This might lead to an unsatisfying end user experience. For example, if access to the DKE service requires a VPN, the Office 365 App will not be able to contact the DKE service if the VPN connection is down. Since the Office application has no idea why the DKE service cannot be reached, it would simply raise an error dialog, but it will not prompt the user to login to the VPN.

3.5 Authorization

Finally, it is important to keep in mind that authorization in the context of RMS permissions always resides with the Azure RMS service, even when DKE is used. Authorization within and to DKE means that the user is allowed to invoke DKE's decryption operation, only. This can result in scenarios where a user is allowed to use DKE, but while enforcing the permissions granted by the original creator of a document, the application leaves the user with no permissions. The stricter authorization wins.



4. How to protect the DKE service itself?

4.1 Threats and Security Goals

In this chapter we will focus on some top threats and resulting objectives. There are for sure more threats to consider or some threats need a higher priority depending on the specific customer infrastructure or requirements.

Sensitive content automatically gets indexed by Microsoft Online Services

The objective here is to prevent access to a subset of documents by Microsoft Cloud Services to avoid data being used by the index and search infrastructure. As already mentioned in chapter 3.3, Microsoft cloud services cannot interact with DKE-protected content. The reference implementation and any of the flavors shown in chapter 5 would fulfill the objective. This goal is referred to as *Prevent indexing certain documents by Microsoft Cloud Services*

Predefined label permissions are too wide for a certain set of sensitive documents

Here the objective would be to further reduce the user audience which can access certain documents by adding another layer of control to a given subset of sensitivity labels. Again, all flavors shown in the following chapter would be able to fulfill this objective since they all add some additional control like VPN, IPSec etc. Even the authorization logic in DKE itself could be adjusted to achieve this goal, which is referred to as *Further reduce the user audience by adding another layer of authentication or authorization control*

As there is yet no label-based conditional access available any authorized user may consume sensitive content regardless of location, device health or authentication strength

Objective here would be for example to add certain restrictions on the accessibility of the documents that normally (i.e. for Browser applications) would be achieved by using Conditional Access. By establishing a higher set of requirements for authentication strength, device state and potentially client network location to access highly sensitive content this goal can be achieved, for example with model A by restricting network access to the document by a VPN. The goal would be *Establish a higher set of requirements for authentication strength, device state etc. for highly sensitive content*

A stolen or forged Entra ID token can be used as valid identity proof

If the objective is to protect against faked Entra ID tokens, one way to achieve this, can be to provide a different or an additional, second layer of authen¬tication and authorization independent from Microsoft Entra ID tenant as shown for example in Model C (Federation) or Model D (VPN with Windows Server AD), resulting in the goals:

› *Provide a **different** layer of authentication and authorization or*

› *Provide an **additional** second layer of authentication and authorization*

Customers server administrator may tamper with configuration files (incl. copying of private keys)

If the IT department is not fully trustworthy, an understandable objective would be to protect the DKE service (especially the key store) from one's own IT staff. Model F (Vendor HSM) for example protects the keys by storing them inside an HSM. The goal would be *Protect the DKE service (especially the key store or memory) from my own IT staff*

A cloud service provider may tamper with configuration files (incl. copying of private keys)

To prevent the DKE service from being tampered by anyone outside the organization (like a Cloud Service Provider or CSP), the service and the key store can be protected by using additional safeguards out of reach by the CSP, like shown in Model B (Confidential VM) or Model F (Vendor HSM). As a consequence, *Protect the DKE service (especially the key store or memory) from the hosting CSP*

An attacker could use Man-in-the-Middle (MitM) techniques to sniff network traffic between client and DKE service to access content keys

To prevent this from being possible, the communication link between client and server needs to be protected, ideally with a separate mutual authentication and encryption layer, for example as shown in Model E (IPSec Server Isolation). The goal is stated as *Protect the communication link between client and server with a separate mutual authentication and encryption layer (IPSec or mTLS)*

4.2 Other Operational Considerations

Potential DKE customers will probably face a “make-or-buy” decision. There is a couple of vendors selling DKE solutions in an “appliance”-like form factor. Especially if customers prefer HSM-based key storage, building a solution from scratch is often not an option.

A couple of criteria still apply, regardless of a commercial solution or a self-made solution based on the GitHub reference. In the very beginning, the question should be: “What threats or which actors should the DKE service be protected from?”, examples could be the Cloud Service Provider, external partners or vendors, malicious insiders or any party that can forge valid tokens etc. With the answer to this fundamental question, some more questions typically occur, like:

- Where can the system run? On-premises, in the customer’s data center or with a cloud provider?
- Which platform is it running on? A virtual machine? A container-based architecture?
- Which operating system should be used? What is the web application server of choice?
- How is it operated, monitored, patched?
- What identity model can be used to authenticate and authorize the users? Which other security features can be and should be used for the service?
- Does the access to the service require additional protection, e.g., VPN, or can the service be accessed over the internet at all?
- What level of availability does the service require? Does the SLA require a high availability solution like a failover environment or redundant Internet connections?

5. Possible flavors

As a paper like this cannot provide an exhaustive discussion on all the possible aspects (and, as with all design decisions for an IT system there are probably many), it is still possible to give some guidance by using a generic threat model to align potential threat vectors with corresponding design elements.

The following chapters describe different approaches, preceded with goals and non-goals, meaning that the model suits the goals but will not address the non-goals[5]. These approaches do not necessarily build upon each other, or are better or stronger on a linear scale, but rather should be understood as building blocks as you will recognize while reading. For all following models the reference implementation as described in 2.3 was taken as a "baseline". By combination of those blocks other scenarios are also possible, depending on the discussions and answers to the requirements as mentioned in chapter 3 or 4.

As a baseline these two goals can be achieved by implementing DKE even in the basic reference implementation:

- Prevent indexing certain documents by Microsoft Cloud Services
- Further reduce the user audience by adding another layer of authorization control

Below implementation "models" will only list additional goals on top of the above.

5.1 Model A (DKE Network Isolation)

Threat Model Goals

- Establish a higher set of requirements for authentication strength, device state and potentially, client network location to access highly sensitive content.

Non-Goals

- Provide an additional second layer of authentication and authorization
- Categorically excluding external users who do not belong to my organization.
- Protect the DKE service (especially the DKE private key store) from my own IT staff or from my Cloud Service Provider.

[4] Please note that the list of goals/non-goals is driven by the customer's threat model and security requirements and is certainly not exhaustive.

Approach

- Isolated network segment (might be in the cloud, but preferably on premises)
- Windows Server (as VM) with IIS web server in this isolated network, running the DKE service with reference implementation installed.
- Network isolation can be achieved by putting a VPN gateway in front of the DKE service as shown in the diagram below.
- Identity based on Microsoft Entra ID with a conditional access policy to require authentication strength, device compliance and (possibly) network location.

This model uses the reference implementation as described in chapter 2.3, adding extra protection to the DKE service by using a VPN to restrict access to only defined networks. The VPN gateway in Figure 5 uses Microsoft Entra ID for authentication (via SAML2 or OAuth2 protocol). This allows not only SSO on Windows as well as other device platforms, but it also makes conditional access policies possible to require additional authentication strength, device state and device location.

A similar strategy, however, with equally good or even better cross-platform support, might be Microsoft Entra ID Global Secure Access, which is a modern SASE-style solution (this sometimes also gets categorized as a "Zero-Trust Networking" approach). Again, seamless SSO with Microsoft Entra ID and integration for Conditional Access Policy are important properties for such a solution.

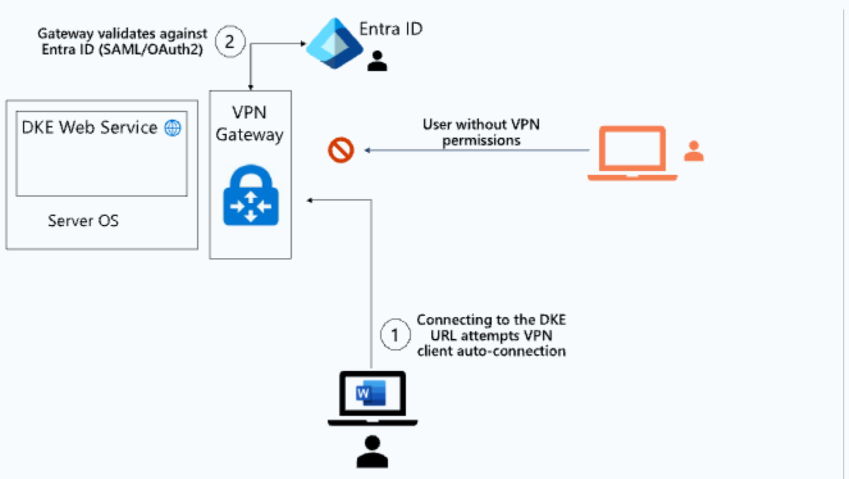

Gateway validates against Entra ID (SAML/OAuth2)
2
Entra ID
VPN Gateway
DKE Web Service
Server OS
User without VPN permissions
1
Connecting to the DKE URL attempts VPN client auto-connection

Figure 5: Additional network isolation with RADIUS service to Microsoft Entra ID

5.2 Model B (Confidential VM)

Threat Model Goals

- Protect the DKE service (especially the key store or memory) from my own IT staff
- Protect the DKE service (especially the key store or memory) from the hosting CSP

Non-Goals

- Adding a second layer of authentication and authorization independent from Microsoft Entra ID.
- Establish a higher set of requirements for authentication strength, device state and potentially, client network location to access highly sensitive content.

Approach

- VM with confidential computing in the cloud

Protecting "data-in-use" from access by privileged actors, be it a server administrator or a cloud service provider, is challenging, since an application needs to compute cleartext data (if we ignore homomorphic encryption cases for now). However, during recent years, a new model has appeared which uses a CPU's capability to encrypt and isolate process memory or the entire memory space of a virtual machine to achieve protection from the underlying hardware, OS, or hypervisor layers. This model is known as "Confidential Computing" and it is a joint effort by CPU vendors, cloud providers and various other players in the market, who participate in the Confidential Computing Consortium with the common goal to further decrease the size of the trusted computing base (TCP) and the need for trust into a cloud service provider, resulting in increased assurance of information protection for sensitive workload and data assets during processing.

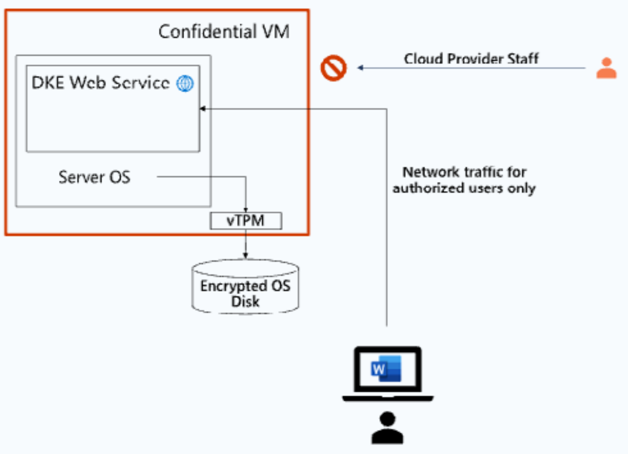


Figure 6: Confidential VM

However, the security of this model does not rely on memory isolation alone. It is paramount that a confidential VM does not get launched on hardware without support for memory encryption, which is achieved by a so-called "Attestation". Furthermore, there may be secrets on the disk as well, so disk encryption is an obvious solution to make sure that the OS disk is not tampered with before, during or after execution of the VM. It is the task of the virtual TPM provided within the guest firmware to make sure that the VM itself can decrypt the disk and boot. Further information on Azure confidential VMs can be found online[6].

It is important to note that Confidential Computing addresses the typical access vectors besides the regular network communication to DKE service (like dumping process memory by the Hypervisor or host hardware layer).

All considerations about authorized network access to DKE as discussed in this paper still apply.

5.3 Model C (Federation)

Threat Model Goals

- Provide a **different** layer (or source) of authentication and authorization

Non-Goals

- Adding a **second** layer of authentication and authorization independent from Microsoft Entra ID.

Approach

- Use identity from Microsoft Entra ID with Conditional Access for Purview Information protection plus an ADFS instance as a claims instance for DKE.

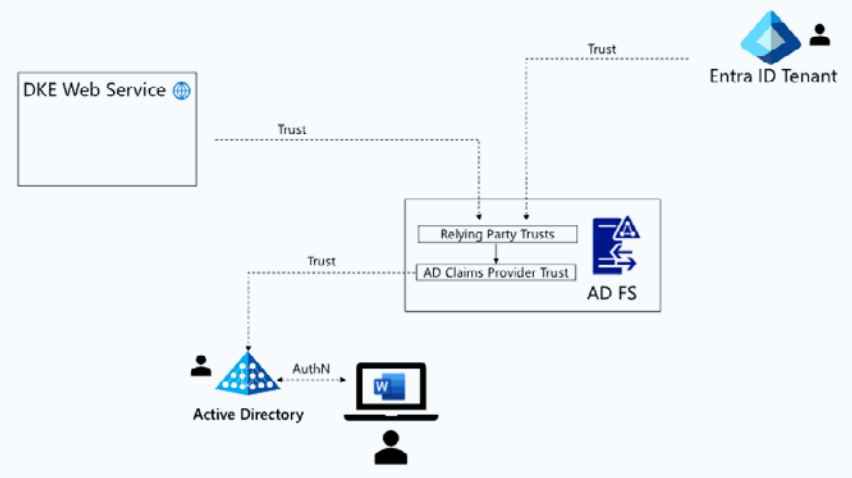


Figure 7: Using ADFS instance.

6 https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview

Model C uses an ADFS instance with federation trust (in ADFS, this is called a "Claims Provider Trust") to an on-premises Active Directory Forest. Furthermore, it uses a "relying party trust relationship" to each of DKE and Microsoft Entra ID.

It should be noted that user identity is still rooted in a single directory service, the local Active Directory and not the Microsoft Entra ID. A user gets access tokens both to Entra ID and to the DKE service after proving their identity to AD and after receiving an initial Kerberos ticket. This ticket is then seamlessly exchanged into the respective token formats for both relying parties by the ADFS instance.

For many customers who still rely on Active Directory and ADFS, this may address the threat vector of a forged Microsoft Entra ID token (although, ultimately, it just shifts the very threat to on-premises). One of the complications is that Entra ID Conditional Access can no longer be used for the DKE application as DKE is no longer registered in Entra ID (there is still a possibility to use some ADFS functionality here, but this is not nearly as rich as with Microsoft Entra ID). ADFS also supports using its token issuance pipeline as another authorization layer, however, due to the scope of this paper we must defer to the ADFS product documentation[7] here.

5.4 Model D (VPN with Windows Server AD)

Threat Model Goals

- Provide an additional second layer of authentication and authorization

Approach

- Use a VPN gateway for network isolation with authentication and authorization done by an on-premises Active Directory (AD) Forest

This model is another variant that addresses the aspect of a second identity provider. Here it could be a VPN with authentication and authorization against an on-premises AD as shown below:

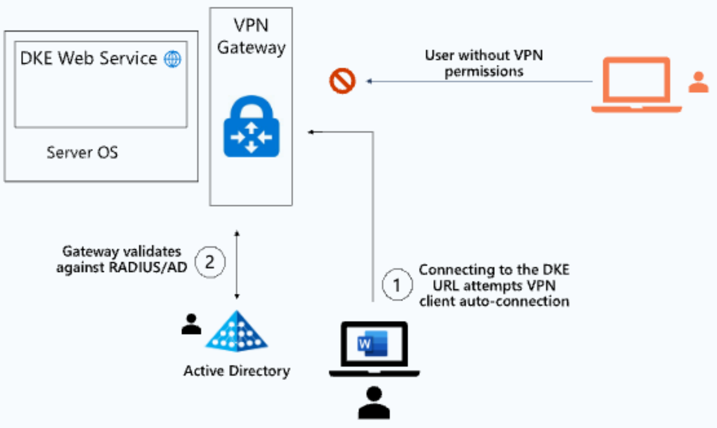

DKE Web Service
Server OS
VPN Gateway
User without VPN permissions
Gateway validates against RADIUS/AD
2
1
Connecting to the DKE URL attempts VPN client auto-connection
Active Directory

Figure 8: Additional network isolation with RADIUS service to on-premises AD.

[7] AD FS OpenID Connect/OAuth concepts | Microsoft Learn

Note that in Figure 8 (in contrast to Figure 5) the VPN gateway points to a RADIUS service which in turn authenticates against Windows Server Active Directory. For future mobile DKE clients this may not be optimal to achieve a seamless SSO experience. And ideally, the VPN connection gets established automatically whenever the client PC tries to reach the DKE URL. So, integrating the VPN with Microsoft Entra ID may be the easier approach as discussed in Model A (DKE Network Isolation).

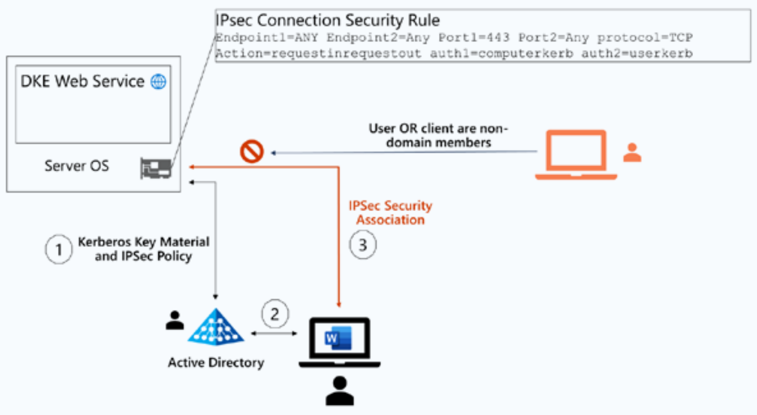
5.5 Model E (IPSec Server Isolation)

Threat Model Goals

- Provide an additional second layer of authentication and authorization
- Protect the communication link between client and server with a separate mutual authentication and encryption layer (IPSec or mTLS)

Approach

- IPSec server isolation (or mutual TLS)


IPsec Connection Security Rule
DKE Web Service
Server OS
User OR client are non-domain members
IPSec Security Association
3
1
Kerberos Key Material and IPSec Policy
2
Active Directory

```
Endpoint1=ANY Endpoint2=Any Port1=443 Port2=Any protocol=TCP
Action=requestinrequestout auth1=computerkerb auth2=userkerb
```

Figure 9: Using IPSec and on-premises AD.

Using IPSec can be a great complementary solution for the DKE server machine due to a couple of properties of the IPSec implementation in Windows[8]:

- The IPSec policy offers both Kerberos and x.509 certificates as authentication options. Since Kerberos requires the machine and user accounts to be part of the local Active Directory Forest, the AD Kerberos infrastructure already takes care of the key management.
- Certificates on the other hand require additional Public Key Infrastructure components but could potentially allow non-domain members and even some other OS platforms to participate. In principle, a PKI does not need to be integrated with Active Directory, although in practice the automated key management in AD over a manual key distribution approach might be the preferred solution.
- Storing machine or user certificates utilizing the TPM-backed "Platform Key Storage Provider"[9] in Windows is another attractive property of a certificate-based solution.
- he IPSec Policy on the server could be extremely simple, requiring successful user and machine authentication (from the same AD forest) during the establishment of IPSec transport mode security association for all remote endpoints (all IP addresses, TCP port 443).

[8] IPsec Configuration - Win32 apps | Microsoft Learn
[9] Trusted Platform Module (TPM) fundamentals - Windows Security | Microsoft Learn

5.6 Model F (Vendor HSM)

Threat Model Goals

- Protect the DKE service (especially the key store or memory) from my own IT staff
- Protect the DKE service (especially the key store or memory) from the hosting CSP

Non-Goals

- Adding a second layer of authentication and authorization independent from Microsoft Entra ID. Note that depending on the selected vendor solution (see below) this could also be fulfilled, but not simply by using a hardware-based keystore per se..

Approach

- Procure an HSM partner appliance with integrated DKE service offering.

There is a group of Microsoft partner companies like Thales, Utimaco or Entrust (to only name a few, some more can be found in the Azure Marketplace[10]), offering a variety of HSM based solutions with integrated DKE functionality. Although practically all of them have an HSM device in common, they will differ in the details about implementation, functionality, support for other security features, cost and many more. This paper will therefore discuss only some common considerations when selecting a partner HSM solution for DKE.

The decision about where to host the module will be again driven by the desired protection level towards Microsoft as the Cloud Service Provider. Some partners provide a managed virtual appliance (with a cloud service front end) which might put the solution inside the Microsoft cloud, while others (or even the same partners with a different product) also offer physical HSM on-premises. For these, physical security measures must be considered like building facilities, or locked racks in the data center, special surveillance, Hardware-SLAs, or further product features just to name the most prominent examples.

Just like in the other cases discussed above, network connectivity and security options will play an equally important role when it comes to selection of a suitable solution.

And, depending on the partner solution, those appliances (or their front-end web part) will offer one or more authentication methods, like Microsoft Entra ID, local AD forests, certificates or maybe even something else. As discussed above, using a Microsoft Entra ID based authentication method to access the service surely has benefits with regards to usability, but DKE will then rely again on Microsoft Entra ID as one single identity source (despite the two independent keys).

We strongly recommend reaching out to those partners to get more clarity on which solution fits your security needs and potentially the regulatory requirements for your business.

[10] https://azuremarketplace.microsoft.com/en-us/marketplace/apps/category/security

6. Summary

Obviously, there is no easy "one-size-fits-all" solution. As discussed in the previous chapters, the solution can be architected with different modules as shown in the models in chapter 5.1 to 5.6 . There are surely even more possibilities to reach the required goals. As already stated above, the project of deploying DKE should start with answers to essential questions like those provided in chapter 4.

7. Figures, Tables and Links

Figures used in this document:

Links used in this document:

How Azure RMS works - Azure Information Protection	https://learn.microsoft.com/en-us/azure/information-protection/how-does-it-work
Tenant management for Microsoft 365 for enterprise	https://learn.microsoft.com/en-us/microsoft-365/solutions/tenant-management-overview
GitHub - Azure-Samples/DoubleKeyEncryptionService: Download, install, and set up the Double Key Encryption service for Microsoft 365.	https://github.com/Azure-Samples/DoubleKeyEncryptionService
Microsoft Information Protection SDK documentation	https://learn.microsoft.com/en-us/information-protection/develop/
About Azure confidential VMs	https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview
AD FS OpenID Connect/OAuth concepts | Microsoft Learn	https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/development/ad-fs-openid-connect-oauth-concepts
IPsec Configuration - Win32 apps	https://learn.microsoft.com/en-us/windows/win32/fwp/ipsec-configuration
Trusted Platform Module (TPM) fundamentals - Windows Security	https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/tpm-fundamentals#tpm-based-certificate-storage
Azure Marketplace - Security	https://azuremarketplace.microsoft.com/en-us/marketplace/apps/category/security