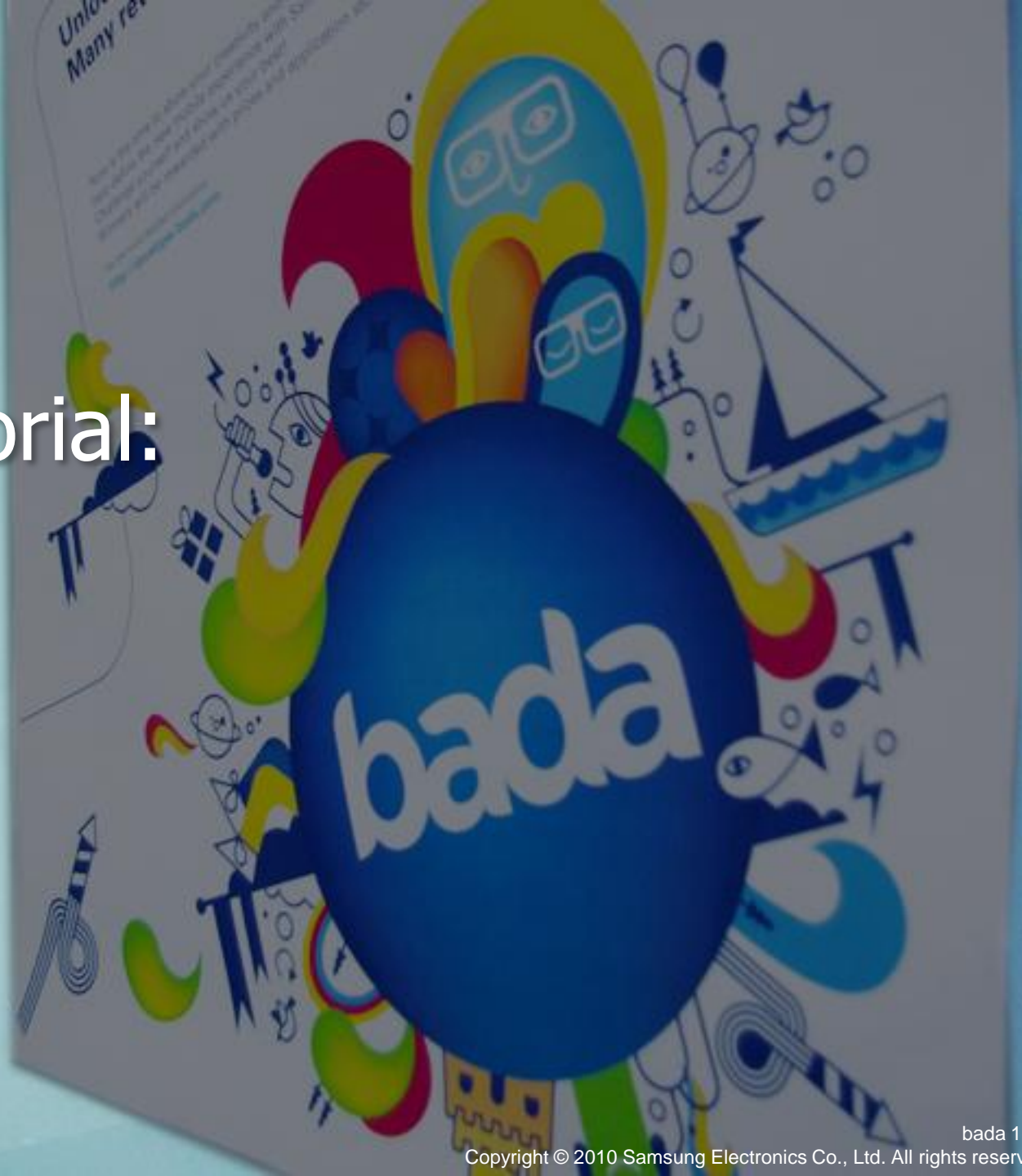


bada Tutorial: Security



Contents

- Key Management
- Cryptography
- Certificate Management



Key Management



Contents

- Essential Classes
- Relationships between Classes
- Overview
- Keys
 - Example: Create a Key
- Cryptographically Secure Pseudo Random Number Generator (CSPRNG)
 - Example: Get a Pseudo Random Number
- Review
- Answers

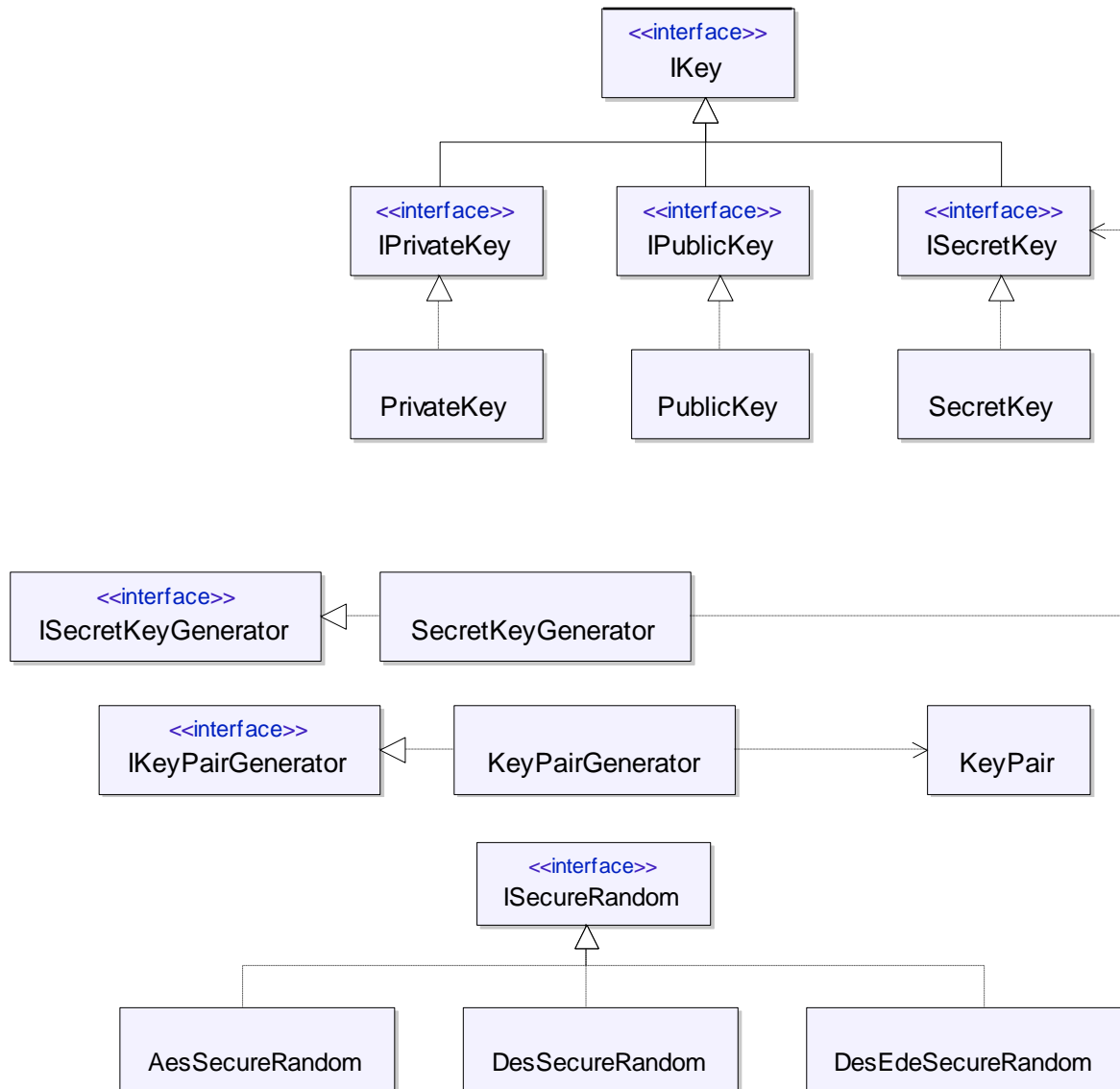


Essential Classes

| Feature | Provided by |
|---|---|
| Provides methods to create cryptographic keys. | SecretKey PublicKey PrivateKey SecretKeyGenerator KeyPair KeyPairGenerator |
| Provides cryptographically secure pseudo random numbers. | AesSecureRandom DesSecureRandom DesEdeSecureRandom |
| Provides interfaces to create cryptographic keys. | IKey IKeyPairGenerator IPrivateKey IPublicKey ISecretKey ISecretKeyGenerator |
| Provides an interface to create cryptographically secure pseudo random numbers. | ISecureRandom |



Relationships between Classes



Overview

- The Security root namespace contains tools to manage cryptographic keys and generate pseudo random numbers.
- Key feature sets include:
 - Key management for encryption, decryption, signatures, verifying signatures, and hashes
 - Cryptographically Secure Pseudo Random Number Generator (CSPRNG)



Keys

- Keys are used to deterministically seed cryptographic algorithms. Without a key, most cryptographic algorithms have no output. There are 2 kinds of keys:
 - Secret keys: These are used in symmetric encryption and hashing.
 - Key pairs: These consist of a private key and a public key.
- Users can directly supply keys, or random keys can be created with a pseudo random number.
- Samsung bada supports secret keys and RSA private/public key pairs.



Example: Create a Key

Create a secret key for use in symmetric encryption.

- Open `\<BADA_SDK_HOME>\Examples\Security\src\Crypto\CryptoExample.cpp`, `HmacSha1Example()`, `AesCbc128Example()`

1. Create a secret key generator:

```
SecretKeyGenerator()
```

2. Construct the secret key generator with a `ByteBuffer`:

```
SecretKeyGenerator::Construct(keyBytes)
```

3. Get a secret key from the generator:

```
SecretKeyGenerator::GenerateKeyN()
```



Cryptographically Secure Pseudo Random Number Generator (CSPRNG)

- All PRNGs are deterministic, and not truly random. True random number generators use external environmental input that is accepted to be random, such as atmospheric noise.
- A CSPRNG differs from a regular (PRNG) in that it creates random numbers that are sufficiently random for use in cryptography and security applications.
- You can generate pseudo random numbers in bada using any of 3 algorithms:
 - **AES:** `AesSecureRandom()`
 - **DES:** `DesSecureRandom()`
 - **3DES:** `DesEdeSecureRandom()`



Example: Get a Pseudo Random Number

Get a pseudo random number from a CSPRNG.

- Open `\<BADA_SDK_HOME>\Examples\Security\src\Key\SecureRandomExample.cpp, RandomAesExample()`

1. **Create an `AesSecureRandom` object:**

```
AesSecureRandom()
```

2. **Generate a pseudo random number:**

```
AesSecureRandom::GenerateRandomBytesN()
```



Review

1. What asymmetric algorithms in bada can you use to get a pseudo random number?
2. What other algorithms in bada can you use to get a pseudo random number?
3. Which algorithm in bada would you use to create keys for use in PKI?
4. You need to encrypt a file with a password. What kind of key would you use?

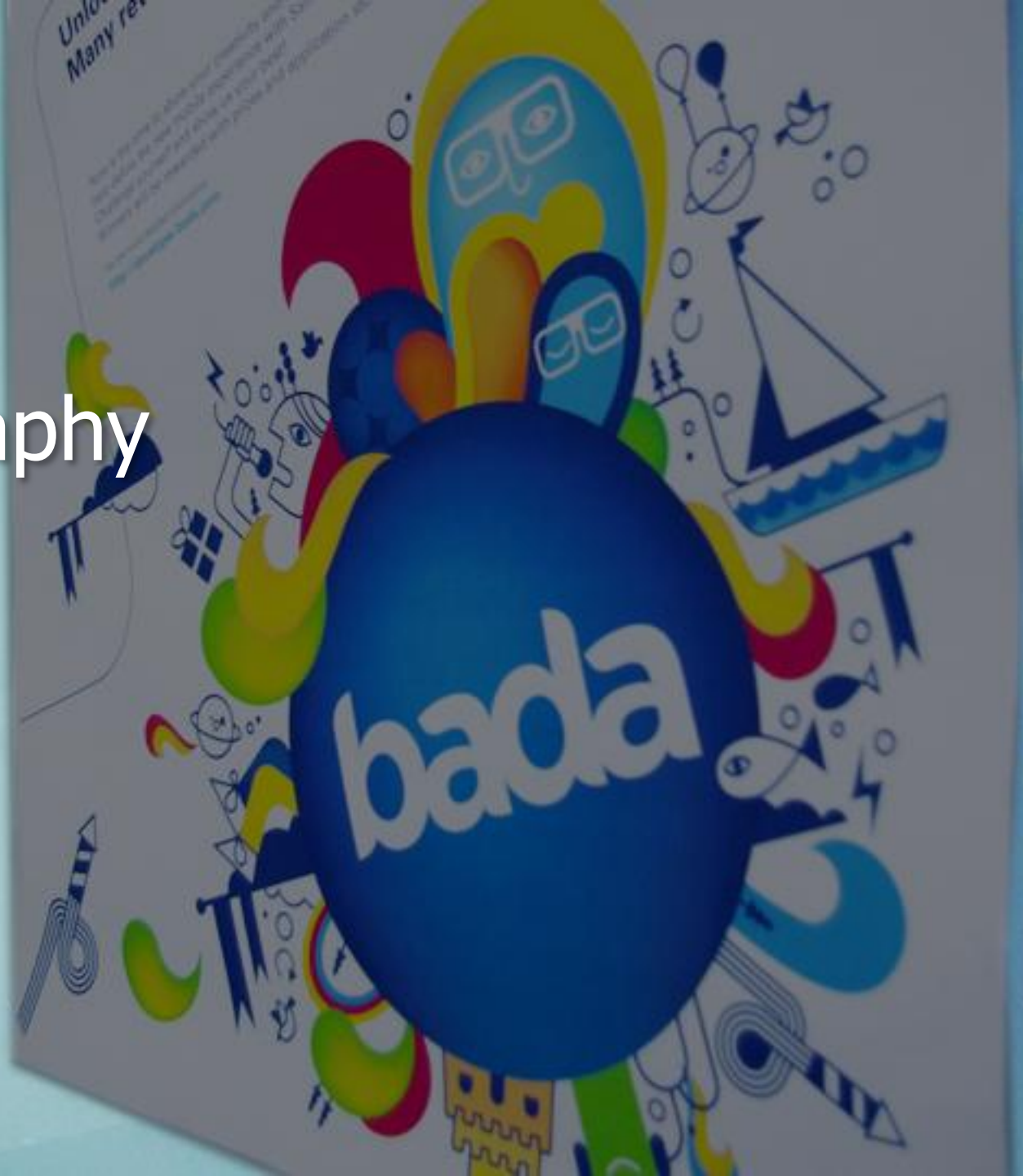


Answers

1. None. Only symmetric algorithms are used.
2. AES, DES, and 3DES.
3. RSA. PKI uses asymmetric encryption.
4. Most likely you would use a symmetric encryption scheme with a secret key. This would let you distribute the same key (the password) to everyone that you wanted to give access to the file.



Cryptography



Contents (1/2)

- Essential Classes
- Relationships between Classes
- Overview
- Encryption
- Hash
- Creating Hash and HMAC Values
 - Example: Hash a Message
- Using HMAC
 - Example: Use HMAC to Hash a Message
- Important Enumerations
 - Cipher Operations
 - Cipher Modes
 - Cipher Padding



Contents (2/2)

- Symmetric Ciphers
 - Example: Symmetrically Encrypt and Decrypt
 - Example: Symmetrically Wrap and Unwrap
- Asymmetric Ciphers
 - Example: Asymmetrically Encrypt and Decrypt
 - Asymmetric Cipher Explanation
- Digital Signatures
- FAQ
- Review
- Answers

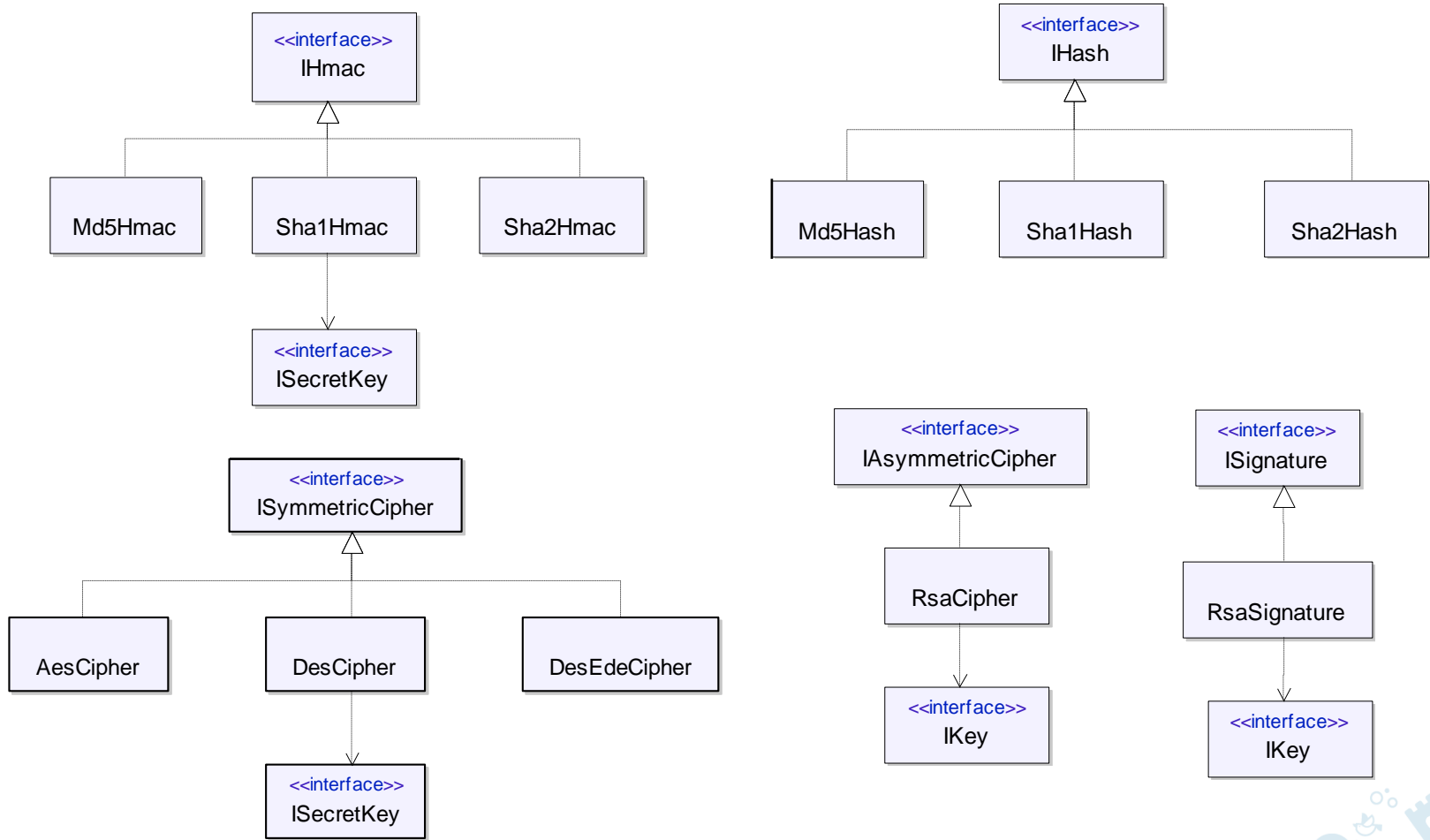


Essential Classes

| Feature | Provided by |
|--|--|
| Provides methods to create a keyed-Hash Message Authentication Code. | Md5Hmac Sha1Hmac Sha2Hmac |
| Provides hashing functions suitable for use in information security. | Md5Hash Sha1Hash Sha2Hash |
| Provides symmetric cryptographic algorithms for encryption and decryption. | AesCipher DesCipher DesEdeCipher |
| Provides an RSA cipher for asymmetric cryptography. | RsaCipher |
| Provides an RSA cipher for digital signatures to verify authenticity with a high degree of confidence. | RsaSignature |
| Provides an interface to consume HMAC algorithms. | IHmac |
| Provides an interface to consume hash algorithms. | IHash |
| Provides an interface to consume symmetric ciphers. | ISymmetricCipher |
| Provides an interface to consume asymmetric ciphers. | IASymmetricCipher |
| Provides an interface to consume digital signatures. | ISignature |



Relationships between Classes

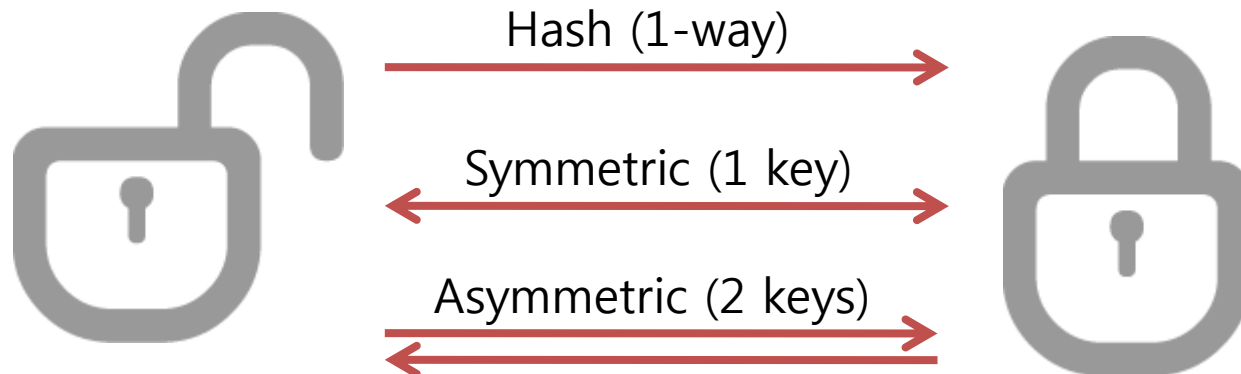


Overview

- The Crypto namespace contains a robust set of cryptography tools to handle various encryption and decryption tasks.
- Key features include:
 - Hash functions including Hash Message Authentication Code (HMAC)
 - Symmetric encryption: Single key
 - Asymmetric encryption: Public Key Infrastructure (PKI)
 - AES, DES, 3DES and RSA algorithms
 - Digital signature support
- **JARGON ALERT:** When talking about encryption, a “message” is the input that you encrypt. This is a distinctly different meaning for “message” than the one used in communication messages, such as SMS or email messages. A message in cryptography can be any kind of data, such as images, audio, video, text, databases, files, or data streams.

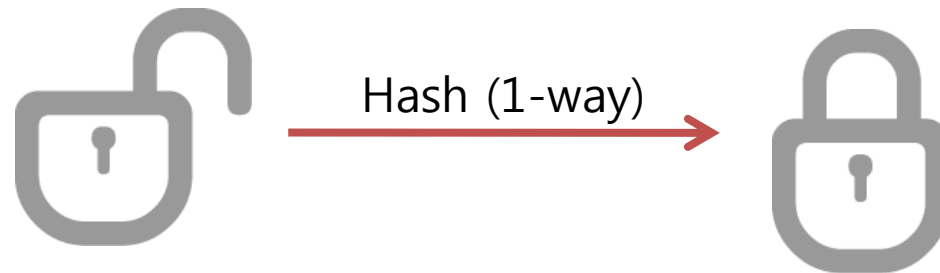
Encryption

- There are many varieties of encryption with different purposes. Encryption can keep information secret, but is also used for verification.
- Samsung bada provides 4 basic types of encryption:
 - One-way encryption
 - Symmetric encryption (uses 1 secret key)
 - Asymmetric encryption (uses a private key and a public key)
 - Digital signatures (used for verification)



Hash

- A hash is one-way deterministic encryption used for verification or indexing. You hash messages into “message digests” (MD). You cannot recover the original message from a message digest.



- Small changes to messages result in radical changes to the message digest.
- It is exceedingly rare for any two messages to result in the same message digest, but it happens. These are called “collisions”.
- Supported hash algorithms include:
 - MD5 (Message-Digest algorithm 5)
 - SHA-1 (Secure Hash Algorithm)
 - The SHA-2 family (SHA-224, SHA-256, SHA-384, SHA-512)

Creating Hash and HMAC Values

- Basic hash procedure:
 1. Choose an algorithm.
 2. Calculate the hash value. For operations with multiple parts:
`Initialize()`, `Update()`, `Finalize()`.
- A message authentication code (MAC) is used to verify the integrity of a message. It comes from a message input and a secret key.
- A Hash Message Authentication Code (HMAC) is essentially just a hash that includes a secret key.
- The HMAC procedure is little different from normal hashing:
 1. Choose an algorithm.
 2. If using HMAC, set a secret key.
 3. Calculate the hash value. For operations with multiple parts:
`Initialize()`, `Update()`, `Finalize()`.



Example: Hash a Message

Calculate a hash value for a message using the MD5 algorithm.

- Open `\<BADA_SDK_HOME>\Examples\Security\src\Crypto\CryptoExample.cpp, HashSha1Example()`

1. Calculate the hash value:

```
Md5Hash::GetHashN()
```



Using HMAC

- Hashing with HMAC uses interfaces for the HMAC and secret key. They must be assigned with a concrete class as shown below and in the following example.
- Choose between HMAC-SHA2, HMAC-MD5 and HMAC-SHA1. For SHA-2, choose a specific SHA-2 algorithm.
- Memory for the hash is allocated using `Crypto::IHmac` and the hash is constructed using `Crypto::Sha1Hmac`.
- Construct a key generator with `Security::SecretKeyGenerator`.
- Use the `Security::SecretKeyGenerator::GenerateKeyN()` method to assign the secret key to an `ISecretKey` instance.
- Set the key to the HMAC using the `Sha1Hmac::SetKey` method.
- Calculate the HMAC using its `GetHmacN()` method.
- Multiple-part operation uses `Initialize()`, `Update()`, and `Finalize()`.

Example: Use HMAC to Hash a Message

Create an HMAC hash value for a message using SHA-1.

- Open `\<BADA_SDK_HOME>\Examples\Security\src\Crypto\CryptoExample.cpp, HmacSha1Example()`

1. **Generate a secret key to use when calculating the HMAC hash value:**

```
SecretKeyGenerator::GenerateKeyN()
```

2. **Set the secret key:**

```
Sha1Hmac::SetKey()
```

3. **Calculate the HMAC value:**

```
Sha1Hmac::GetHmacN()
```



Cipher Operations

- Cipher operations are used in constructing symmetric ciphers and are enumerated for ease-of-use:

| Crypto::CipherOperation | Description |
|-------------------------|--------------|
| CIPHER_ENCRYPT | Encrypt mode |
| CIPHER_DECRYPT | Decrypt mode |
| CIPHER_WRAP | Wrap mode |
| CIPHER_UNWRAP | Unwrap mode |

- Encrypt and decrypt are self-explanatory.
- Wrapping and unwrapping are analogous to encrypting and decrypting, respectively, but use with symmetric keys.

Cipher Modes

- Cipher modes in bada are either cipher block chaining (CBC) or electronic codebook (ECB):

| Crypto::CipherMode | Description |
|--------------------|-----------------------|
| CBC | Cipher block chaining |
| ECB | Electronic codebook |

- ECB does not hide patterns well, and is not recommended for use in any security protocol.
- Use CBC for security and ECB for non-security related encryption.



Cipher Padding

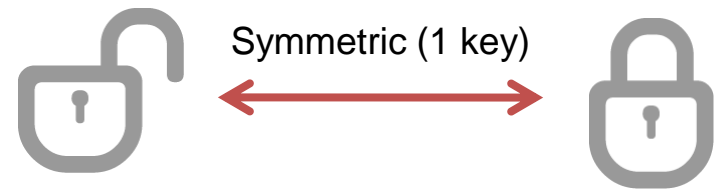
Ciphers support the following padding schemes:

| Crypto::CipherPadding | Description |
|-----------------------|--|
| NOPADDING | No padding is used. |
| PKCS7PADDING | The PKCS7 padding string consists of a sequence of bytes. Each sequence is equal to the total number of padding bytes added. |



Symmetric Ciphers (1/2)

- Symmetric ciphers are used in stereo-typical encryption with a single key used for encryption and decryption.
- bada provides 3 symmetric ciphers:
 - AES
 - DES
 - 3DES
- Each symmetric cipher is constructed with a transformation and a cipher operation.
 - Transformations include the following options:
 - Encryption mode (CBC or ECB)
 - Key bit size (128, 192, or 256)
 - Padding scheme (no padding or PKCS7 padding)
 - Cipher operations are:
 - Encrypt or decrypt
 - Wrap or unwrap



Symmetric Ciphers (2/2)

Basic encryption in bada with a symmetric cipher:

- Choose an algorithm and use its corresponding class to construct the cipher with a “Transformation” and a “CipherOperation”:
 - AES (`AesCipher`)
 - DES (`DesCipher`)
 - 3DES (`DesEdeCipher`)
- Use `Security::SecretKeyGenerator` and `Security::ISecretKey` to generate a secret key to encrypt and decrypt the message.
- Optionally, set an initialization vector (IV) if required. (Needed for CBC, but not ECB.)
- Handle multiple-part operations with the `Initialize()`, `Update()`, and `Finalize()` methods.



Example: Symmetrically Encrypt and Decrypt

Encrypt and decrypt a message using a secret key and AES with cipher block chaining (CBC).

– Open `<BADA_SDK_HOME>\Examples\Security\src\Crypto\CryptoExample.cpp, AesCbc128Example()`

1. Construct a symmetric cipher object:

`AesCipher::Construct()`

2. Generate a secret key to encrypt or decrypt the message with:

`SecretKeyGenerator::GenerateKeyN()`

3. Set the secret key to the cipher:

`AesCipher::SetKey()`

4. Set an initialization vector (required for non-ECB):

`AesCipher::SetInitialVector()`

5. Encrypt or decrypt the message:

`AesCipher::EncryptN|DecryptN()`

Example: Symmetrically Wrap and Unwrap

Wrap and unwrap using symmetric keys:

1. Construct a symmetric cipher object:

```
AesCipher::Construct()
```

2. Generate a secret key to wrap or unwrap the message with:

```
SecretKeyGenerator::GenerateKeyN()
```

3. Set the secret key to the cipher:

```
AesCipher::SetKey()
```

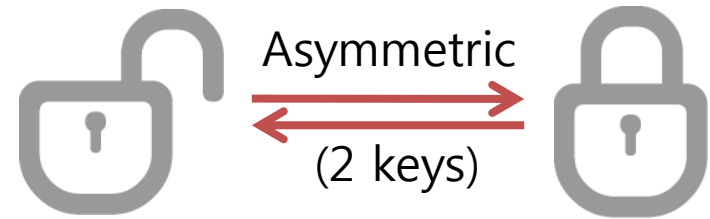
4. Wrap or unwrap a key:

```
AesCipher::WrapN|UnwrapN()
```



Asymmetric Ciphers (1/2)

- An asymmetric cipher uses 2 keys:
 - Public keys
 - Private key



- There are 2 basic uses of asymmetric ciphers:
 - Public-key encryption for sending a confidential message
 - Digitally signing a message
- Public-key encryption encrypts a message so that only the person with the corresponding private key can read the message.
- Using a private key to encrypt a message provides a digital signature for it. Whoever has the public key can open the message. This does not provide confidentiality. It only provides verification that the person with the private key encrypted the message.

Asymmetric Ciphers (2/2)

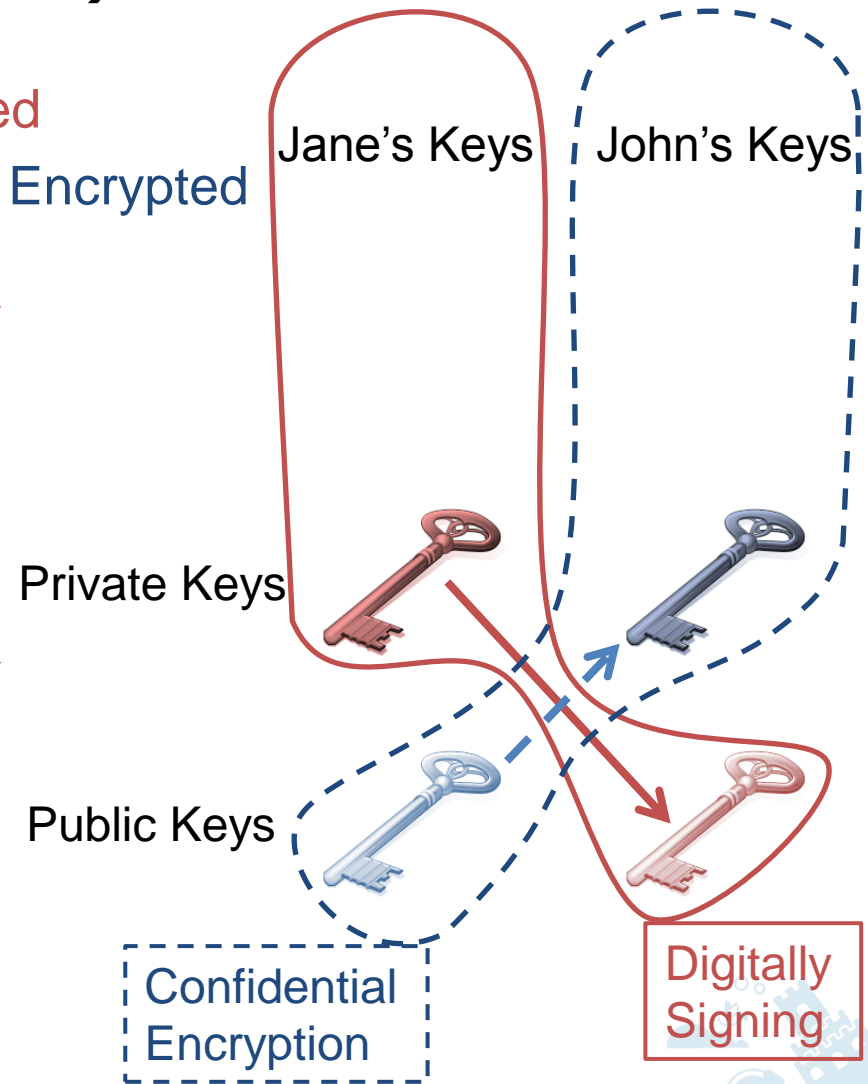
- Private to Public = Digitally Signed
- Public to Private = Confidentially Encrypted

→
• Jane sends a **digitally signed** message with her **private** key.

• John verifies the message from Jane with her **public** key.

→
• Jane sends a **confidentially encrypted** message to John with his **public** key.

• John decrypts the message with his **private** key.



Example: Asymmetrically Encrypt and Decrypt

Asymmetrically encrypt and decrypt a message using a public and a private key.

– Open `\<BADA_SDK_HOME>\Examples\Security\src\Crypto\CryptoExample.cpp`, `RsaCipherExample()`

1. **Generate a key pair:**

```
KeyPairGenerator::GenerateKeyPairN()
```

2. **Get a public and a private key from the key pair:**

```
KeyPair::GetPublicKey()
```

```
KeyPair::GetPrivateKey()
```

3. **Set the public key for the cipher:**

```
RsaCipher::SetPublicKey()
```

4. **Encrypt the message:**

```
RsaCipher::EncryptN()
```

5. **Set the private key for the cipher:**

```
RsaCipher::SetPrivateKey()
```

6. **Decrypt the message:**

```
RsaCipher::DecryptN()
```



Digital Signatures

- Digital signatures allow messages to be sent over insecure channels, and verified at the receiving end. Digital signatures use asymmetric encryption.
- Private keys sign messages.
- Public keys verify signatures.
- Use the RSA algorithm to sign messages or verify signatures.
- Set an asymmetric key:
 - Set a private key to sign the plain message.
 - Set a public key to verify the digitally signed message.
 - `IKey` interface is used. `IPublicKey` and `IPrivateKey` are derived from `IKey`.



FAQ

Which algorithms are supported?

- Hash and HMAC:
 - SHA-1, SHA-2, MD5,
- Symmetric ciphers:
 - AES, DES, 3DES
- Asymmetric ciphers:
 - RSA
- Digital signatures:
 - RSA



Review (1/2)

1. You are a secret agent and need to send a top secret file to one of your spies by email (an insecure method). What kind of encryption do you use?
2. You receive an encrypted email from your spy handler. You know that your handler used the `RsaCipher::SetPrivateKey()` method to encrypt it. You also know that the email was intercepted by the enemy. Were they able to read it?
3. Having learned from his mistakes, your spy handler stops sending you emails encrypted with his private key. Instead, your handler writes an email then uses the `Md5Hash::GetHashN()` method. How can you read the message?
4. Your handler sends the message again, but this time using symmetric encryption and a password that you know. He used this command: `AesCipher::Construct("ECB/128/NOPADDING", CIPHER_ENCRYPT)`. What did he do wrong? (Hint: Read #2 above.)

Review (2/2)

5. You hear that your spy handler has been shot (hopefully, but not likely, by the enemy and not your own side), and moments later you get an encrypted email from the regional spy-master. He is much smarter than your (previous) boss. What do you use to decrypt the email?



Answers

1. You can use symmetric encryption or if you have the recipient's public key, you can use that to asymmetrically encrypt it.
2. Yes. If the enemy has your handler's public key (which is highly likely), then they could read it. Using private keys to encrypt messages is only for proving the identity of the person doing the encrypting, and not for confidentiality.
3. You cannot read it. Hashes are 1-way encryption. There is no way to get the original message back from a hash value.
4. You already know that your emails are being intercepted by the enemy, so your handler should have used "CBC" instead of "ECB" and also used an initialization vector. ECB is not considered secure.
5. There are two possibilities:
 - a. Use the `RsaCipher::DecryptN()` method with your private key, assuming he used your public key.
 - b. Use a key that both you and your spy-master know.

Contents

- Essential Classes
- Relationships between Classes
- Overview
- Certificates
 - Example: Get Information from a Certificate
- FAQ
- Review
- Answers

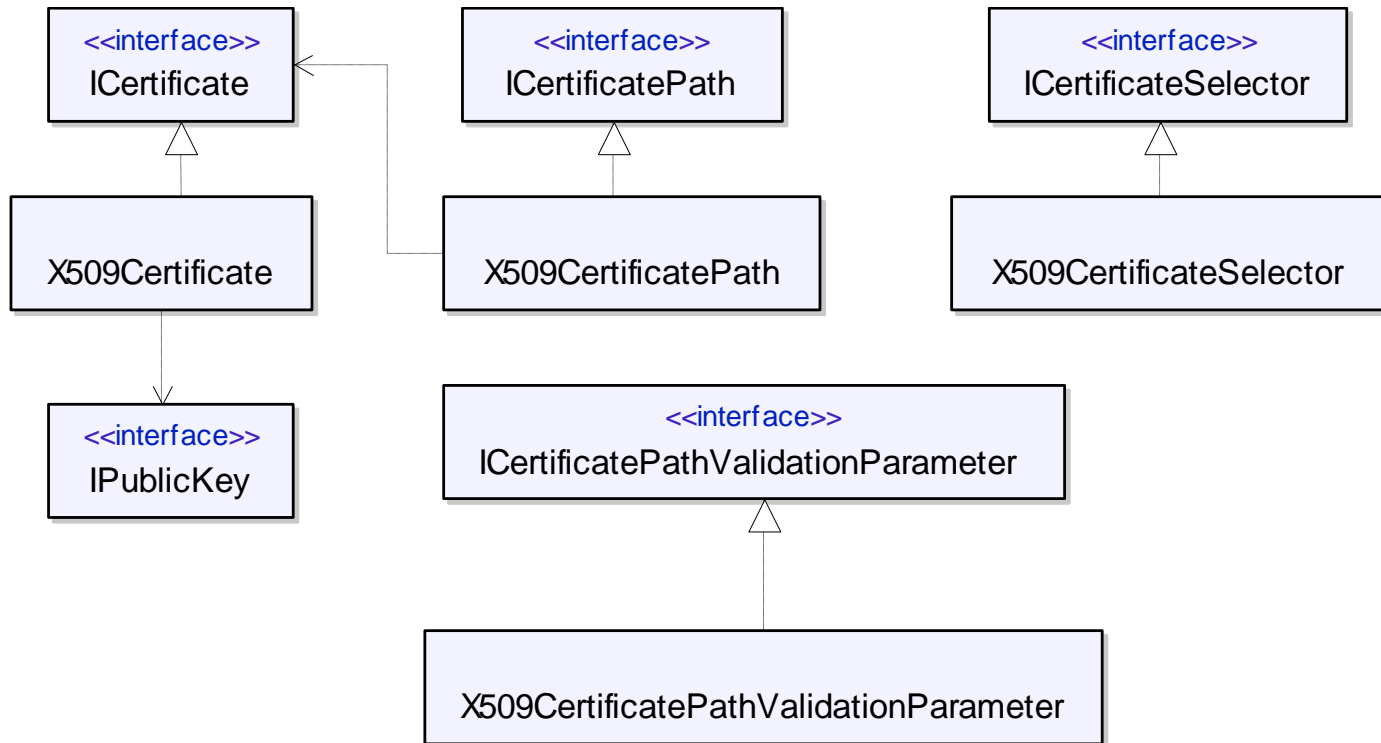


Essential Classes

| Feature | Provided by |
|---|---|
| Represents an X509 certificate. | X509Certificate |
| Represents an X509 certificate storage. | X509CertificateSelector X509CertificateStore |
| Represents a certificate. | ICertificate |
| Represents a certificate storage. | ICertificateSelector |



Relationships between Classes



Overview

- The Cert namespace lets you get information from security certificates.
- Key features include:
 - X.509 certificates.



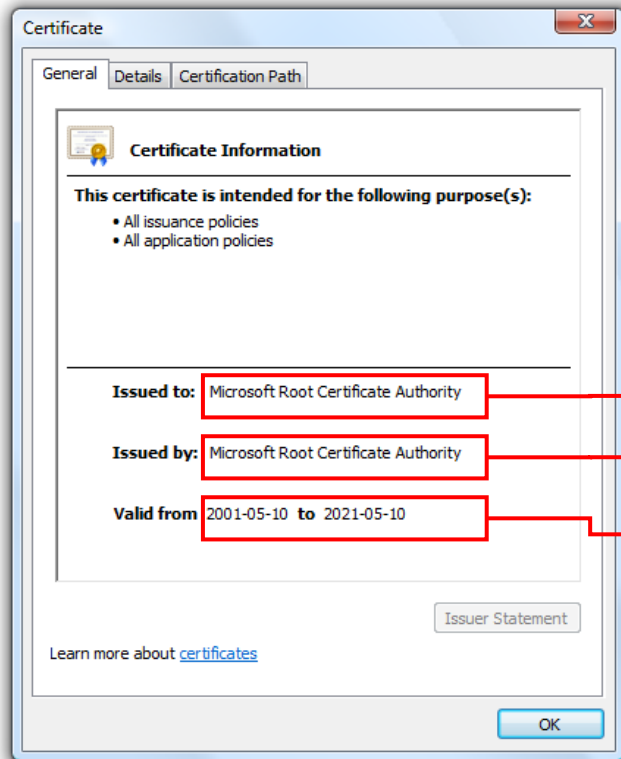
Certificates (1/2)

- Public key certificates (also called digital certificates or identity certificates) are used to validate the identities of people and organizations.
- Validation is performed by binding a public key to an identity and then verifying the authenticity through a path of certificates. The path leads to a root certificate issued by a certificate authority (CA). Trust in the identity then relies on trust in the root certificate and certification authority.
- Certificates can come in many forms. Some common certificate file extensions are .der, .crt, .pem, .p12, and .cer.

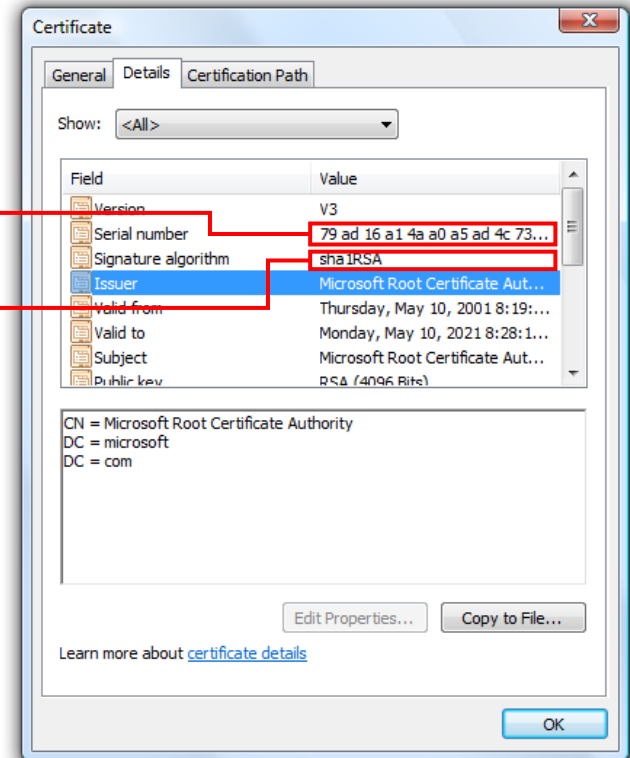


Certificates (2/2)

Some information that you can get from a certificate includes:



- Serial Number (GetSerialNumber)
- Signature algorithm (GetSignatureAlgorithm)
- Subject name (GetSubject)
- Issuer name (GetIssuer)
- Validity (GetNotBefore, GetNotAfter)



Example: Get Information from a Certificate

Load a certificate and get some information from it.

- Open `\<BADA_SDK_HOME>\Examples\Security\src\Cert\CertExample.cpp, CertificateExample()`

1. **Get a certificate file into a ByteBuffer:**

```
File::Read(input)
```

2. **Construct a certificate object with the input:**

```
X509Certificate::Construct(input)
```

3. **Get information from the certificate:**

```
X509Certificate::GetSerialNumber()
```

```
X509Certificate::GetSignatureAlgorithm()
```

```
X509Certificate::GetNotBefore()
```

```
X509Certificate::GetNotAfter()
```

```
X509Certificate::GetSubject()
```

```
X509Certificate::GetIssuer()
```

```
X509Certificate::GetSignatureN()
```

```
X509Certificate::GetFingerprintN()
```



FAQ

Can I validate a revoked certificate?

- No. Samsung bada does not support CRL or OCSP.



Review

1. True or false. Certificates have only 5 forms: *.der, *.crt, *.pem, *.p12, and *.cer.
2. True or false. You can store passwords in certificate stores.
3. True or false. You cannot issue your own certificates. You must purchase certificates from a CA.



Answers

1. False. Certificates can be stored in any number of ways. Those are merely 5 different file types that contain certificates.
2. Trick question. Passwords are not relevant to the way the question is phrased. You merely store information in certificates. That information is used to identify someone or something.
3. False. You can issue your own certificates for free using commonly available software. However, your certificate will not be in a trusted path.





bada

The platform with more opportunities
Invitation to Adventure