Asymmetric Encryption

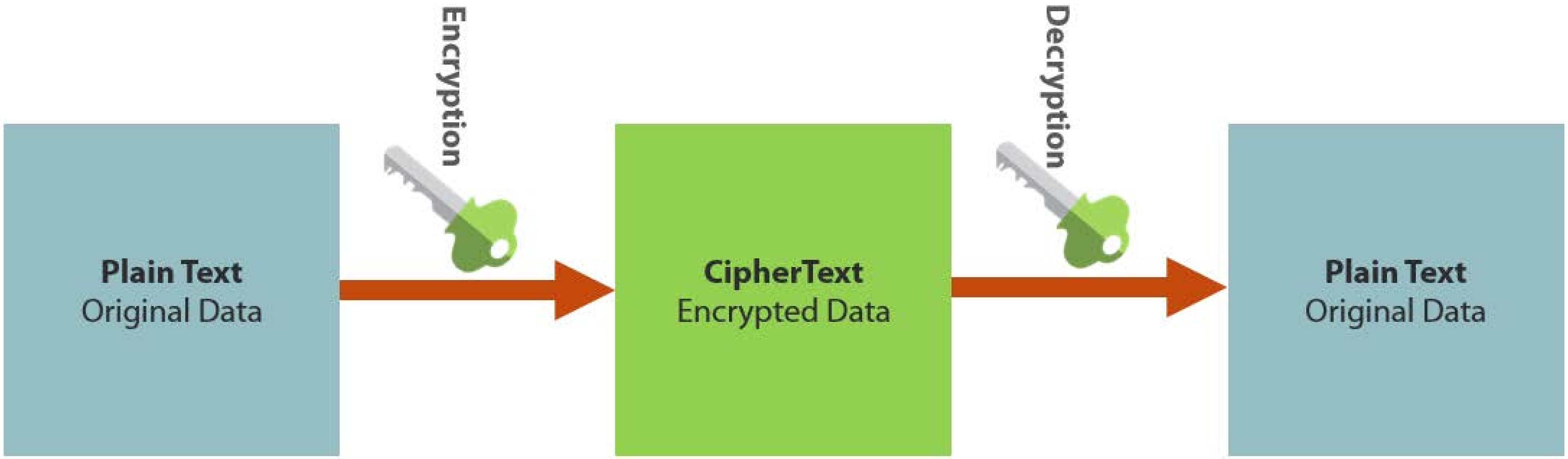
Stephen Haunts

@stephenhaunts | www.stephenhaunts.com

Overview

- Symmetric encryption recap
- What is asymmetric encryption?
- History of RSA
- How does RSA work?
- RSA in the .NET Framework

Symmetric Encryption Recap



Plain Text
Original Data

Encryption

CipherText
Encrypted Data

Decryption

Plain Text
Original Data

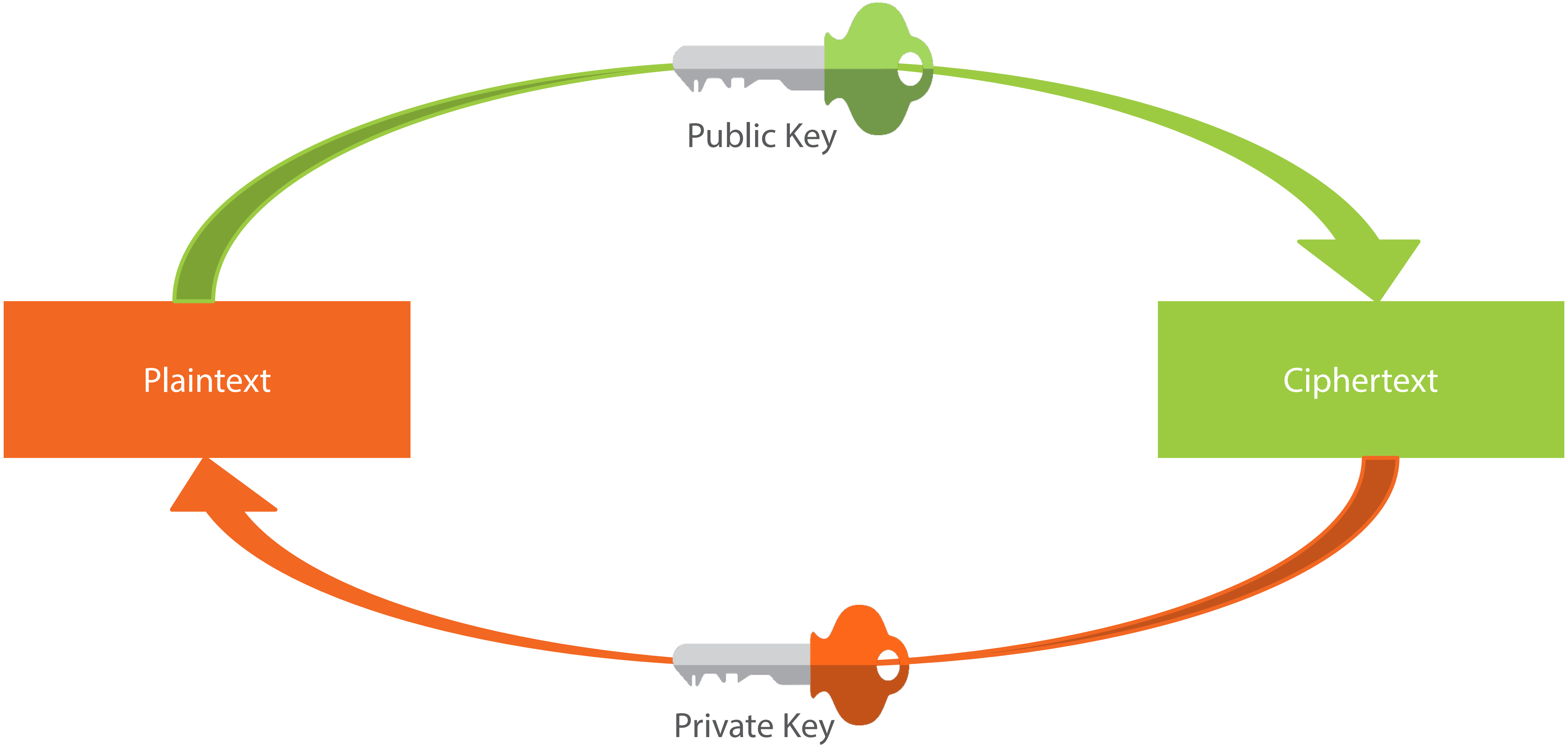
Symmetric Encryption Advantages

- Extremely secure
- Relatively fast

Symmetric Encryption Disadvantages

- Key sharing
- More damage if compromised

What Is Asymmetric Encryption



Public Key

Plaintext

Ciphertext

Private Key

What Is Asymmetric Encryption

- Sender and receiver don’t need to share keys prior
- Sender only needs the recipient’s public key

What Is Asymmetric Encryption

- Asymmetric encryption is slow compared to symmetric encryption

RSA History

- RSA was developed by RSA Security LLC
- RSA stands for Rivest, Shamir and Adelman, the inventors of the technique
- No efficient way to factor large numbers
- RSA is the de-facto standard for industrial strength encryption
- There are limits to the amount of data you can encrypt in one go
- RSA is commonly used to encrypt symmetric encryption keys

How Does RSA Work?

1024 bit keys

2048 bit keys

4096 bit keys

Key Derivation

- Public and private keys are based on prime numbers
- Factoring a number back into constituent prime numbers is hard

$$23 \times 17 = ?$$

Key Derivation

- Public and private keys are based on prime numbers
- Factoring a number back into constituent prime numbers is hard

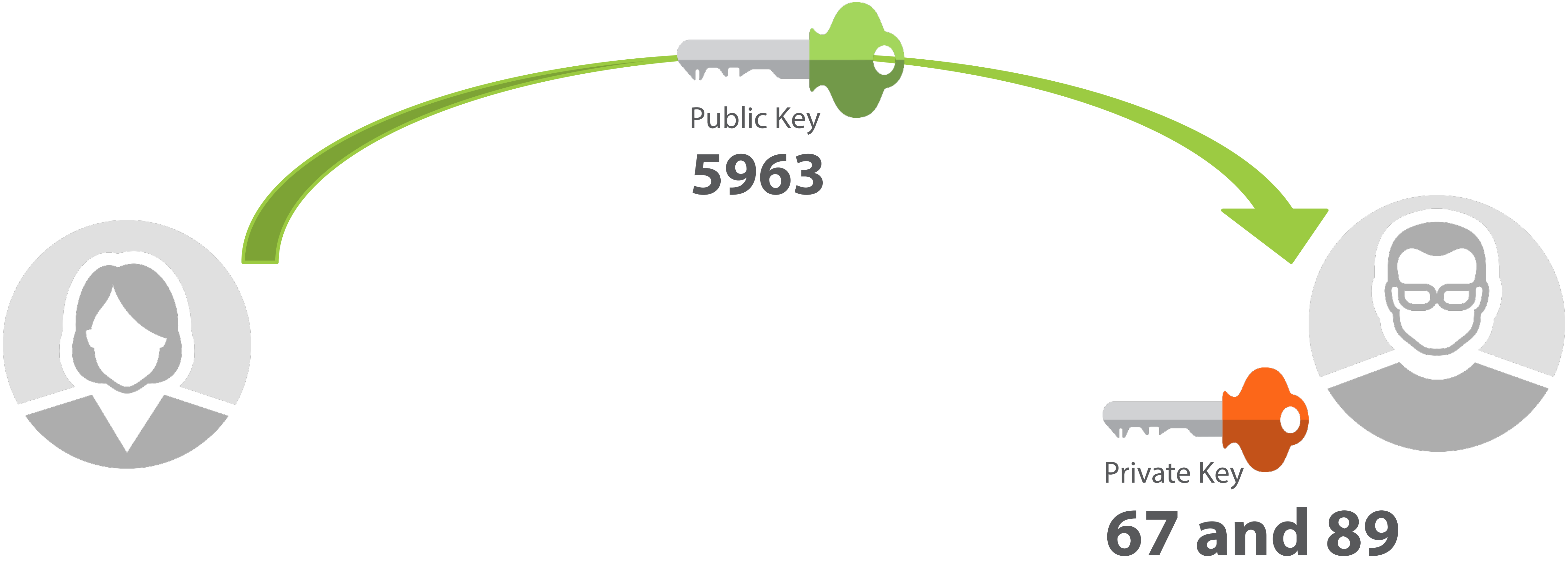
$$? \text{ x } ? = 5963$$

Key Derivation

- Public and private keys are based on prime numbers
- Factoring a number back into constituent prime numbers is hard

$$67 \times 89 = 5963$$

Key Derivation



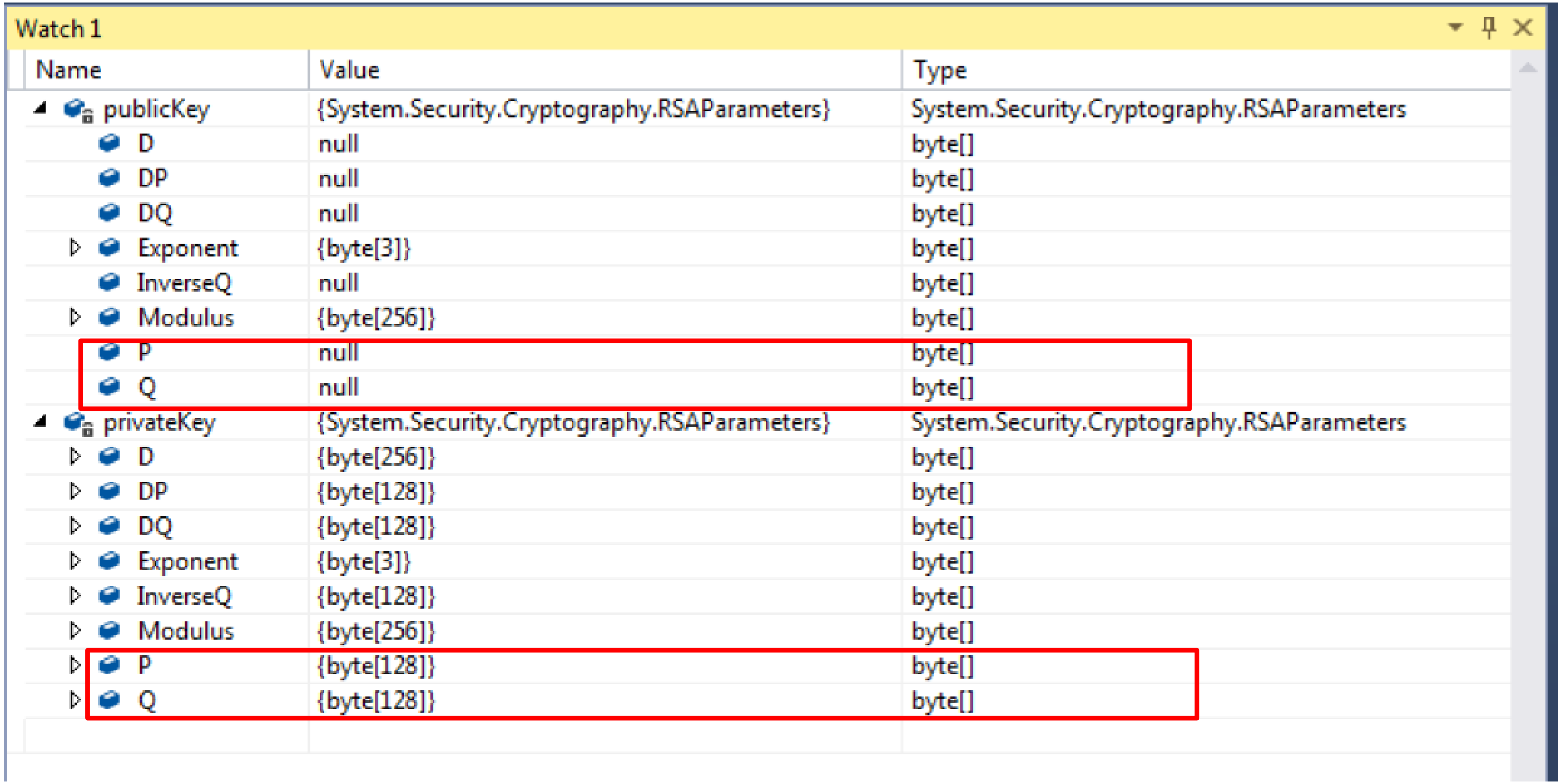
Public Key

5963

Private Key

67 and 89

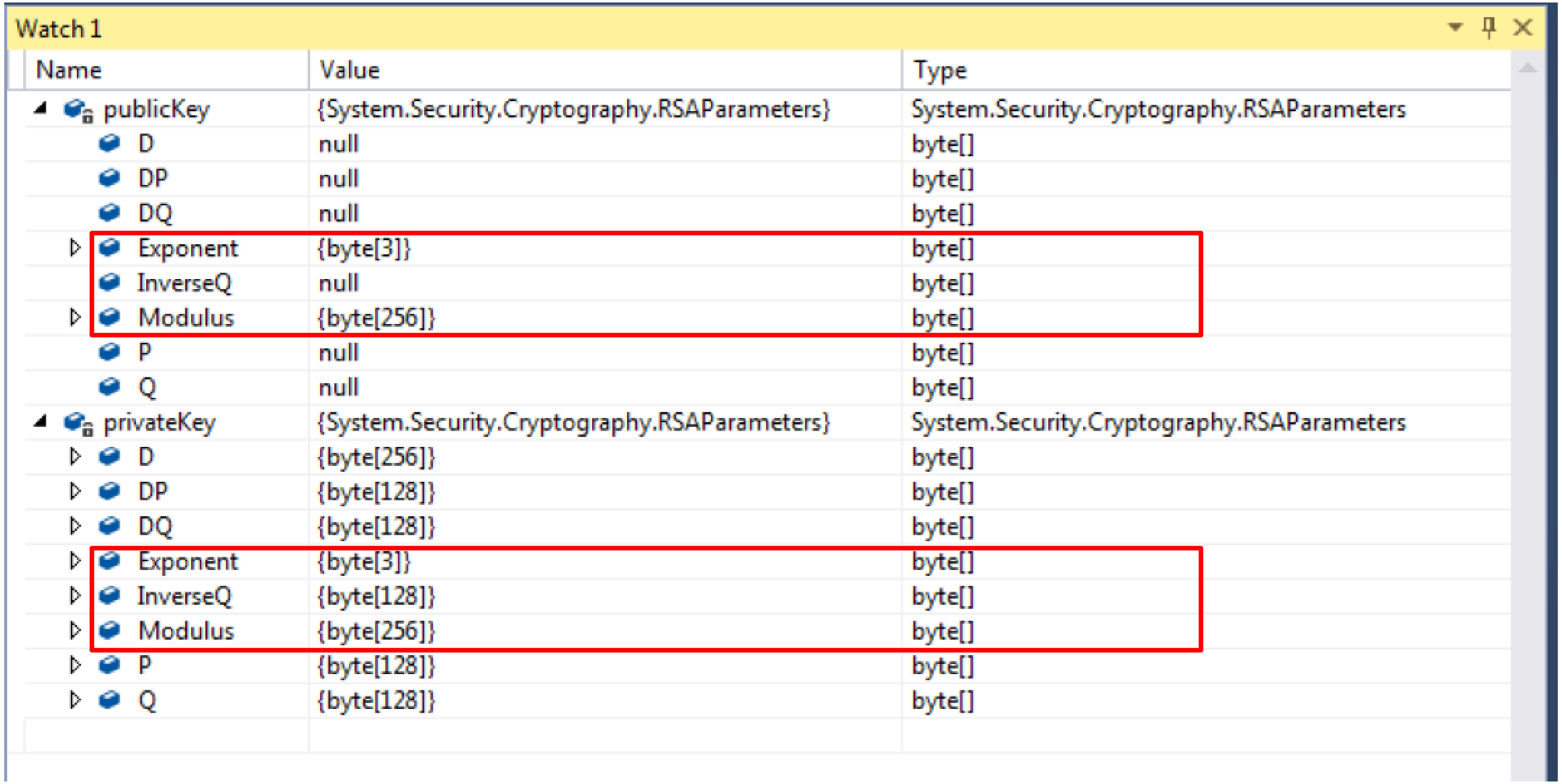
Key Derivation



Watch 1

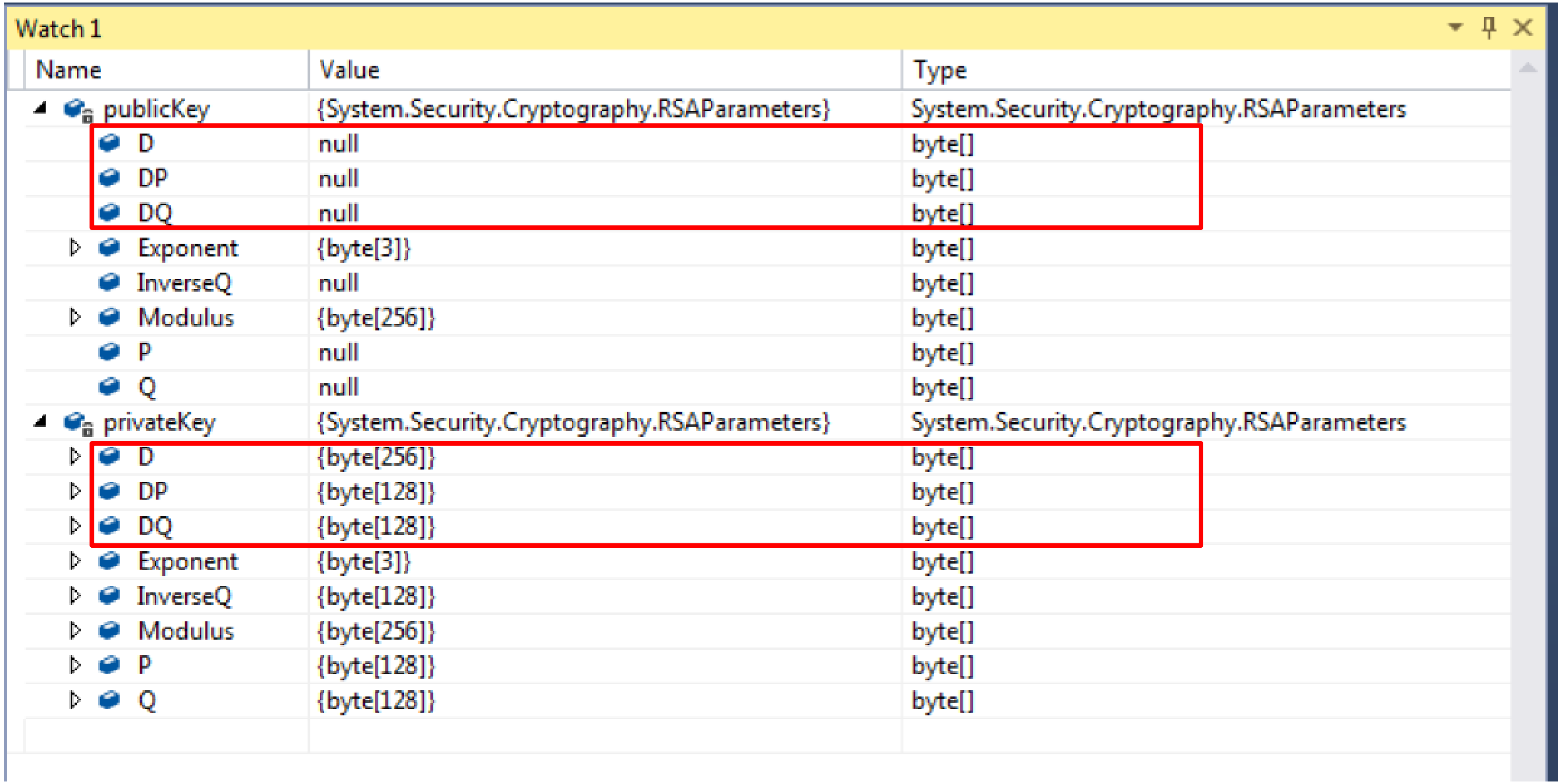
Name	Value	Type
◢ publicKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
D	null	byte[]
DP	null	byte[]
DQ	null	byte[]
▷ Exponent	{byte[3]}	byte[]
InverseQ	null	byte[]
▷ Modulus	{byte[256]}	byte[]
P	null	byte[]
Q	null	byte[]
◢ privateKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
▷ D	{byte[256]}	byte[]
▷ DP	{byte[128]}	byte[]
▷ DQ	{byte[128]}	byte[]
▷ Exponent	{byte[3]}	byte[]
▷ InverseQ	{byte[128]}	byte[]
▷ Modulus	{byte[256]}	byte[]
▷ P	{byte[128]}	byte[]
▷ Q	{byte[128]}	byte[]

Key Derivation



Name	Value	Type
◢ publicKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
D	null	byte[]
DP	null	byte[]
DQ	null	byte[]
▷ Exponent	{byte[3]}	byte[]
InverseQ	null	byte[]
▷ Modulus	{byte[256]}	byte[]
P	null	byte[]
Q	null	byte[]
◢ privateKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
▷ D	{byte[256]}	byte[]
▷ DP	{byte[128]}	byte[]
▷ DQ	{byte[128]}	byte[]
▷ Exponent	{byte[3]}	byte[]
▷ InverseQ	{byte[128]}	byte[]
▷ Modulus	{byte[256]}	byte[]
▷ P	{byte[128]}	byte[]
▷ Q	{byte[128]}	byte[]

Key Derivation

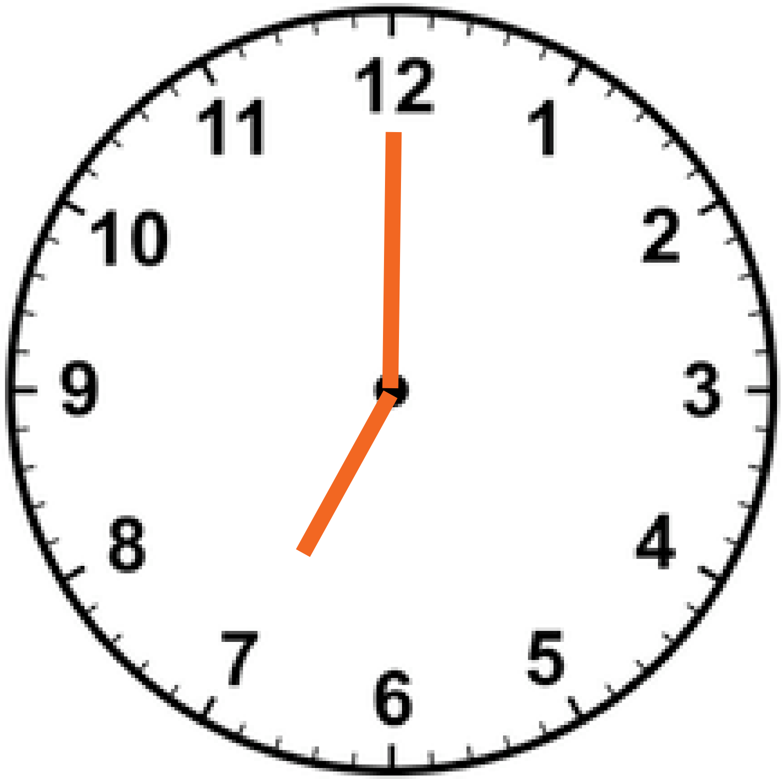


Watch 1

Name	Value	Type
◢ publicKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
D	null	byte[]
DP	null	byte[]
DQ	null	byte[]
▷ Exponent	{byte[3]}	byte[]
InverseQ	null	byte[]
▷ Modulus	{byte[256]}	byte[]
P	null	byte[]
Q	null	byte[]
◢ privateKey	{System.Security.Cryptography.RSAParameters}	System.Security.Cryptography.RSAParameters
▷ D	{byte[256]}	byte[]
▷ DP	{byte[128]}	byte[]
▷ DQ	{byte[128]}	byte[]
▷ Exponent	{byte[3]}	byte[]
▷ InverseQ	{byte[128]}	byte[]
▷ Modulus	{byte[256]}	byte[]
▷ P	{byte[128]}	byte[]
▷ Q	{byte[128]}	byte[]

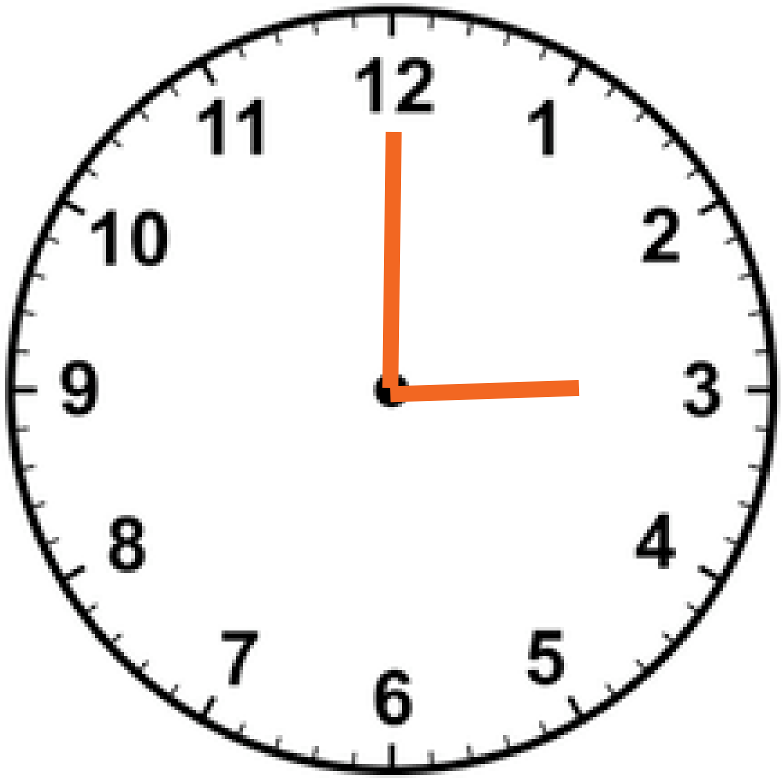
Encryption and Decryption

- RSA encryption and decryption is a mathematical operation
- Based on modular math



Encryption and Decryption

- RSA encryption and decryption is a mathematical operation
- Based on modular math



RSA in the .NET Framework

```
private RSAParameters _publicKey;
private RSAParameters _privateKey;
public void AssignNewKey()
{
    using (var rsa = new RSACryptoServiceProvider(2048))
    {
        rsa.PersistKeyInCsp = false;
        _publicKey = rsa.ExportParameters(false);
        _privateKey = rsa.ExportParameters(true);
    }
}
```

RSA in the .NET Framework

```
public void AssignNewKey()
{
    using (var rsa = new RSACryptoServiceProvider(2048))
    {
        rsa.PersistKeyInCsp = false;

        File.WriteAllText(publicKeyPath, rsa.ToXmlString(false));
        File.WriteAllText(privateKeyPath, rsa.ToXmlString(true));
    }
}
```

RSA in the .NET Framework

```
public void AssignNewKey()
{
    const int providerRsaFull = 1;

    CspParameters cspParams = new CspParameters(providerRsaFull);
    cspParams.KeyContainerName = “MyContainerName”;
    cspParams.Flags = CspProviderFlags.UseMachineKeyStore;
    cspParams.ProviderName = "Microsoft Strong Cryptographic Provider";
    var rsa = new RSACryptoServiceProvider(cspParams);
    rsa.PersistKeyInCsp = true;
}
```

RSA in the .NET Framework

```
public void DeleteKeyInCsp()
{
    var cspParams = new CspParameters();

    cspParams.KeyContainerName = “MyContainerName”;
    var rsa = new RSACryptoServiceProvider(cspParams);
    rsa.PersistKeyInCsp = false;

    rsa.Clear();
}
```

RSA in the .NET Framework

```
private RSAParameters _publicKey;

public byte[] EncryptData(byte[] dataToEncrypt)
{
    byte[] cipherbytes;

    using (var rsa = new RSACryptoServiceProvider(2048))
    {
        rsa.ImportParameters(_publicKey);
        cipherbytes = rsa.Encrypt(dataToEncrypt, false);
    }
    return cipherbytes;
}
```

RSA in the .NET Framework

```
private RSAParameters _publicKey;


public byte[] EncryptData(byte[] dataToEncrypt)
{
    byte[] cipherbytes;


    using (var rsa = new RSACryptoServiceProvider(2048))
    {
        rsa.ImportParameters(_publicKey);
        cipherbytes = rsa.Encrypt(dataToEncrypt, false);
    }
    return cipherbytes;
}
```

RSA in the .NET Framework

```
public byte[] EncryptData(byte[] dataToEncrypt)
{
    byte[] cipherbytes;
    var cspParams = new CspParameters();
    cspParams.KeyContainerName = “MyContainerName”;

    using (var rsa = new RSACryptoServiceProvider(2048, cspParams))
    {
        cipherbytes = rsa.Encrypt(dataToEncrypt, false);
    }
    return cipherbytes;
}
```

RSA in the .NET Framework

```
private RSAParameters _privateKey;
public byte[] DecryptData(byte[] dataToEncrypt)
{
    byte[] plain;
    using (var rsa = new RSACryptoServiceProvider(2048))
    {
        rsa.PersistKeyInCsp = false;
        rsa.ImportParameters(_privateKey);
        plain = rsa.Decrypt(dataToEncrypt, true);
    }
    return plain;
}
```

RSA in the .NET Framework

```
public byte[] DecryptData(byte[] dataToDecrypt)
{
    byte[] plain;
    var cspParams = new CspParameters();
    cspParams.KeyContainerName = “MyContainerName”;

    using (var rsa = new RSACryptoServiceProvider(2048, cspParams))
    {
        plain = rsa.Decrypt(dataToDecrypt, false);
    }
    return plain;
}
```

Code Demo

Using RSA in the .NET Framework

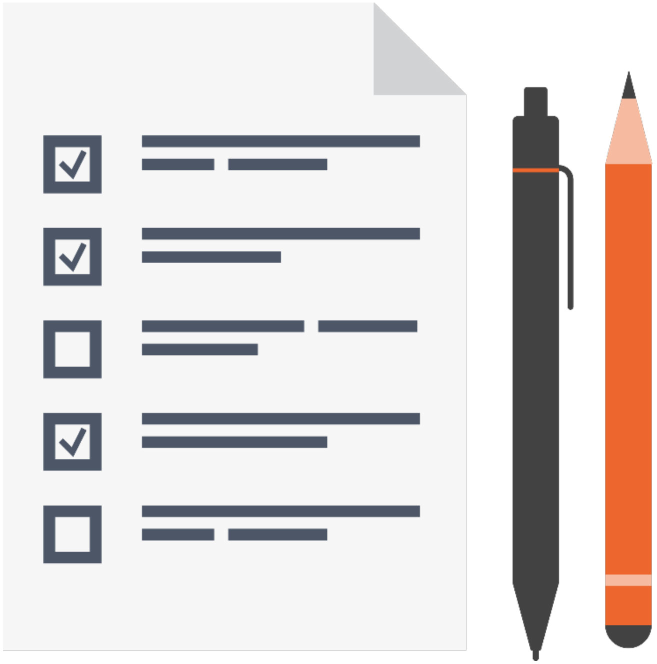
Module Summary

- Key sharing is hard with symmetric encryption
- Asymmetric encryption provides a better solution for key management
- Private and public keys

Module Summary

- Encrypt with recipient’s public key
- Recipient decrypts message with their private key
- RSA is the most popular public private key encryption system
- RSA is modular math based whereas AES etc. is more algorithmic

Module Summary



- RSA can only encrypt data up to its key length and is very slow
- Common usage is to use RSA to encrypt symmetric key
- Then use symmetric (AES) to encrypt your data