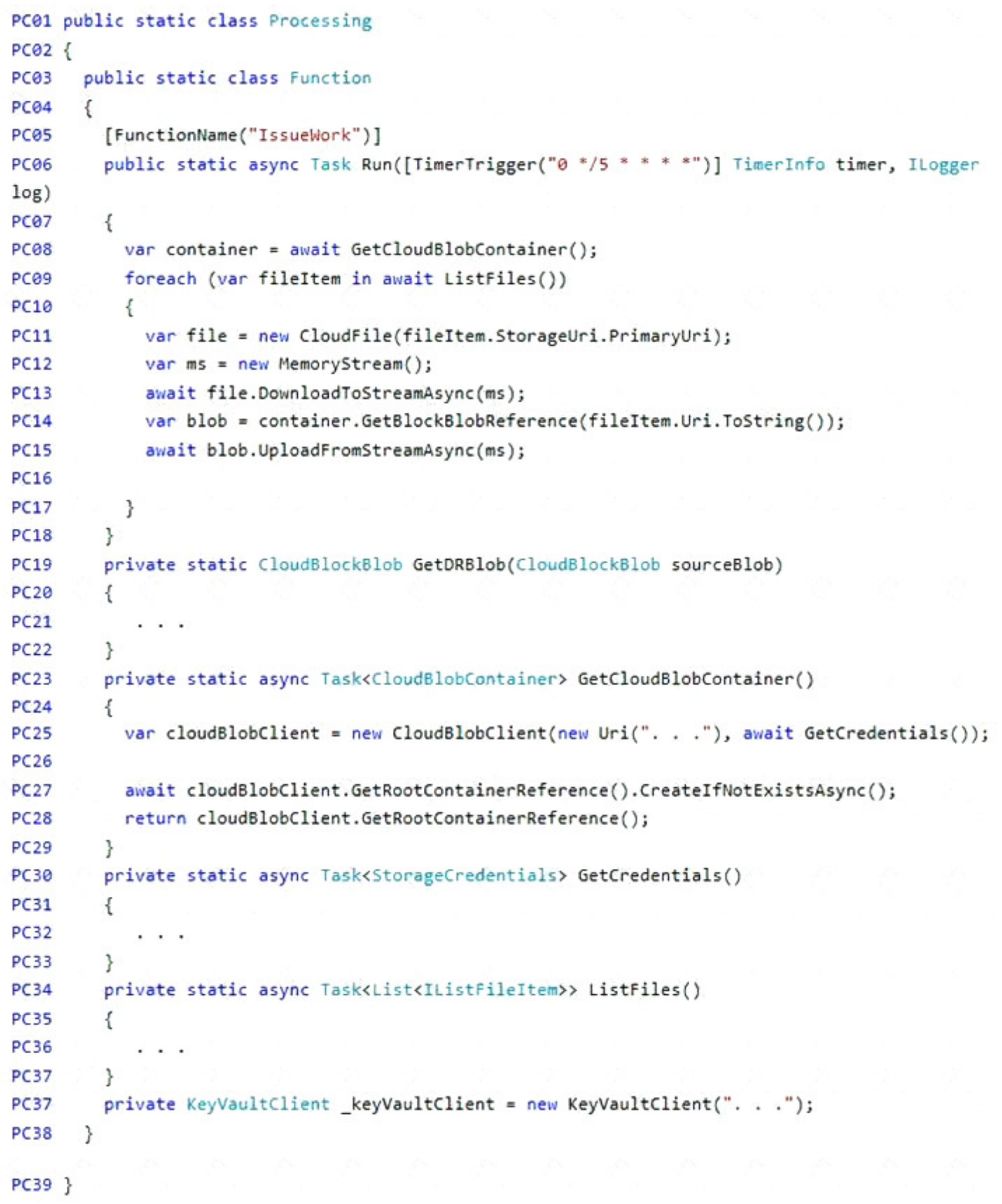
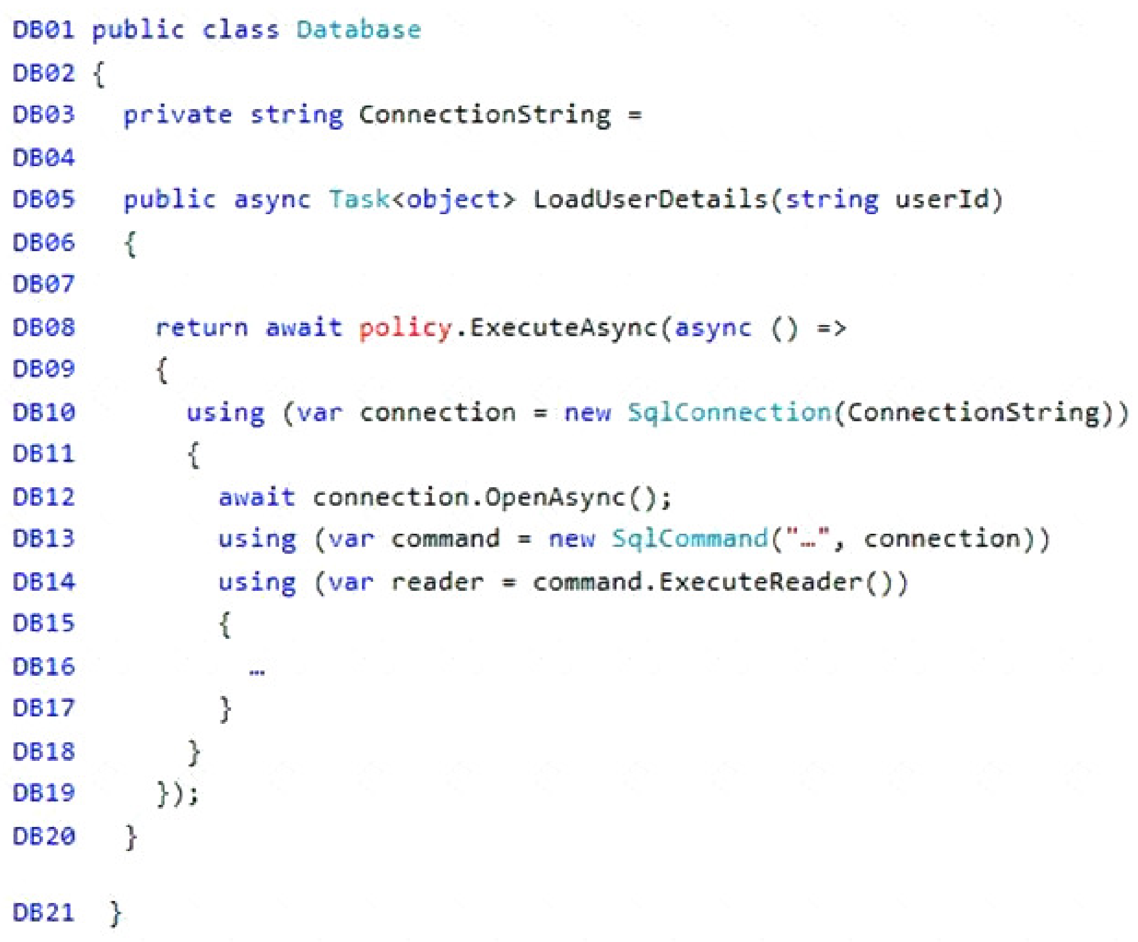


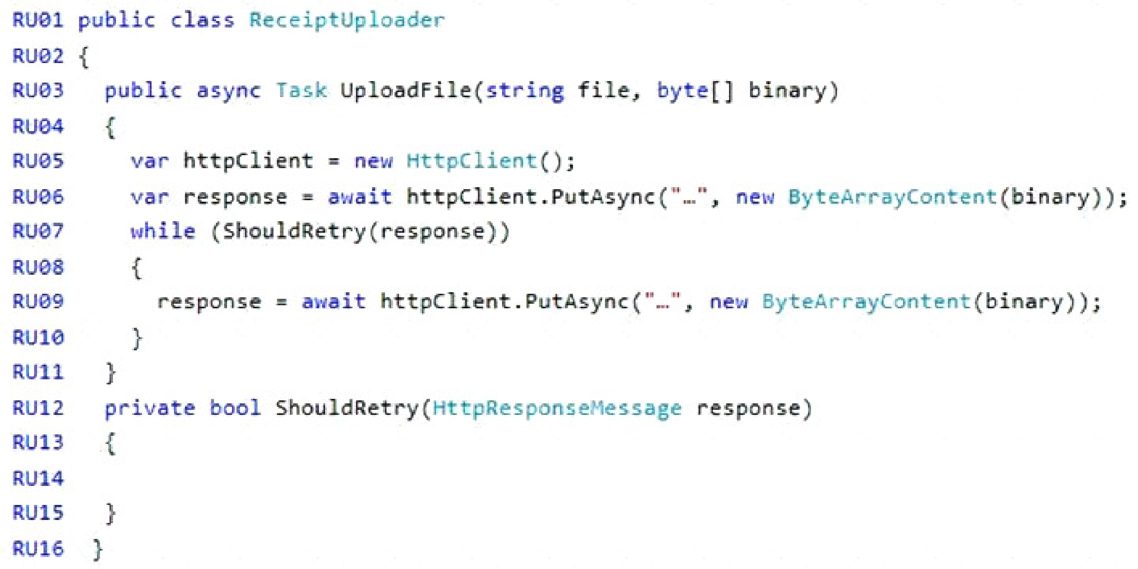
```
PC01 public static class Processing
PC02 {
PC03   public static class Function
PC04   {
PC05     [FunctionName("IssueWork")]
PC06     public static async Task Run([TimerTrigger("0 */5 * * * *")] TimerInfo timer, ILogger
log)
PC07     {
PC08       var container = await GetCloudBlobContainer();
PC09       foreach (var fileItem in await ListFiles())
PC10       {
PC11         var file = new CloudFile(fileItem.StorageUri.PrimaryUri);
PC12         var ms = new MemoryStream();
PC13         await file.DownloadToStreamAsync(ms);
PC14         var blob = container.GetBlockBlobReference(fileItem.Uri.ToString());
PC15         await blob.UploadFromStreamAsync(ms);
PC16
PC17       }
PC18     }
PC19     private static CloudBlockBlob GetDRBlob(CloudBlockBlob sourceBlob)
PC20     {
PC21        . . .
PC22     }
PC23     private static async Task<CloudBlobContainer> GetCloudBlobContainer()
PC24     {
PC25       var cloudBlobClient = new CloudBlobClient(new Uri(". . ."), await GetCredentials());
PC26
PC27       await cloudBlobClient.GetRootContainerReference().CreateIfNotExistsAsync();
PC28       return cloudBlobClient.GetRootContainerReference();
PC29     }
PC30     private static async Task<StorageCredentials> GetCredentials()
PC31     {
PC32        . . .
PC33     }
PC34     private static async Task<List<IListFileItem>> ListFiles()
PC35     {
PC36        . . .
PC37     }
PC37     private KeyVaultClient _keyVaultClient = new KeyVaultClient(". . .");
PC38   }

PC39 }
```

Database.cs



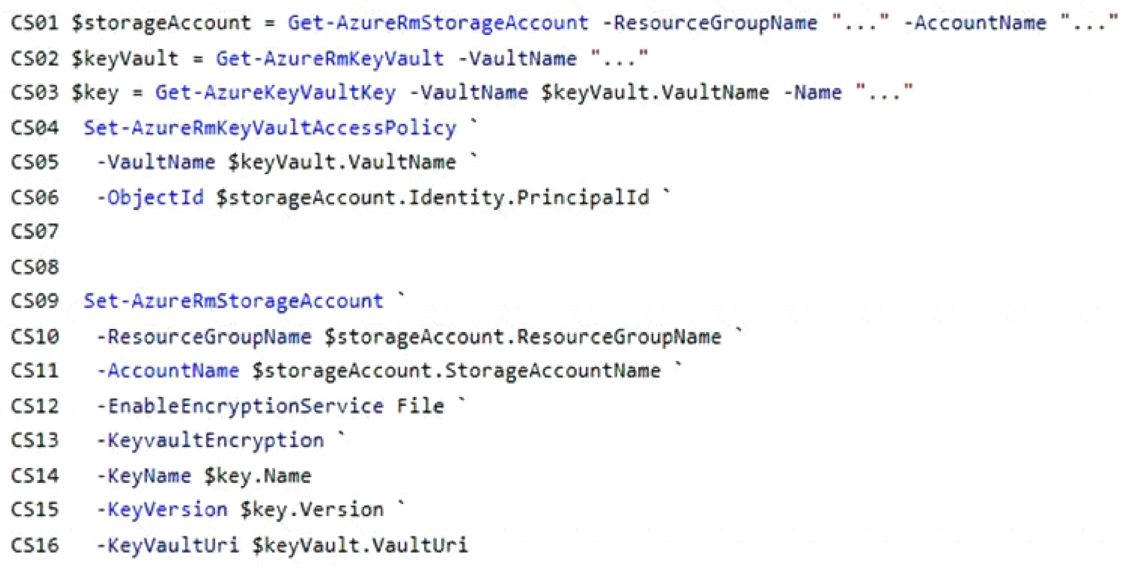
```
DB01 public class Database
DB02 {
DB03   private string ConnectionString =
DB04
DB05   public async Task<object> LoadUserDetails(string userId)
DB06   {
DB07
DB08     return await policy.ExecuteAsync(async () =>
DB09     {
DB10       using (var connection = new SqlConnection(ConnectionString))
DB11       {
DB12         await connection.OpenAsync();
DB13         using (var command = new SqlCommand("…", connection))
DB14         using (var reader = command.ExecuteReader())
DB15         {
DB16           …
DB17         }
DB18       }
DB19     });
DB20   }

DB21 }
```

ReceiptUploader.cs



```
RU01 public class ReceiptUploader
RU02 {
RU03   public async Task UploadFile(string file, byte[] binary)
RU04   {
RU05     var httpClient = new HttpClient();
RU06     var response = await httpClient.PutAsync("…", new ByteArrayContent(binary));
RU07     while (ShouldRetry(response))
RU08     {
RU09       response = await httpClient.PutAsync("…", new ByteArrayContent(binary));
RU10     }
RU11   }
RU12   private bool ShouldRetry(HttpResponseMessage response)
RU13   {
RU14
RU15   }
RU16 }
```

ConfigureSSE.ps1



```
CS01 $storageAccount = Get-AzureRmStorageAccount -ResourceGroupName "..." -AccountName "..."
CS02 $keyVault = Get-AzureRmKeyVault -VaultName "..."
CS03 $key = Get-AzureKeyVaultKey -VaultName $keyVault.VaultName -Name "..."
CS04  Set-AzureRmKeyVaultAccessPolicy `
CS05   -VaultName $keyVault.VaultName `
CS06   -ObjectId $storageAccount.Identity.PrincipalId `
CS07
CS08
CS09  Set-AzureRmStorageAccount `
CS10   -ResourceGroupName $storageAccount.ResourceGroupName `
CS11   -AccountName $storageAccount.StorageAccountName `
CS12   -EnableEncryptionService File `
CS13   -KeyvaultEncryption `
CS14   -KeyName $key.Name
CS15   -KeyVersion $key.Version `
CS16   -KeyVaultUri $keyVault.VaultUri
```

QUESTION 1
You need to resolve the log capacity issue. What should you do?

A. Create an Application Insights Telemetry Filter
B. Change the minimum log level in the host.json file for the function
C. Implement Application Insights Sampling
D. Set a LogCategoryFilter during startup

Correct Answer: C
Explanation:
Scenario, the log capacity issue: Developers report that the number of log message in the trace output for the processor is too high, resulting in lost log messages.

Sampling is a feature in Azure Application Insights. It is the recommended way to reduce telemetry traffic and storage, while preserving a statistically correct analysis of application data. The filter selects items that are related, so that you can navigate between items when you are doing diagnostic investigations. When metric counts are presented to you in the portal, they are renormalized to take account of the sampling, to minimize any effect on the statistics.

Sampling reduces traffic and data costs, and helps you avoid throttling.

Reference:
https://docs.microsoft.com/en-us/azure/azure-monitor/app/sampling

QUESTION 2
You need to ensure the security policies are met. What code do you add at line CS07 of ConfigureSSE.ps1?

A. -PermissionsToKeys create, encrypt, decrypt
B. -PermissionsToCertificates create, encrypt, decrypt
C. -PermissionsToCertificates wrapkey, unwrapkey, get
D. -PermissionsToKeys wrapkey, unwrapkey, get

Correct Answer: B
Explanation:
Scenario: All certificates and secrets used to secure data must be stored in Azure Key Vault.

You must adhere to the principle of least privilege and provide privileges which are essential to perform the intended function.

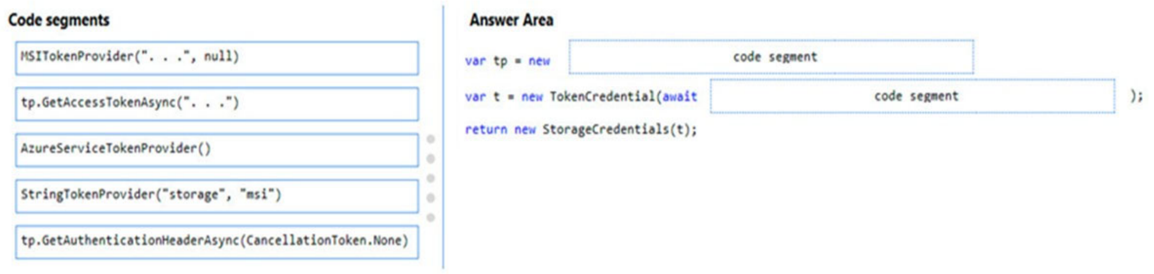
The Set-AzureRmKeyValutAccessPolicy parameter -PermissionsToKeys specifies an array of key operation permissions to grant to a user or service principal. The acceptable values for this parameter: decrypt, encrypt, unwrapKey, wrapKey, verify, sign, get, list, update, create, import, delete, backup, restore, recover, purge

Reference:
https://docs.microsoft.com/en-us/powershell/module/azurerm.keyvault/set-azurermkeyvaultaccesspolicy

QUESTION 3
DRAG DROP
You need to add code at line PC32 in Processing.cs to implement the GetCredentials method in the Processing class.

How should you complete the code? To answer, drag the appropriate code segments to the correct locations. Each code segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.



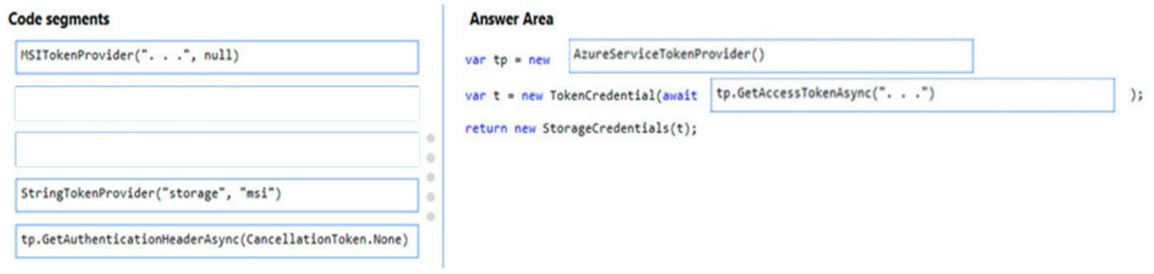
Code segments

```
MSITokenProvider(". . .", null)
tp.GetAccessTokenAsync(". . .")
AzureServiceTokenProvider()
StringTokenProvider("storage", "msi")
tp.GetAuthenticationHeaderAsync(CancellationToken.None)
```

Answer Area

```
var tp = new [code segment]
var t = new TokenCredential(await [code segment] );
return new StorageCredentials(t);
```

Correct Answer:



Code segments

```
MSITokenProvider(". . .", null)


StringTokenProvider("storage", "msi")
tp.GetAuthenticationHeaderAsync(CancellationToken.None)
```

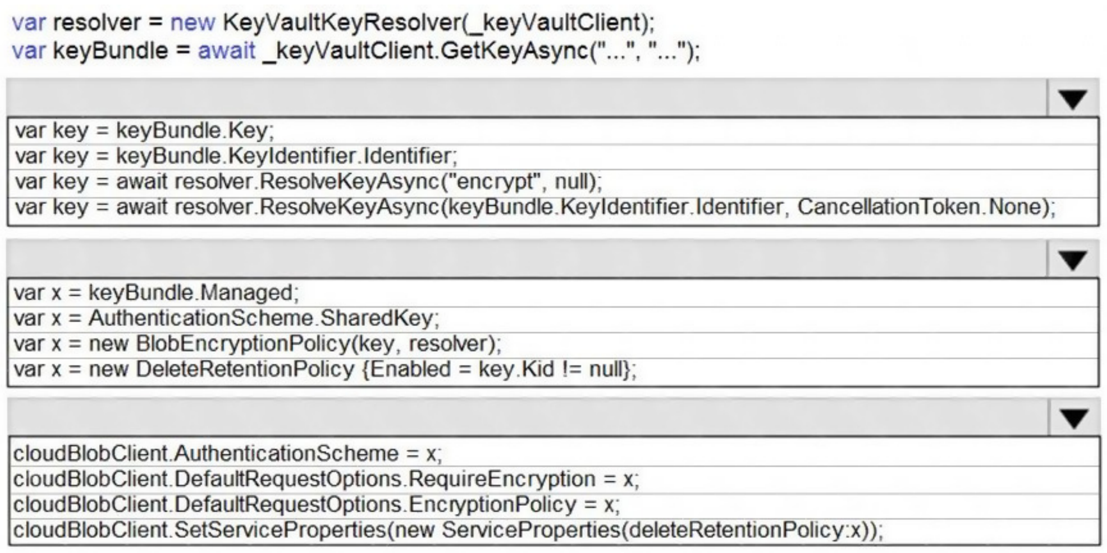
Answer Area

```
var tp = new AzureServiceTokenProvider()
var t = new TokenCredential(await tp.GetAccessTokenAsync(". . .") );
return new StorageCredentials(t);
```

QUESTION 4
HOTSPOT
You need to add code at line PC26 of Processing.cs to ensure that security policies are met.

How should you complete the code that you will add at line PC26? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



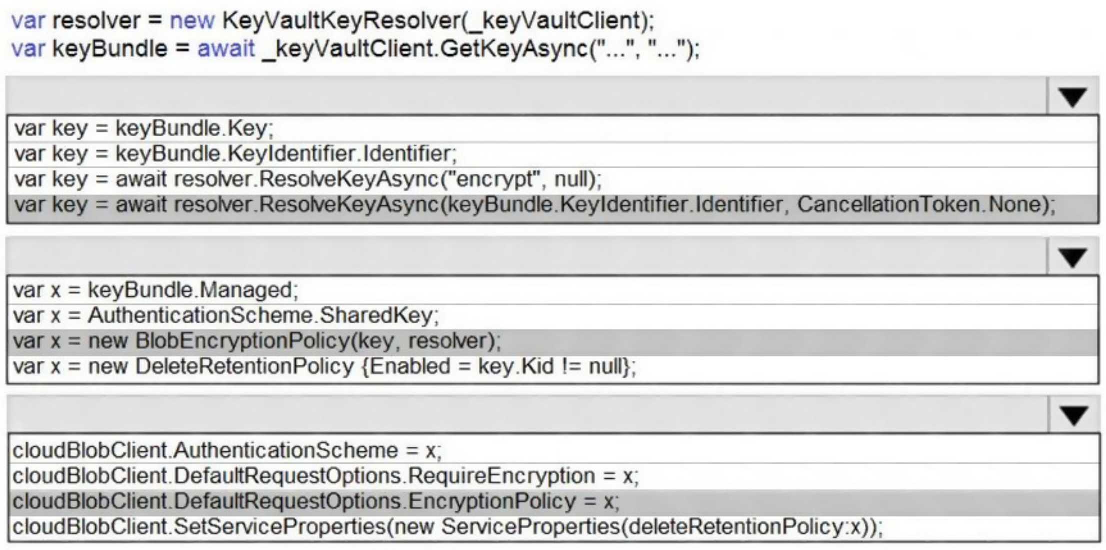
```
var resolver = new KeyVaultKeyResolver(_keyVaultClient);
var keyBundle = await _keyVaultClient.GetKeyAsync("...", "...");
```

- var key = keyBundle.Key;
- var key = keyBundle.KeyIdentifier.Identifier;
- var key = await resolver.ResolveKeyAsync("encrypt", null);
- var key = await resolver.ResolveKeyAsync(keyBundle.KeyIdentifier.Identifier, CancellationToken.None);

- var x = keyBundle.Managed;
- var x = AuthenticationScheme.SharedKey;
- var x = new BlobEncryptionPolicy(key, resolver);
- var x = new DeleteRetentionPolicy {Enabled = key.Kid != null};

- cloudBlobClient.AuthenticationScheme = x;
- cloudBlobClient.DefaultRequestOptions.RequireEncryption = x;
- cloudBlobClient.DefaultRequestOptions.EncryptionPolicy = x;
- cloudBlobClient.SetServiceProperties(new ServiceProperties(deleteRetentionPolicy:x));

Correct Answer:



```
var resolver = new KeyVaultKeyResolver(_keyVaultClient);
var keyBundle = await _keyVaultClient.GetKeyAsync("...", "...");
```

- var key = keyBundle.Key;
- var key = keyBundle.KeyIdentifier.Identifier;
- var key = await resolver.ResolveKeyAsync("encrypt", null);
- [✓ var key = await resolver.ResolveKeyAsync(keyBundle.KeyIdentifier.Identifier, CancellationToken.None);]

- var x = keyBundle.Managed;
- var x = AuthenticationScheme.SharedKey;
- [✓ var x = new BlobEncryptionPolicy(key, resolver);]
- var x = new DeleteRetentionPolicy {Enabled = key.Kid != null};

- cloudBlobClient.AuthenticationScheme = x;
- cloudBlobClient.DefaultRequestOptions.RequireEncryption = x;
- [✓ cloudBlobClient.DefaultRequestOptions.EncryptionPolicy = x;]
- cloudBlobClient.SetServiceProperties(new ServiceProperties(deleteRetentionPolicy:x));