Azure Confidential VM: Platform Guest Attestation

Overview

Confidential VM, an Azure Confidential Computing offering, is an IaaS Virtual Machine for tenants with stringent security and confidentiality requirements. A basic capability in a confidential computing environment is to have assurance that the platform in which the Confidential VM is running can be trusted. If the platform cannot be trusted, then the workload and associated data, if deployed, will be at a greater risk for compromise.

With Platform guest attestation, a relying party can assure and have increased confidence that their software inside a Confidential VM runs on expected confidential hardware platform (e.g: AMD-SEVSNP).

Scenarios

The common scenarios for Platform guest attestation are as follows:

- Ensure a Confidential VM is running on an expected confidential hardware platform (currently AMD SEV-SNP only)
- Ensure a Confidential VM (or Trusted Launch VM) has secure boot enabled that protects lowers layers (firmware, bootloader, kernel) of the VM from malware (rootkit, bootkit).
- Ensure that a relying party is presented evidence that a Confidential VM is running on a confidential hardware platform

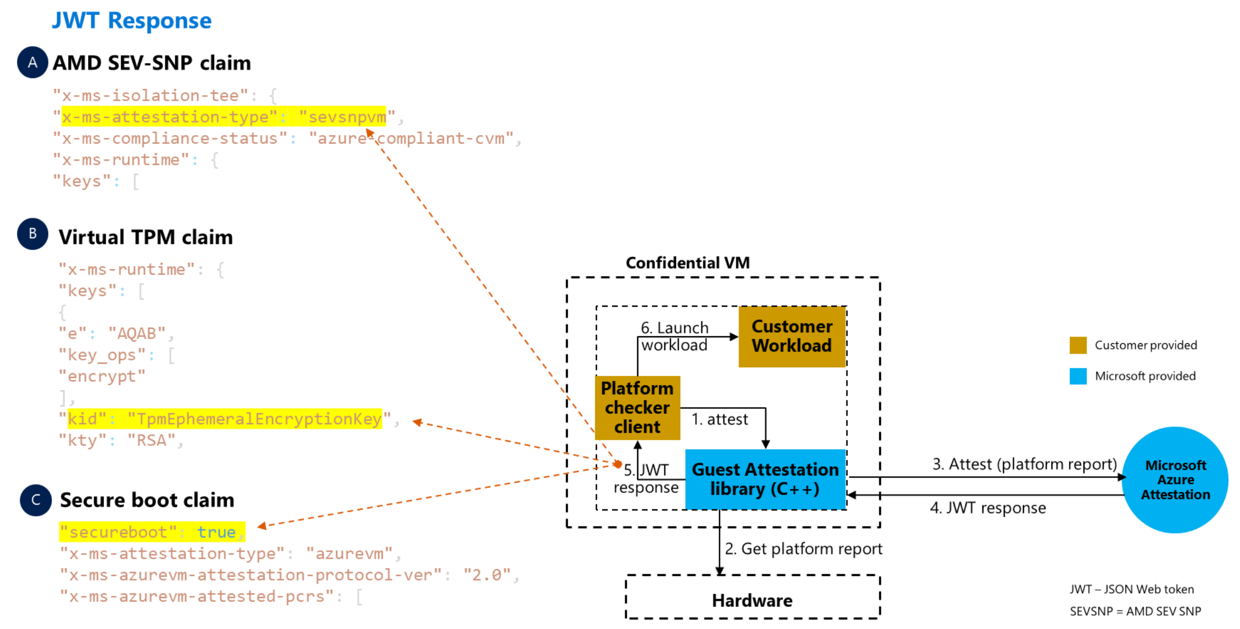
Workflow

To realize the scenarios above, a few components and services are involved; namely: client program, platform guest attestation client library, hardware (for report), and Microsoft Azure Attestation service. The mechanics of the request flow is shown in the diagrams below.

Typical operational workflows to incorporate the client library and make attestation requests are described below.

Platform Attestation: request in separate client program

In this workflow, attestation requests are performed in a separate client program to determine if the Confidential VM is running on desired hardware platform before a workload is launched.



```
"x-ms-isolation-tee": {
"x-ms-attestation-type": "sevsnpvm",
"x-ms-compliance-status": "azure-compliant-cvm",
"x-ms-runtime": {
"keys": [
```

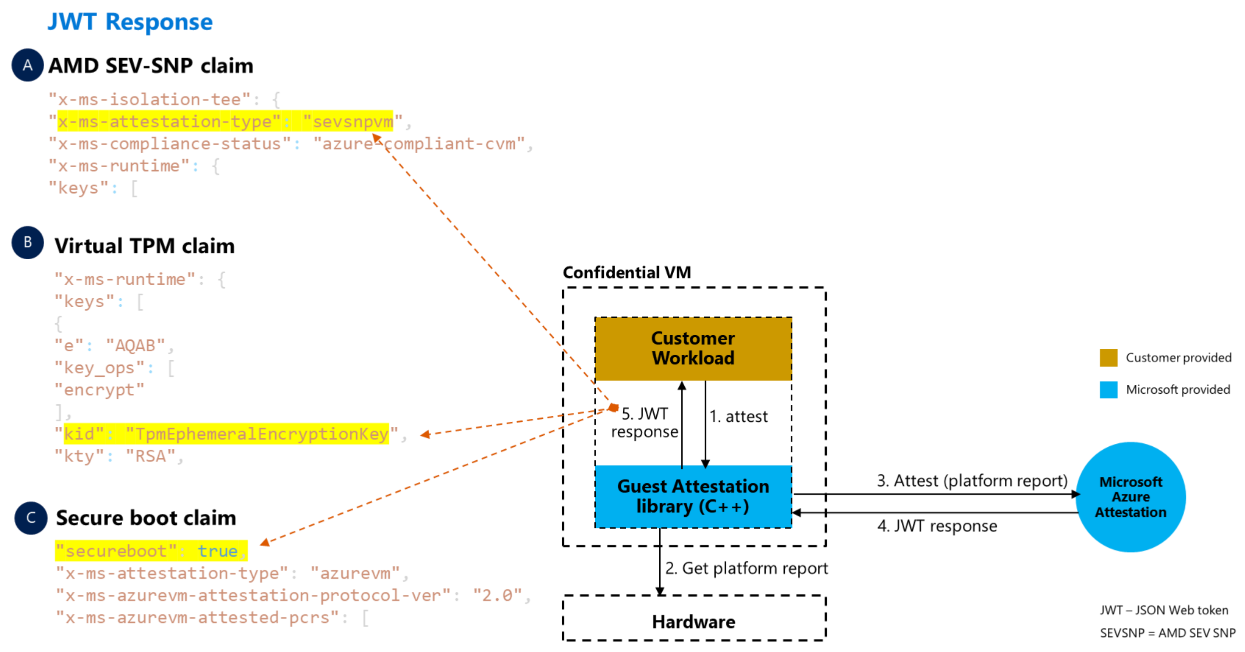
```
"x-ms-runtime": {
"keys": [
{
"e": "AQAB",
"key_ops": [
"encrypt"
],
"kid": "TpmEphemeralEncryptionKey",
"kty": "RSA",
```

```
"secureboot": true,
"x-ms-attestation-type": "azurevm",
"x-ms-azurevm-attestation-protocol-ver": "2.0",
"x-ms-azurevm-attested-pcrs": [
```

To perform attestation, a client program (called Platform checker client in diagram) must integrate with the attestation library and run inside a confidential VM. Upon making a request to the attestation library, the client program can parse the response to determine if the VM is running on a desired hardware platform, and/or secure boot setting to launch the sensitive workload.

Platform Attestation: request from inside a workload

In this workflow, attestation requests are performed inside the workload (at the start of this program) to determine if the Confidential VM is running on desired hardware platform before a workload is launched.



```
"x-ms-isolation-tee": {
"x-ms-attestation-type": "sevsnpvm",
"x-ms-compliance-status": "azure-compliant-cvm",
"x-ms-runtime": {
"keys": [
```

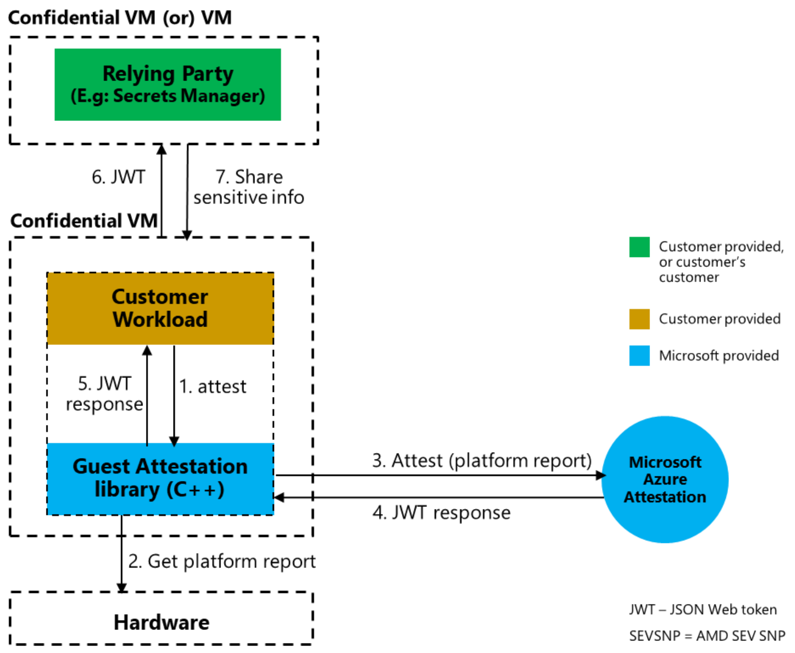
```
"x-ms-runtime": {
"keys": [
{
"e": "AQAB",
"key_ops": [
"encrypt"
],
"kid": "TpmEphemeralEncryptionKey",
"kty": "RSA",
```

```
"secureboot": true,
"x-ms-attestation-type": "azurevm",
"x-ms-azurevm-attestation-protocol-ver": "2.0",
"x-ms-azurevm-attested-pcrs": [
```

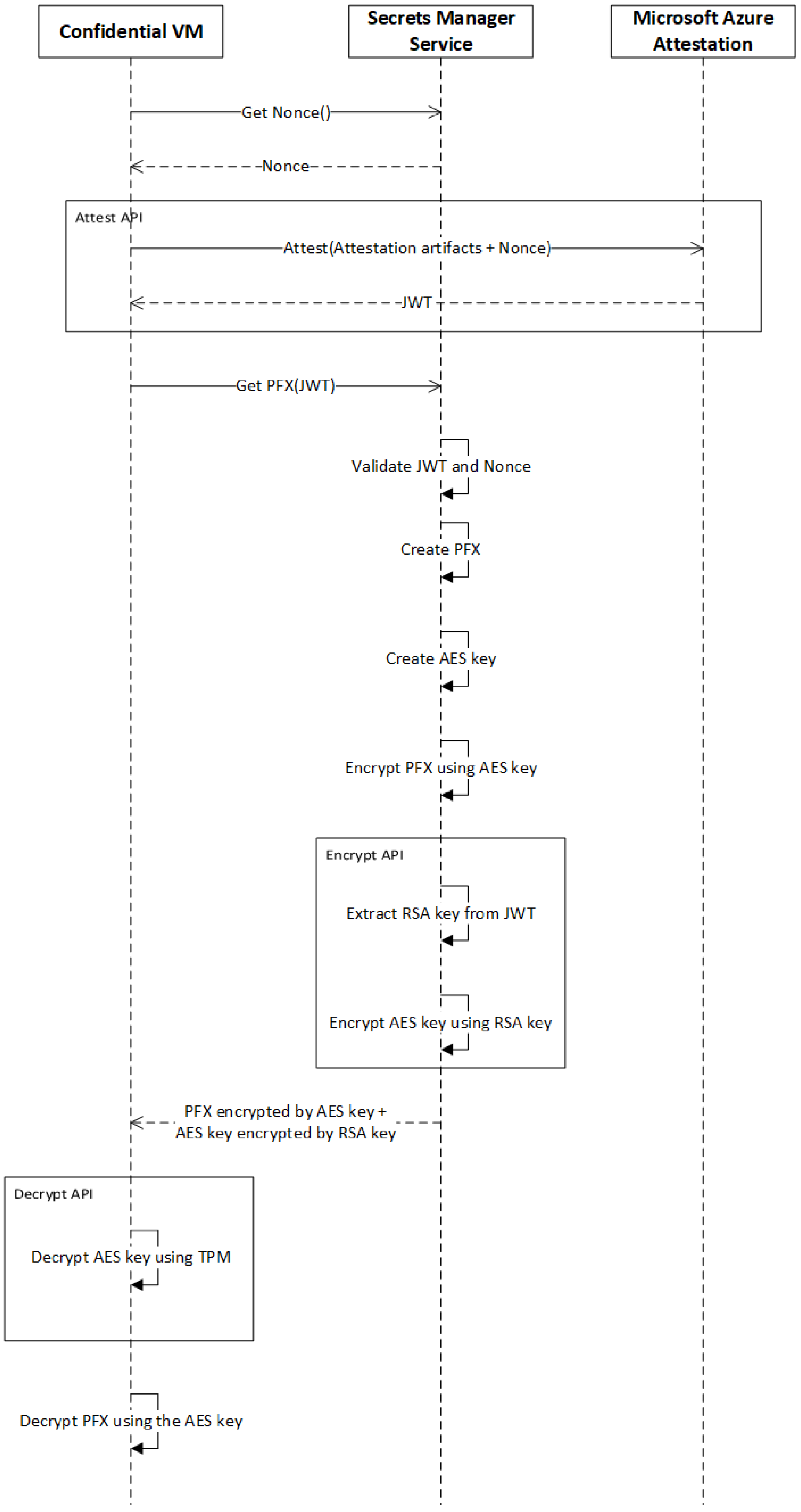
To perform attestation, the customer workload must integrate with the attestation library and run inside a confidential VM. Upon making a request to the attestation library, the customer workload can parse the response to determine if the VM is running on a desired hardware platform, and/or secure boot setting to continue to fully setup the sensitive workload.

Platform attestation: Relying party handshake

In addition to platform attestation, in one of the scenarios, the Confidential VM (CVM) must prove its running on a confidential platform before a relying party will engage. The Confidential VM must present the attestation token to the relying party to start the engagement. Examples of engagement are the CVM want secrets from a Secrets management service, or a client wants to ensure the CVM is running on confidential platform before it divulges PII data for CVM to process. The handshake between a Confidential VM and relying party is depicted in the diagram below.



To understand the relying party scenario, the sequence diagram shows the request/response between the involved systems using guest attestation library APIs. In this example, the Confidential VM interacts with the Secrets manager to bootstrap itself using the received secrets.


Confidential VM
Secrets Manager Service
Microsoft Azure Attestation
Get Nonce()
Nonce
Attest API
Attest(Attestation artifacts + Nonce)
JWT
Get PFX(JWT)
Validate JWT and Nonce
Create PFX
Create AES key
Encrypt PFX using AES key
Encrypt API
Extract RSA key from JWT
Encrypt AES key using RSA key
PFX encrypted by AES key +
AES key encrypted by RSA key
Decrypt API
Decrypt AES key using TPM
Decrypt PFX using the AES key

API description

Platform guest attestation library provides APIs to perform attestation, encrypt and decrypt data. APIs listed here can be used to accomplish scenarios and workflows described above. Note that the APIs can only be invoked after the CVM is in a running state.

Definition: Attest The Attest API takes ‘Client Parameters’ object as input, initiates an attestation request to Microsoft Azure Attestation Service (MAA), and returns a decrypted attestation token.	

`AttestationResult Attest (const attest::ClientParameters& client_params,`
`unsigned char** jwt_token)`	

Parameters	

`[in]` **client_params**	ClientParameters object containing the parameters from the client needed for attestation.
	Zero or more key value pairs for any client/customer metadata to be returned in response payload. Key value pairs must be in JSON string format:
	"{\"key1\":\"value1\",\"key2\":\"value2\"}"
	
	`E.g: attestation freshness`
	`{\”Nonce\”:\”011510062022\”}`

`[out]` **jwt_token**	The decrypted jwt token that will be returned by MAA as a response to the attestation request. The memory is allocated by the method and the caller is expected to free this memory by calling Attest::Free() method
Returns	
AttestationResult	In case of success, AttestationResult object with error code ErrorCode::Success will be returned. In case of failure, an appropriate ErrorCode will be set in the AttestationResult object and error description will be provided. Return one of the following error codes: • -1: Error initializing failure • -2: Error parsing response • -3: MSI token not found • -4: Request exceeded retries • -5: Request failed • -6: Attestation failed • -7: Send request failed • -8: Invalid input parameter • -9: Attestation parameters validation failed

	• -10: Memory allocation failed • -11: Failed to get os info • -12: TPM internal failure • -13: TPM operation failed • -14: JWT decryption failed • -15: JWT decryption TPM error • -16: Invalid JSON response • -17: Empty VCEK certificate • -18: Empty response • -19: Empty request body • -20: Report parsing failure • -21: Report empty • -22: Error extracting JWK info • -23: Error converting JWK to RSA public key • -24: EVP pkey encryption init failed • -25: EVP pkey encrypt failed • -26: Data decryption TPM error • -27: Error parsing DNS info

Definition: Encrypt
The Encrypt API takes data to be encrypted and JWT as input and encrypts the data using public ephemeral key present in the JWT.

```
AttestationResult Encrypt (const attest::EncryptionType encryption_type,
                           const unsigned char * jwt_token,
                           const unsigned char * data,
                           uint32_t data_size,
                           unsigned char ** encrypted_data,
                           uint32_t * encrypted_data_size,
                           unsigned char ** encryption_metadata,
                           uint32_t * encryption_metadata_size)
```

Parameters

`[in]` **encryption_type**	the type of encryption currently the only encryption type supported is 'NONE', which expects the caller to pass symmetric key as the data to be encrypted. The RSA Public key present in the JWT is used to perform the encryption.
`[in]` **jwt_token**	the attestation JWT (null terminated string)
`[in]` **data**	the data to be encrypted
`[in]` **data_size**	the size of the data to be encrypted
`[out]` **encrypted_data**	the encrypted data (the memory is allocated by the method and the caller is expected to free this memory by calling Attest::Free() method)
`[out]` **encrypted_data_size**	the size of the encrypted data

`[out]` **encryption_metadata**	the encryption metadata in form of base64 encoded JSON (the memory is allocated by the method and the caller is expected to free this memory by calling Attest::Free() method)
`[out]` **encryption_metadata_size**	the size of the encryption metadata
Returns	
AttestationResult	Return one of the following error codes: • -1: Error initializing failure • -2: Error parsing response • -3: MSI token not found • -4: Request exceeded retries • -5: Request failed • -6: Attestation failed • -7: Send request failed • -8: Invalid input parameter • -9: Attestation parameters validation failed • -10: Memory allocation failed • -11: Failed to get os info • -12: TPM internal failure • -13: TPM operation failed • -14: JWT decryption failed • -15: JWT decryption TPM error • -16: Invalid JSON response • -17: Empty VCEK certificate • -18: Empty response • -19: Empty request body • -20: Report parsing failure • -21: Report empty • -22: Error extracting JWK info • -23: Error converting JWK to RSA public key • -24: EVP pkey encryption init failed • -25: EVP pkey encrypt failed • -26: Data decryption TPM error • -27: Error parsing DNS info

Definition: Decrypt

The Decrypt API takes encrypted data as input and decrypts the data using private ephemeral key sealed to the TPM.

```
AttestationResult Decrypt(const attest::EncryptionType encryption_type,
                          const unsigned char * encrypted_data,
                          uint32_t encrypted_data_size,
const unsigned char * encryption_metadata,
uint32_t encryption_metadata_size,
unsigned char ** decrypted_data,
uint32_t * decrypted_data_size)
```

Parameters

`[in]` **encryption_type**	the type of encryption currently the only encryption type supported is 'NONE', which expects the caller to pass the encrypted symmetric key as input. The RSA Private key present in the TPM is used to perform the decryption.
`[in]` **encrypted_data**	The encrypted data
`[in]` **encrypted_data_size**	The size of encrypted data
`[in]` **encryption_metadata**	The encryption metadata
`[in]` **encryption_metadata_size**	The size of encryption metadata
`[out]` **decrypted_data**	The decrypted data (the memory is allocated by the method and the caller is expected to free this memory by calling Attest::Free() method)
`[out]` **decrypted_data_size**	The size of decrypted data

Returns

AttestationResult	Return one of the following error codes: • -1: Error initializing failure • -2: Error parsing response • -3: MSI token not found • -4: Request exceeded retries • -5: Request failed • -6: Attestation failed • -7: Send request failed • -8: Invalid input parameter • -9: Attestation parameters validation failed • -10: Memory allocation failed • -11: Failed to get os info • -12: TPM internal failure • -13: TPM operation failed • -14: JWT decryption failed • -15: JWT decryption TPM error • -16: Invalid JSON response • -17: Empty VCEK certificate • -18: Empty response • -19: Empty request body • -20: Report parsing failure

	• -21: Report empty • -22: Error extracting JWK info • -23: Error converting JWK to RSA public key • -24: EVP pkey encryption init failed • -25: EVP pkey encrypt failed • -26: Data decryption TPM error • -27: Error parsing DNS info

Definition: Free This API deallocates the memory previously allocated by the library	
`Free(void* ptr);`	
Parameters	
`[in]` **ptr**	Pointer to memory block previously allocated
Returns	
-	-

JWT Information

The full structure of the JWT is available here. Different parts of the JWT can be extracted to fulfill the scenarios described above. Key fields for platform guest attestation are listed below.

Claim	Attribute	Value (sample)
-	**x-ms-azurevm-vmid**	2DEDC52A-6832-46CE-9910-E8C9980BF5A7
AMD SEV-SNP hardware	**x-ms-isolation-tee**	-
	x-ms-isolation-tee	sevsnpvm
	x-ms-compliance-status	azure-compliant-cvm
Secure boot	**x-ms-runtime -> vm-configuration**	-
	secure-boot	true
Virtual TPM	tpm-enabled	true
	x-ms-runtime->keys	-
	kid	TpmEphemeralEncryptionKey

Compilation instructions

Linux

Create a Linux Confidential or Trusted Launch virtual machine in Azure and clone the application.

Use the command below to install the build-essential package. This package will install everything required for compiling our sample application written in C++.

```
$ sudo apt-get install build-essential
```

Use the below commands to install libcurl4-openssl-dev and libjsoncpp-dev packages

```
$ sudo apt-get install libcurl4-openssl-dev

$ sudo apt-get install libjsoncpp-dev
```

Download the attestation package from the following location - https://packages.microsoft.com/repos/azurecore/pool/main/a/azguestattestation1/

Use the below command to install the attestation package

```
$ sudo dpkg -i azguestattestation1_<latest-version>_amd64.deb
```

Windows

Create a Windows Confidential or Trusted Launch virtual machine in Azure and clone the sample application.

Install Visual Studio with the Desktop development with C++ workload installed and running on your computer. If it's not installed yet, follow the steps in Install C++ support in Visual Studio.

To build your project, choose **Build Solution** from the **Build** menu. The **Output** window shows the results of the build process.

Once the build is successful, to run the application navigate to the `Release` build folder and run the `AttestationClientApp.exe` file.

Sample Code

Sample code on how to use the attest API and make attestation requests for a Confidential VM is available in GitHub [Linux, Windows]. Depending on your operational workflow (see Workflows section), the sample code can be reused in your client program or workload code, or you can use it as is.