Crypto Adapter

With crypto components, it is possible to generate private/public key pairs to create and validate signatures with the RSA encryption algorithm.

Example File (Builder project Advanced Modeling/Security):

<your example path>\Advanced Modeling\Security\uml\crypt.xml

All these can be done in an action node with the stereotype <<Crypto>> and its specific actions.You can

- create a key pair
- create a signature
- validate a signature against a key
- create a hash value of a given data blob

As this is a static adapter which cannot be configured in any way, the names of the input and output object flow states must be exactly named like described below.

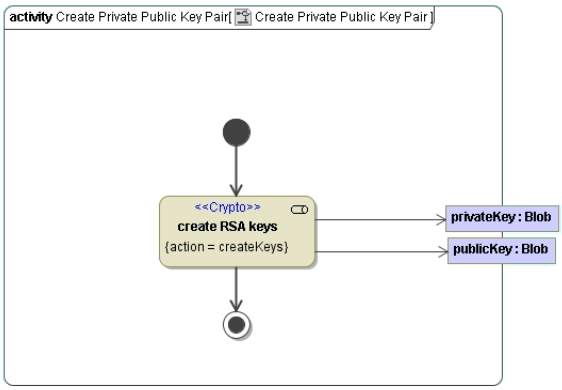
On this Page:

Creating a Key Pair

The first step is to create a private/public key pair.

- You can use a key pair in format **PKCS#8** (generated e.g. by OpenSSL). In this case, you do not need to generate a key pair with the Bridge and you can proceed with the next steps.
- You can use the Bridge to generate a private/public key pair in format **PKCS#1**. This can be done via the action **createKeys**. The adapter does not need any input but creates two **Blob** objects named **privateKey** and **publicKey**.

The following figure shows the necessary activity diagram to create the private and public keys in format **PKCS#1**.



With this pair of keys it is possible to identify the sender of any data.

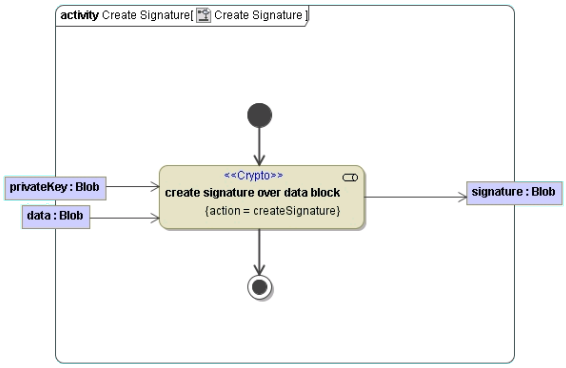
Creating a Signature

A signature can be generated by using the action **createSignature**. This action node expects three parameters of base type **Blob**:

- **privateKey**
- **data**
- **signature**

The signed data must be a **Blob**. So if you want to furnish a **String**, **Integer** or other base types with a signature, you have first to convert it to a **Blob** using the convert operation (see String Operations).
You do not have to specify the key format. The Runtime will try to read the private key as **PKCS#8**. If this fails, the Runtime will re-try to read the private key as PKCS#1.

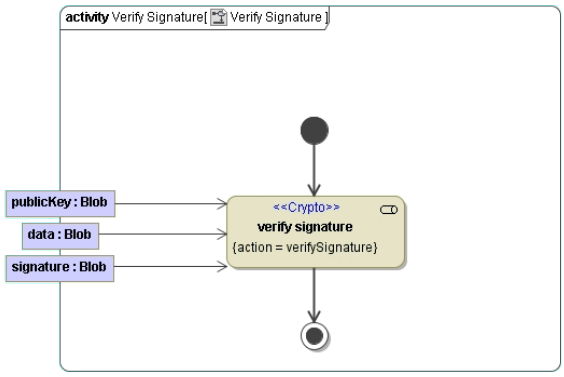
The following action node adds the **privateKey** to the data object and creates the object **signature** containing the data.



Validating a Signature

To check if the created signature is valid, you can use action **verifySignature**. This action takes a signature, the corresponding public key and data blob and returns an error message like "Verifying signature failed.", if one parameter is not valid or does not correspond to the others.
You do not have to specify the key format. The Runtime will try to read the public key as PKCS#8. If this fails, the Runtime will re-try to read the public key as PKCS#1.

The following figure shows how to verify a signature:

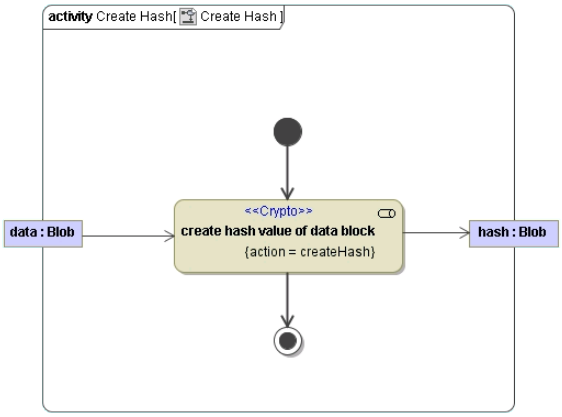


Hashing Data

Sometimes it is desirable to get a hash value of a given blob. This is done using an action node having the stereotype <<Crypto>> and the action **createHash**.

Builder 7.12.0 The **Crypto** adapter uses the hash algorithm specified with parameter **algorithm**, or the **digestAlgorithm** specified on the <<Crypto>> action node if no parameter has been supplied. If no algorithm is supplied at all, the adapter uses default algorithm **sha1**.
Refer to the Crypto Adapter Reference for more details.

The following figure shows how to create a hash with the default algorithm:


activity Create Hash[Create Hash]
<<Crypto>>
create hash value of data block
{action = createHash}
data : Blob
hash : Blob