ENT170

Securing Local Data

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Store secrets with Xamarin.Auth
2. Introduce PCLCrypto
3. Hash passwords with PCLCrypto
4. Encrypt/decrypt data with PCLCrypto

Store secrets with Xamarin.Auth

Xamarin University

Tasks

1. Introduce Xamarin.Auth
2. Create an **`AccountStore`**
3. Store a user account
4. Retrieve an account
5. Delete an account

Motivation [personal and sensitive data]

- Some data might damage the user or hurt your app's credibility if it was revealed publicly

Credit cards
Bank details
Passwords
OAuth tokens
…

Name
ID numbers
Physical address
Email address

Height
Age
Eye color
Medical history
…

Appointments
Photos
Diary entries
Reminders
…

Some data has legal requirements (e.g. financial and medical); consult with legal and security experts when handling this type of information

Motivation [data security]

- If you store the user's personal data on-device, you should do it safely

Motivation [platform-specific APIs]

- Each platform has APIs to encrypt and store data on-device

iOS

KeyChain

KeyStore

PasswordVault

Using platform-specific APIs requires learning multiple styles and many special cases (e.g. PasswordVault on Windows has a limit on the number of stored entries)

What is Xamarin.Auth?

- **Xamarin.Auth** provides **on-device storage** of user information

Open source, but maintained by Xamarin

Cross-platform

Can be installed through Xamarin Component Store or Nuget

Xamarin.Auth also helps you do OAuth; however, that capability is not discussed here.

Xamarin.Auth security

- Xamarin.Auth uses secure, platform-specific APIs to store user data

iOS: stored in **KeyChain** - a secure database for passwords, certificates, etc.

Android: stored in a **`KeyStore`** which is encrypted and saved to a private file

Windows: data is encrypted and saved in **Isolated Storage** files

What is an Account?

- An **`Account`** represents a collection of **user information**

```
public class Account
{ ...
  public virtual string                          Username   { get; set; }
  public virtual Dictionary<string, string> Properties { get; private set; }
  public virtual CookieContainer                 Cookies    { get; private set; }
}
```

Xamarin.Auth source

You should provide a user name, but the other values are optional.
E.g. **`Cookies`** are for accounts that represent web-service registrations.

How to create an Account

- You create an **`Account`** object and load it with **user data**

```
var account = new Account();
account.Username = "Ann";
  ...
account.Properties.Add("password", "...");
account.Properties.Add("access_token", "...");
```

Can store arbitrary data in the **`Properties`** dictionary: passwords, OAuth tokens, encryption keys, etc.

What is AccountStore?

- **AccountStore** provides persistent storage for **Accounts**

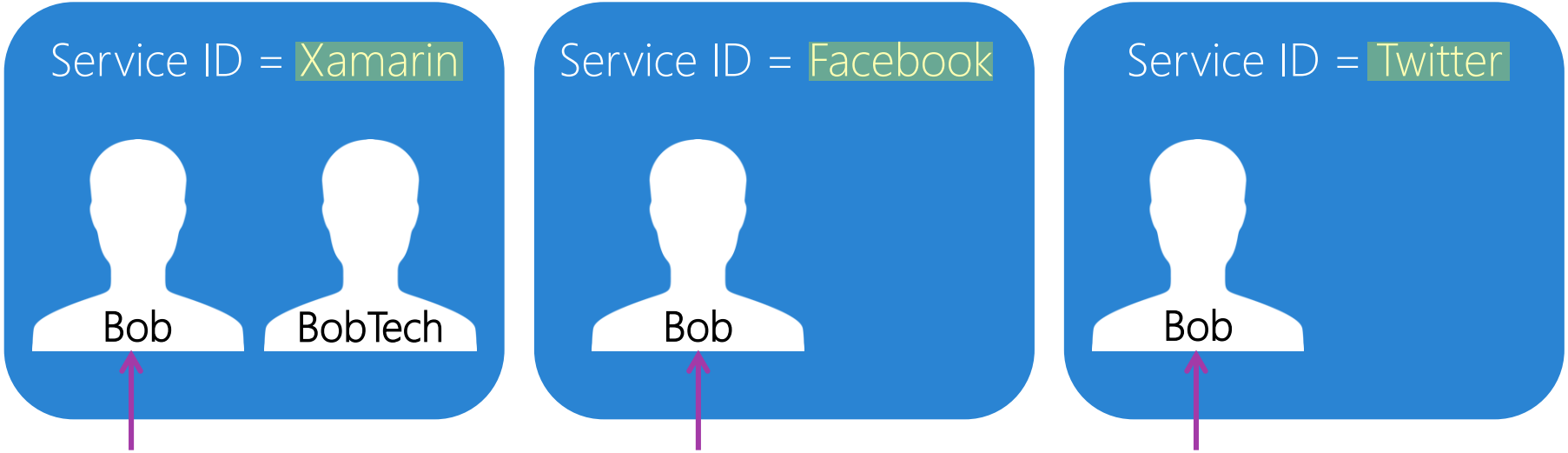
```
public abstract class AccountStore
{
  public abstract IEnumerable<Account> FindAccountsForService(
                                                  string serviceId);

  public abstract void Save  (Account account, string serviceId);
  public abstract void Delete(Account account, string serviceId);
  ...
}
```

Xamarin.Auth source

What is Service ID?

- A *Service ID* is a string of your choice that lets you group related accounts



The **`Username`** can be repeated as long as the **`serviceId`** is different

Specifying Service ID

- You include your chosen **Service ID** with all **`AccountStore`** operations

```
public abstract class AccountStore
{
  public abstract IEnumerable<Account> FindAccountsForService(
                                                  string serviceId);

  public abstract void Save  (Account account, string serviceId);
  public abstract void Delete(Account account, string serviceId);
  ...
}
```

Xamarin.Auth source

Account identity

- **`AccountStore`** combines the **`Account.Username`** and **`serviceId`** to form the identity of the entry (analogous to a primary key)

Both contribute to an identifier for the account

```
public abstract class AccountStore
{
  public abstract void Save   (Account account, string serviceId);
  public abstract void Delete(Account account, string serviceId);
  ...
}
```

Xamarin.Auth source

How to create an AccountStore [iOS, Windows]

- Use **`AccountStore.Create()`** to create an **`AccountStore`** in iOS and WinPhone projects

```
...
var store = AccountStore.Create();
...
```

No arguments needed for creation in iOS and WinPhone

How to create an AccountStore [Android]

- Use **`AccountStore.Create(context)`** to create an **`AccountStore`** in Android projects

```
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        var store = AccountStore.Create(this);
        ...
    }
    ...
}
```

`AccountStore.Create` requires a **`Context`** in Android

How to create an AccountStore [PCL]

- When Xamarin.Auth is used within a PCL, it will create the correct AccountStore for the executing platform

```
...
var store = AccountStore.Create();
...
```

No arguments needed when called from a PCL

How to store an Account

- To store user data, load it into an **`Account`** object and **save** it in the **`AccountStore`**

```
var account = new Account();
account.Username = "Ann";
account.Properties.Add("password", "...");
account.Properties.Add("access_token", "...");

AccountStore store = ... // Creation varies by platform
store.Save(account, "Xamarin");
```

Save this account and associate it with this service ID

How to retrieve Accounts

- To retrieve an **`Account`**, first **get all accounts** for a service ID, then filter by **`Username`**

```
AccountStore store = ... // Creation varies by platform

var accounts = store.FindAccountsForService("Xamarin");

foreach (var account in accounts)
  if (account.Username == "Ann")
    ... // found
```

`AccountStore` gives you all accounts for a service ID, you must then search for the username you want

How to delete an Account

- To delete an account, create an **`Account`** with the **`Username`**, then pass the account and service ID to the **`AccountStore`**'s **`Delete`** method

```
var account = new Account();
account.Username = "Ann";

AccountStore store = ... // Creation varies by platform
store.Delete(account, "Xamarin");
```

The username in the account is combined with the service ID to determine the identity of the account to delete

Individual Exercise

Store user data with Xamarin.Auth

Xamarin University

Summary

1. Introduce Xamarin.Auth
2. Create an **`AccountStore`**
3. Store a user account
4. Retrieve an account
5. Delete an account

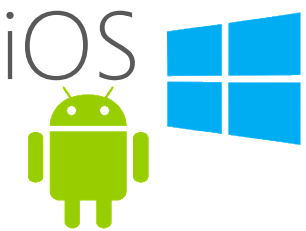
Introduce PCLCrypto

Tasks

1. Introduce PCLCrypto
2. Survey its capabilities

Motivation

- You have multiple APIs to choose from when implementing cryptography in your cross-platform code



Each platform includes a platform-specific API

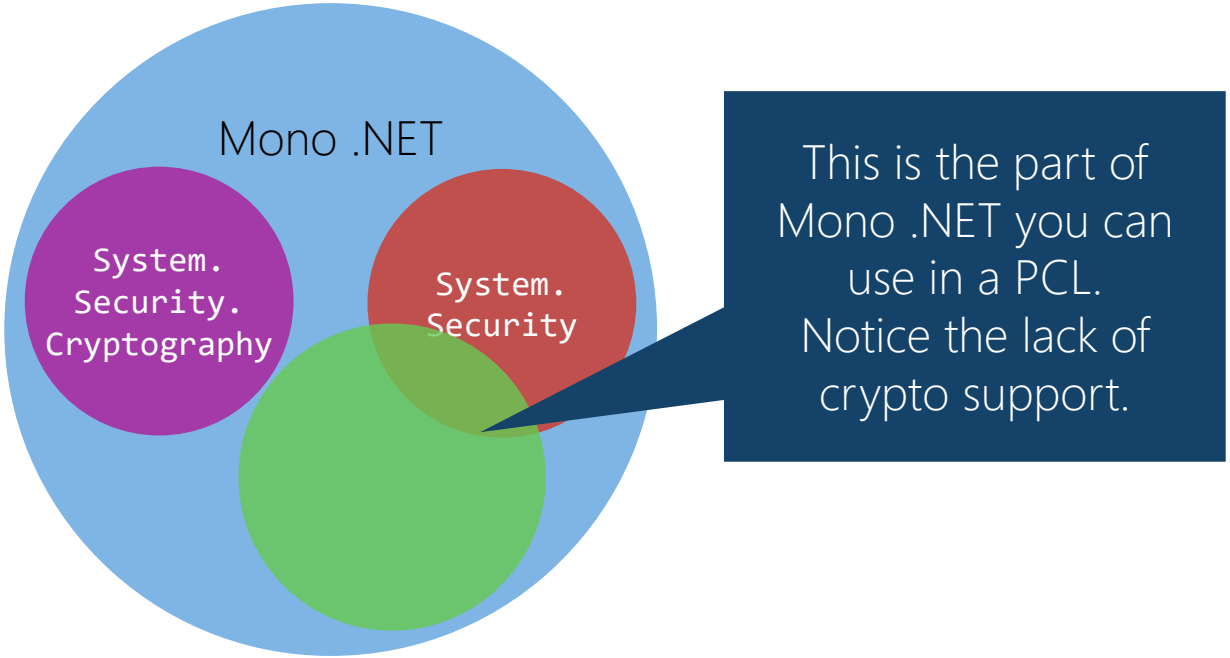
3rd-party libraries (e.g. Bouncy Castle)

3rd-party layer over existing APIs

Mono provides crypto APIs

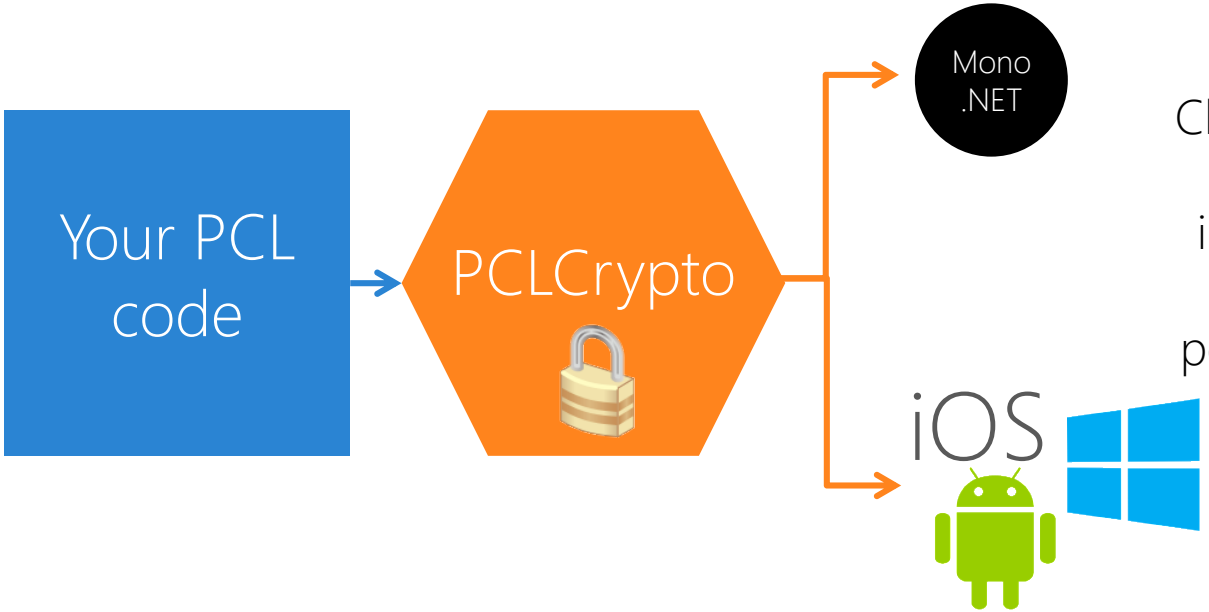
Why not .NET crypto?

- Crypto APIs are included in Mono; however, most of them **are not available** to your PCL cross-platform code



What is PCLCrypto?

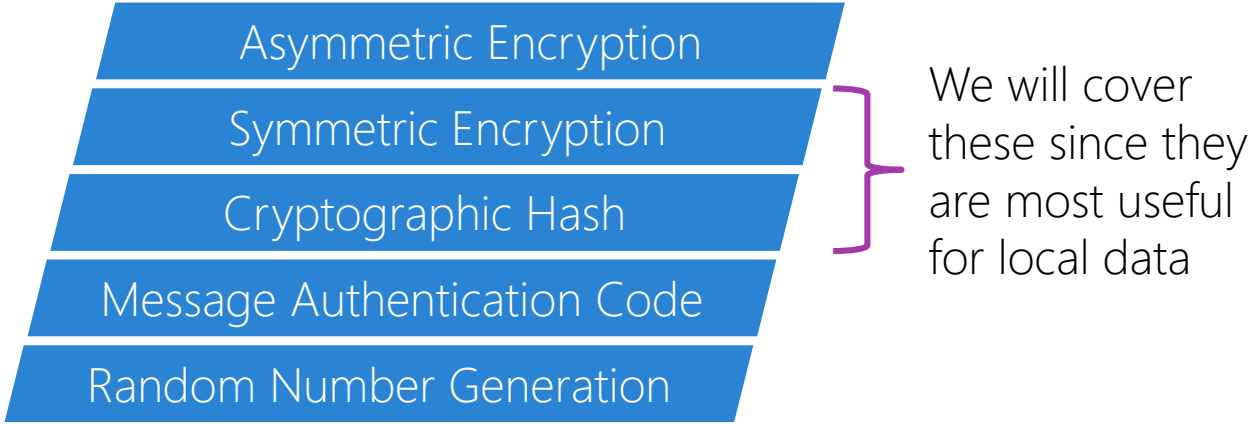
- PCLCrypto is an open-source project that puts a uniform interface over the platform and Mono crypto APIs



Chooses Mono or platform implementation based on performance and availability

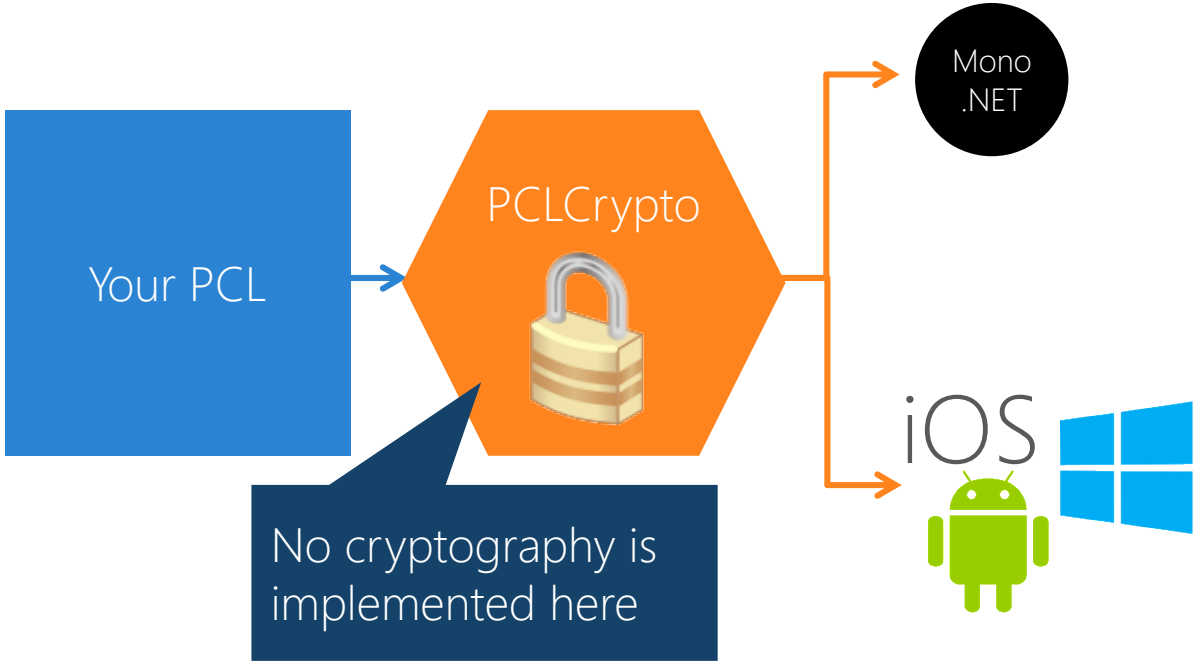
PCLCrypto services

- PCLCrypto supports most common cryptographic operations



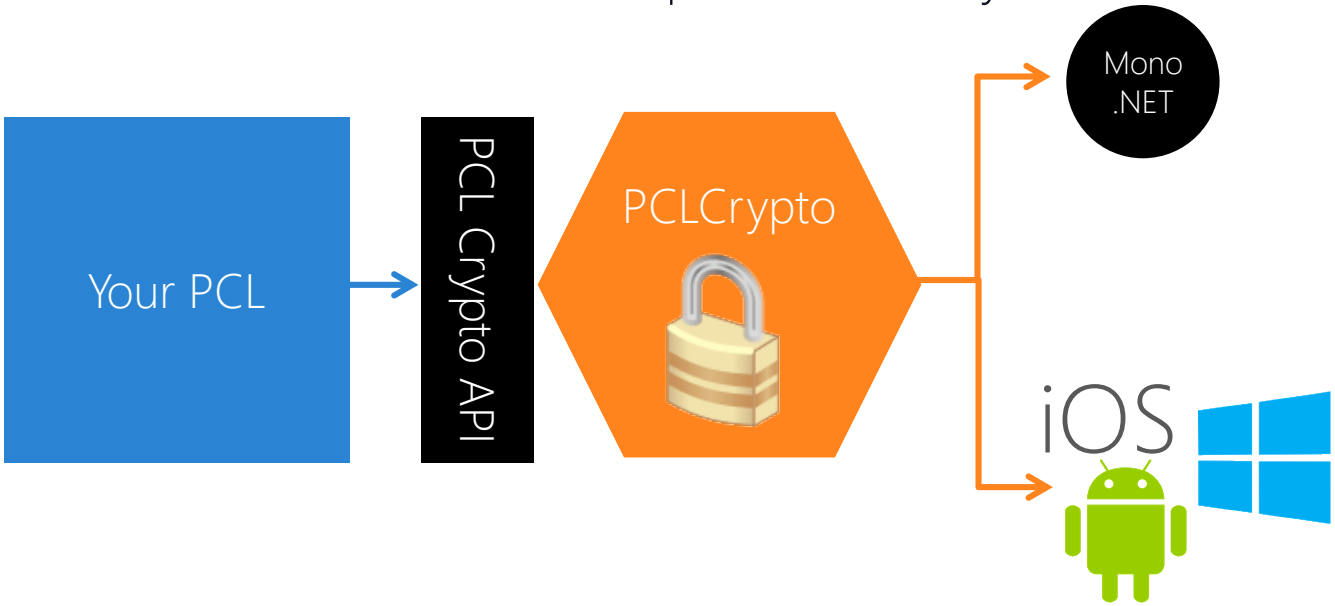
No crypto included

- PCLCrypto never does its own cryptography; it always delegates to the platform or Mono



PCLCrypto API

- PCLCrypto's API is modeled on the WinRT and .NET crypto APIs, this is a practical choice for a C#/.NET developer community



Not a perfect match

- PCLCrypto's API is not exactly the same as WinRT/.NET; strings became enums, **`IBuffer`** became **`byte[]`**, methods were moved, etc.

WinRT

```
class SymmetricKeyAlgorithmProvider
{
  ...CreateSymmetricKey...
  ...BlockLength...
  ...AlgorithmName...

  ...OpenAlgorithm...
}
```

PCLCrypto

```
class SymmetricKeyAlgorithmProvider
{
  ...CreateSymmetricKey...
  ...BlockLength...
  ...AlgorithmName...
}

class SymmetricKeyAlgorithmProviderFactory
{
  ...OpenAlgorithm...
}
```

PCLCrypto source

What is WinRTCrypto?

- **`WinRTCrypto`** class is the entry point to PCLCrypto's WinRT-style API

```
public static class WinRTCrypto
{
  public static IAsymmetricKeyAlgorithmProviderFactory AsymmetricKeyAlgorithmProvider { ... }
  public static ISymmetricKeyAlgorithmProviderFactory  SymmetricKeyAlgorithmProvider  { ... }
  public static IHashAlgorithmProviderFactory          HashAlgorithmProvider          { ... }
  public static IMacAlgorithmProviderFactory           MacAlgorithmProvider           { ... }
  public static IKeyDerivationAlgorithmProviderFactory KeyDerivationAlgorithmProvider { ... }
  public static IKeyDerivationParametersFactory        KeyDerivationParameters        { ... }
  public static ICryptographicEngine                   CryptographicEngine            { ... }
  public static ICryptographicBuffer                   CryptographicBuffer            { ... }
  ...
}
```

PCLCrypto source

The WinRT-style API is PCLCrypto's primary API, it supports encryption, hash, MAC, etc.

What is NetFxCrypto?

- **`NetFxCrypto`** class is the entry point to PCLCrypto's .NET-style API

```
public static class NetFxCrypto
{
  public static IRandomNumberGenerator                RandomNumberGenerator       { ... }
  public static IDeriveBytes                          DeriveBytes                 { ... }
  public static IECDiffieHellmanFactory               ECDiffieHellman             { ... }
  public static IECDiffieHellmanCngPublicKeyFactory ECDiffieHellmanCngPublicKey { ... }
  ...
}
```

PCLCrypto source

PCLCrypto does not attempt to fully cover the .NET crypto API, this class just fills in a few gaps for things not available in WinRT

Algorithm variety

- Each cryptographic operation has many potential algorithms; the enumerations in PCLCrypto list the full variety

```
namespace PCLCrypto
{
  public enum SymmetricAlgorithm
  {
    AesCbc, AesCbcPkcs7, AesCcm, AesEcb, AesEcbPkcs7, AesGcm,
    DesCbc, DesCbcPkcs7, DesEcb, DesEcbPkcs7,
    TripleDesCbc, TripleDesCbcPkcs7, TripleDesEcb, TripleDesEcbPkcs7,
    Rc2Cbc, Rc2CbcPkcs7, Rc2Ecb, Rc2EcbPkcs7, Rc4,
  }
}
```

PCLCrypto source

At first glance, it looks like you have a lot of algorithms available

Algorithm availability

- PCLCrypto cannot provide every algorithm on every platform because Mono and each platform implement only **some of the algorithms**

```
static Platform.SymmetricAlgorithm GetAlgorithm(SymmetricAlgorithm algorithm)
{
  #if SILVERLIGHT || __IOS__
  switch (algorithm)
  {
    case SymmetricAlgorithm.AesCbcPkcs7: return new Platform.AesManaged();
    default: throw new NotSupportedException();
  }
  #else
  ...
```

PCLCrypto throws an exception to indicate an algorithm is not available on the platform running your app

PCLCrypto source

Algorithm selection

- If you want to use the same algorithm everywhere, choose one supported on all of your target platforms

Read the PCLCrypto documentation or source code to find an appropriate cross-platform algorithm

AArnott / **PCLCrypto**

Code | Issues 3 | Pull requests 1 | Wiki | Pulse | Graphs

Algorithms X platforms support

Andrew Arnott edited this page on Jan 2 · 16 revisions

PCLCrypto does not implement cryptography. It merely exposes cryptography that is in the underlying platform in a common PCL-compatible API. The API may allow calling into crypto functions or algorithms that are not available on an individual platform that you run on, in which case you may get a `NotSupportedException` at runtime. The tables below serve as a guide so you can predict what you can expect to work:

** NOTE: This page is under construction and still somewhat incomplete. **

Symmetric encryption algorithms

Algorithm	desktop	win80	wpa81	wp80	monotouch/ios	android
AES	✔	✔	✔	✔	✔	✔
DES	✔	✔	✔	✔		✔
3DES	✔	✔	✔	✔		
RC2	✔	✔	✔	✔		
RC4		✔	✔	✔		✔

Summary

1. Introduce PCLCrypto
2. Survey its capabilities

Hash passwords with PCLCrypto

Xamarin University

Tasks

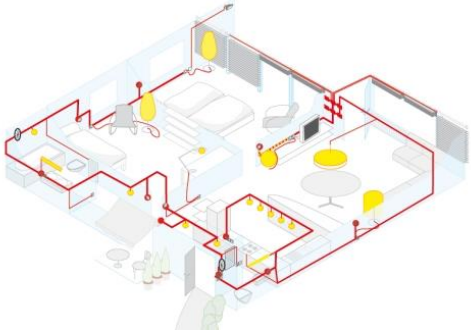
1. Generate a cryptographic hash
2. Hash passwords
3. Validate passwords

Motivation

- App may have locally-stored, confidential information that you want to protect with a pin/password beyond the user's login

Personal or financial data

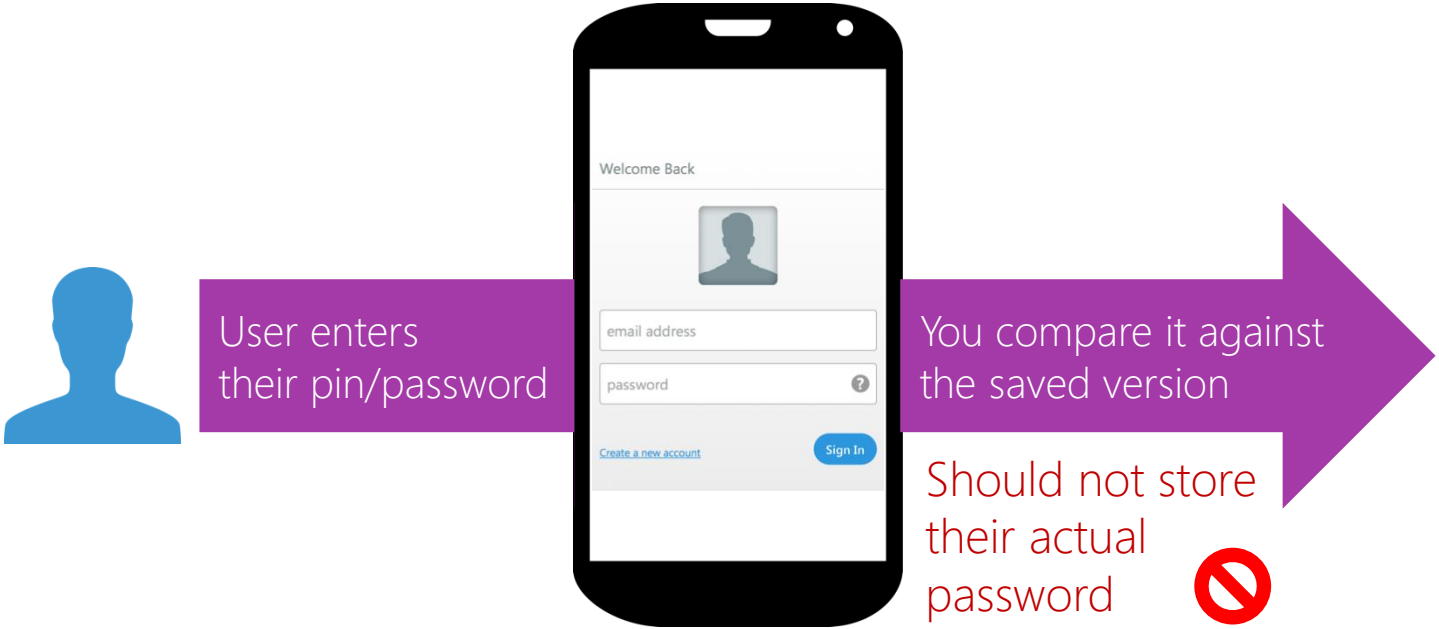
Cached data from web services



Home automation

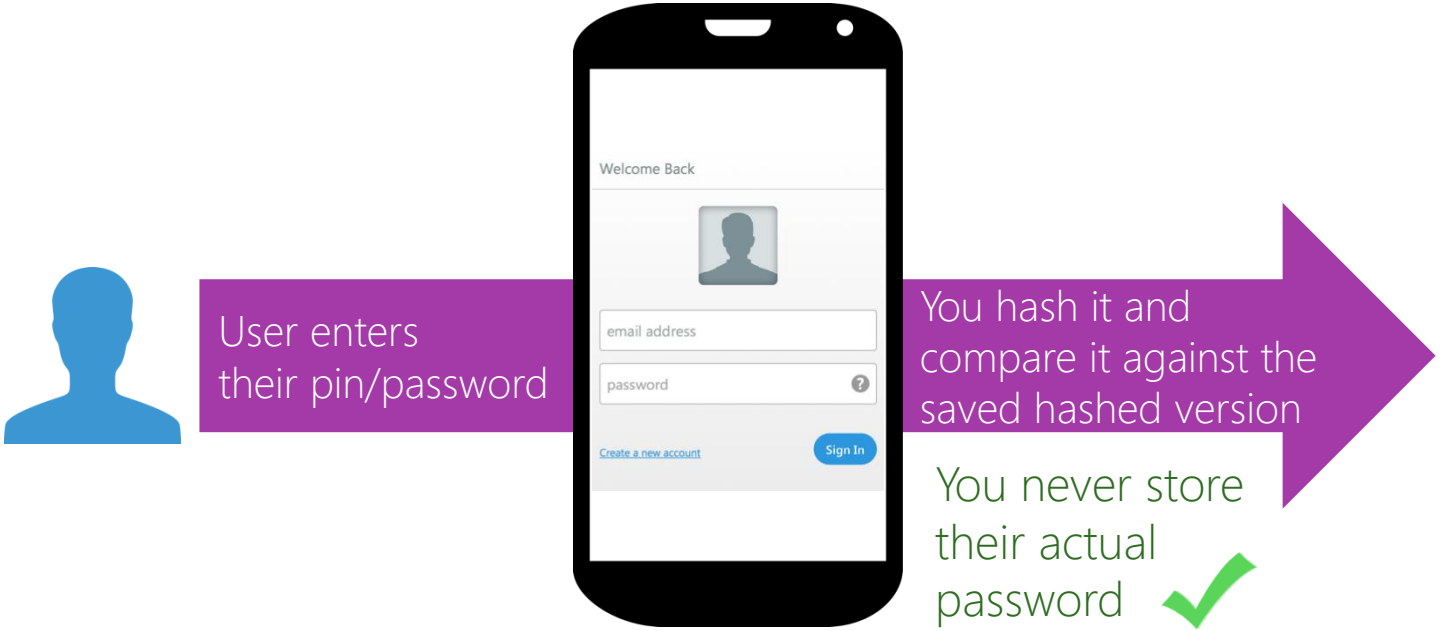
Prefer to avoid pin/password storage

- It is risky to store the user's raw pin/password on the device



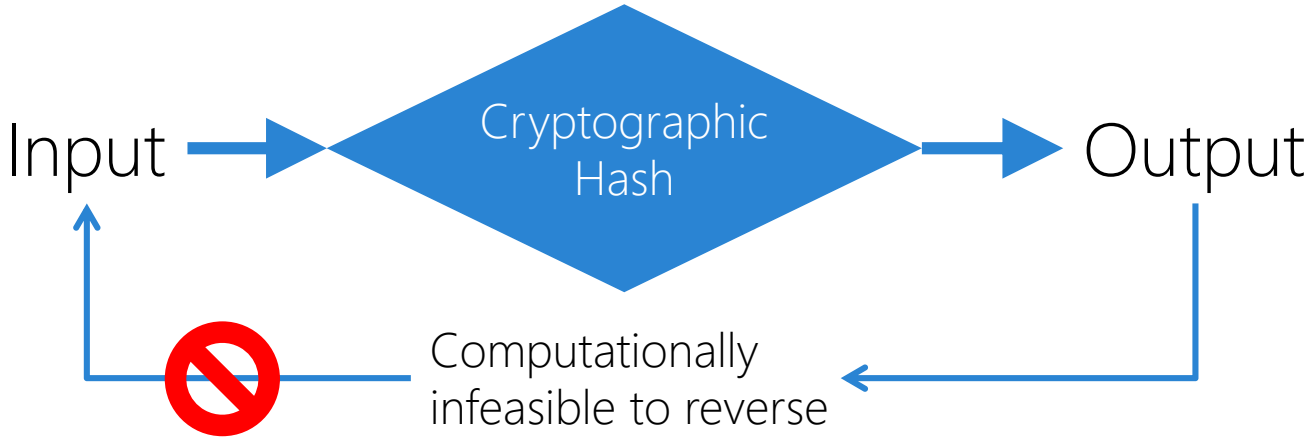
Exact password not needed

- You do not need to store the user's actual password for authentication; you can store a hash of the password and use that instead



What is a cryptographic hash?

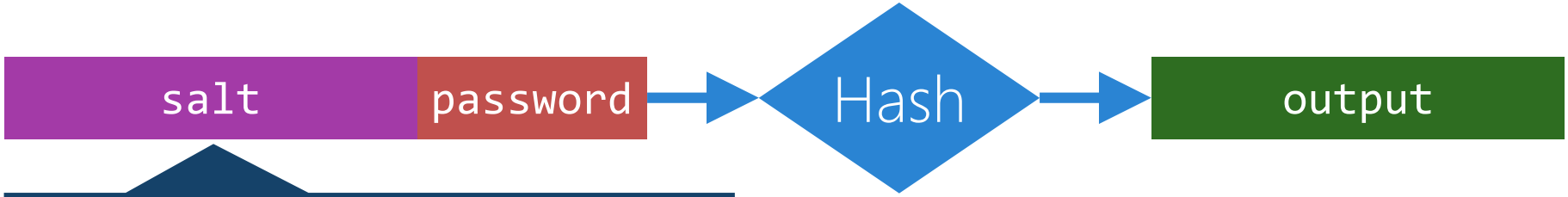
- A *cryptographic hash* is a function that maps inputs to fixed-length outputs with a low probability that two distinct inputs will yield the same output*



A precise definition is more complex than this but is beyond the scope of this course

What is salt?

- Should add a random value to the password before hashing (called *salt*) to ensure two users with the same password have different hashed values



- ✓ Each password needs unique salt
- ✓ Saved for use in verification
- ✓ Need not be secret
- ✓ Can prepend or append to password

How to generate salt?

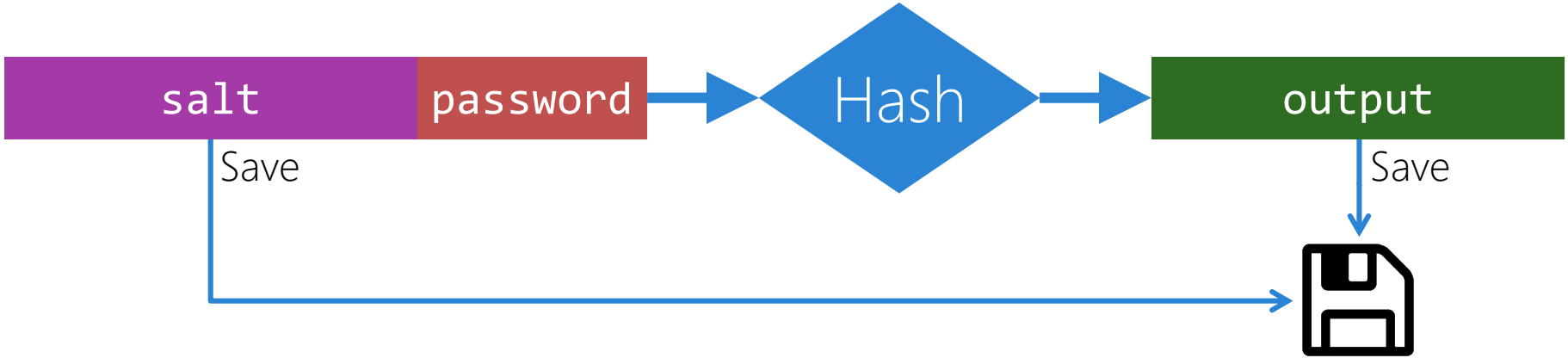
- Use a cryptographically-secure pseudo-random number generator to create salt; PCLCrypto provides an API to generate random bytes

```
byte[] salt = WinRTCrypto.CryptographicBuffer.GenerateRandom(32);
```

Typically use the same length as the hash output (e.g. 32 bytes = 256 bits)

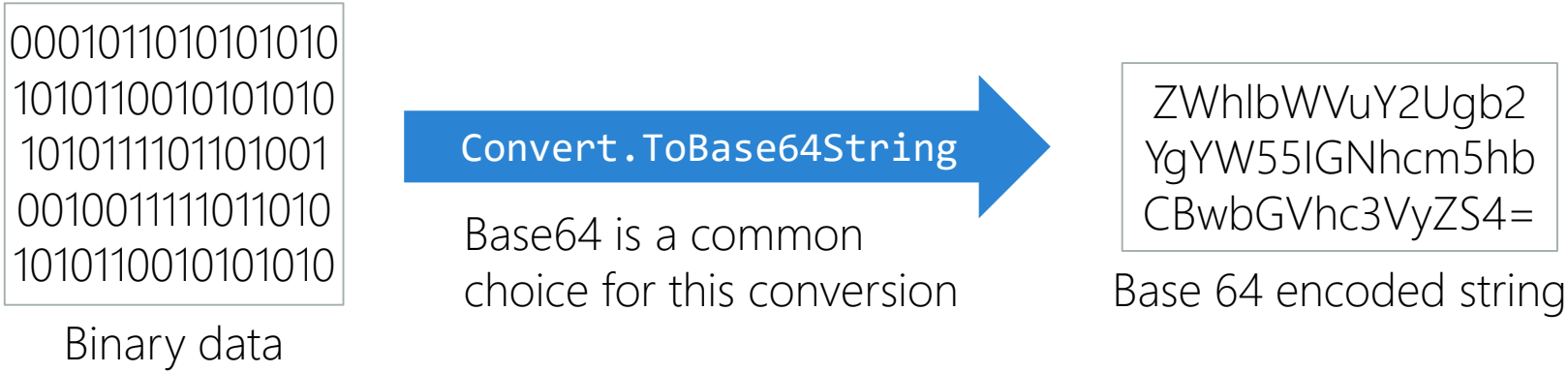
What to persist when hashing?

- You need to store the salt and the hash output for use in verification



Store a byte array as a String

- To store arbitrary binary data in Xamarin.Auth it first needs to be encoded as a string



Hash algorithms

- PCLCrypto lists many cryptographic hash algorithms; however, not every algorithm is available on every platform

You should verify the algorithm is available on your target platforms (i.e. read the PCLCrypto docs and/or source code)

```
public enum HashAlgorithm
{
  Md5,
  Sha1,
  Sha256,
  Sha384,
  Sha512,
}
```

PCLCrypto source

Hash algorithm choice

- Consider using the most secure algorithm available on your target platform(s)

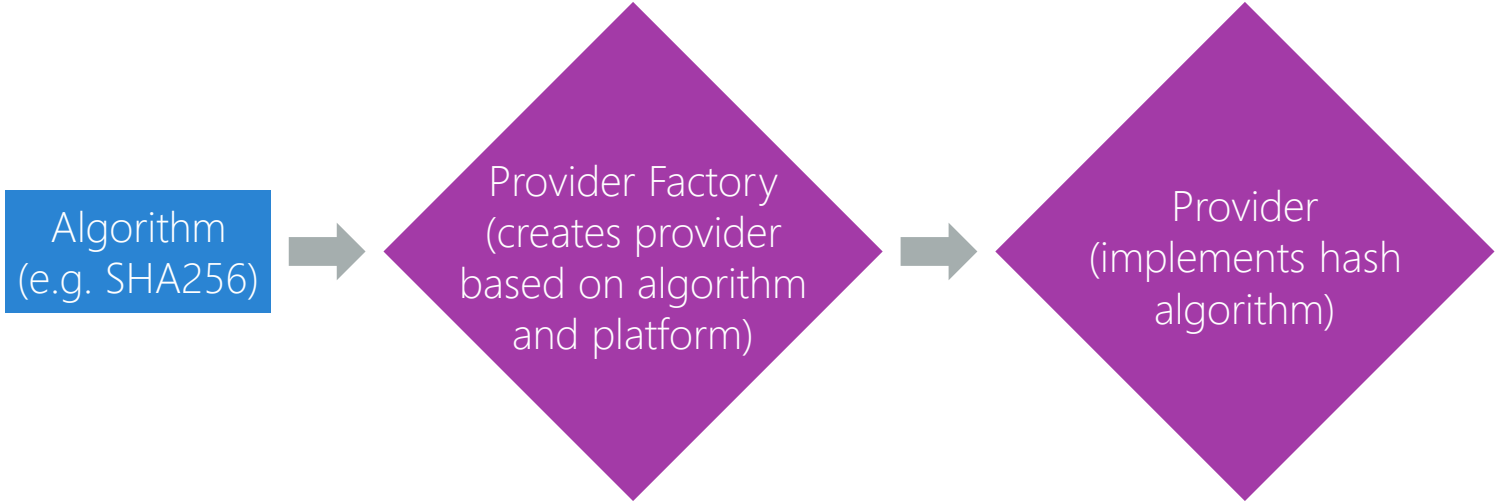
SHA256 or better is generally a reasonable choice

```
public enum HashAlgorithm
{
  Md5,
  Sha1,
  Sha256,
  Sha384,
  Sha512,
}
```

PCLCrypto source

How to generate a hash provider

- PCLCrypto uses a factory pattern to create a hash provider; the provider implements your requested hash algorithm



How to hash a password

- First create an algorithm provider, then use it to hash the salted password

```
static byte[] MyHashFunction(byte[] password, byte[] salt)
{
    byte[] saltedPassword = salt.Concat(password).ToArray();

    var algorithm = HashAlgorithm.Sha256;
    var sha = WinRTCrypto.HashAlgorithmProvider.OpenAlgorithm(algorithm);

    byte[] hash = sha.HashData(saltedPassword);

    return hash;
}
```

Choose algorithm

Hash the salted password

Convert a string to a byte array

- Cryptographic algorithms operate on byte arrays - to convert an arbitrary string to a byte array use the static **`Encoding`** class

"MyPa55w0rdz"

User-entered password string

`Encoding.UTF8.GetBytes`

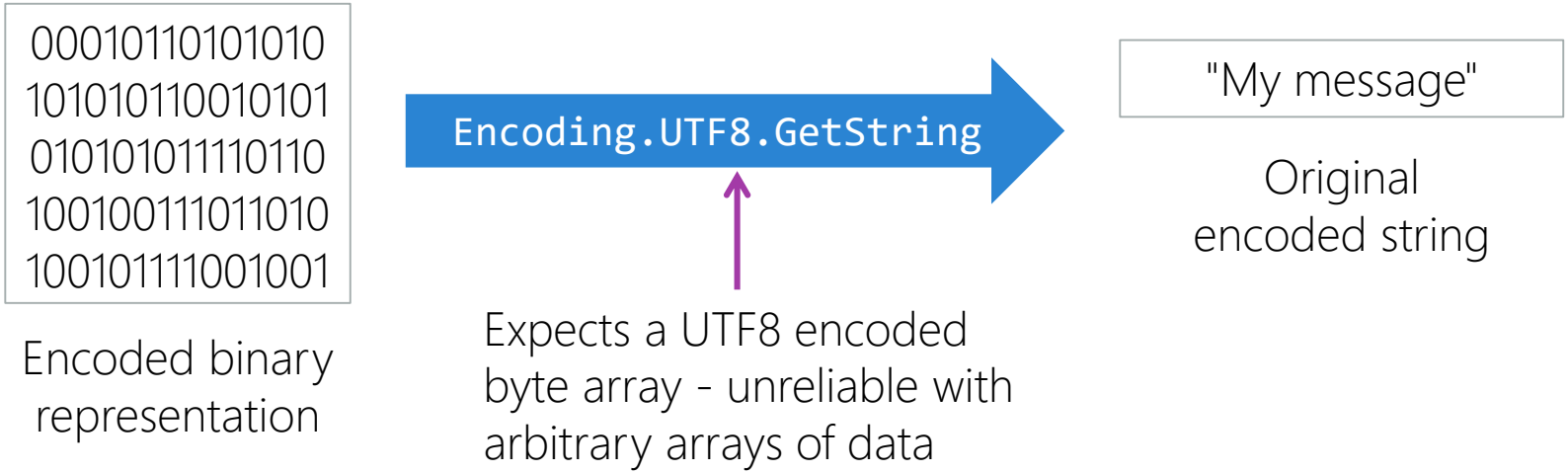
Common to use **`UTF8.GetBytes`** or **`Unicode.GetBytes`**

00010110101010
101010110010101
0101010111110110
1001001110110 10
101011010110101

Binary representation

Convert an encoded byte array to String

- Use an **`Encoding GetString`** method to return an encoded byte array back to its original string



How to validate a password

- Hash the entered password using the same salt, then **compare** it against the saved value

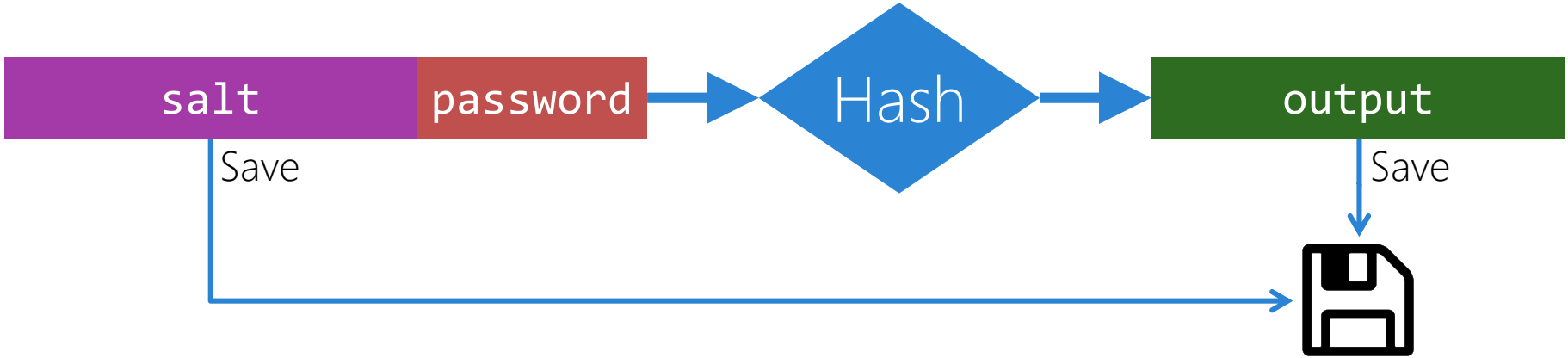
```
static bool MyValidateFunction(byte[] password, byte[] salt, byte[] savedPassword)
{
  var hash = MyHashFunction(password, salt);

  return StructuralComparisons.StructuralEqualityComparer.Equals(hash, savedPassword);
}
```

This equality test is simple but can **leak information** to someone performing a timing attack since it may take different time depending on how many bytes match, see https://crackstation.net/hashing-security.htm

What to persist when hashing?

- You need to store the salt and the hash output for use in verification



Store a byte array as a String

- To store arbitrary binary data in Xamarin.Auth it first needs to be encoded as a string

```
00010110101010
101010110010101
010101011110110
100100111011010
110110011110000
```

Binary data

`Convert.ToBase64String`

Base64 is a common choice for this conversion

```
ZWhlbWVuY2Ugb2
YgYW55IGNhcm5hb
CBwbGVhc3VyZS4=
```

Base 64 encoded string

Base64 is a preferred approach – the **`Encoding GetString`** methods should only be used with binary data returned by the **`Encoding GetBytes`** methods

Individual Exercise

Hash and validate a password

Xamarin University

Summary

1. Generate a cryptographic hash
2. Hash passwords
3. Validate passwords

Encrypt/decrypt data with PCLCrypto

Tasks

1. Create a symmetric key
2. Encrypt data
3. Decrypt data

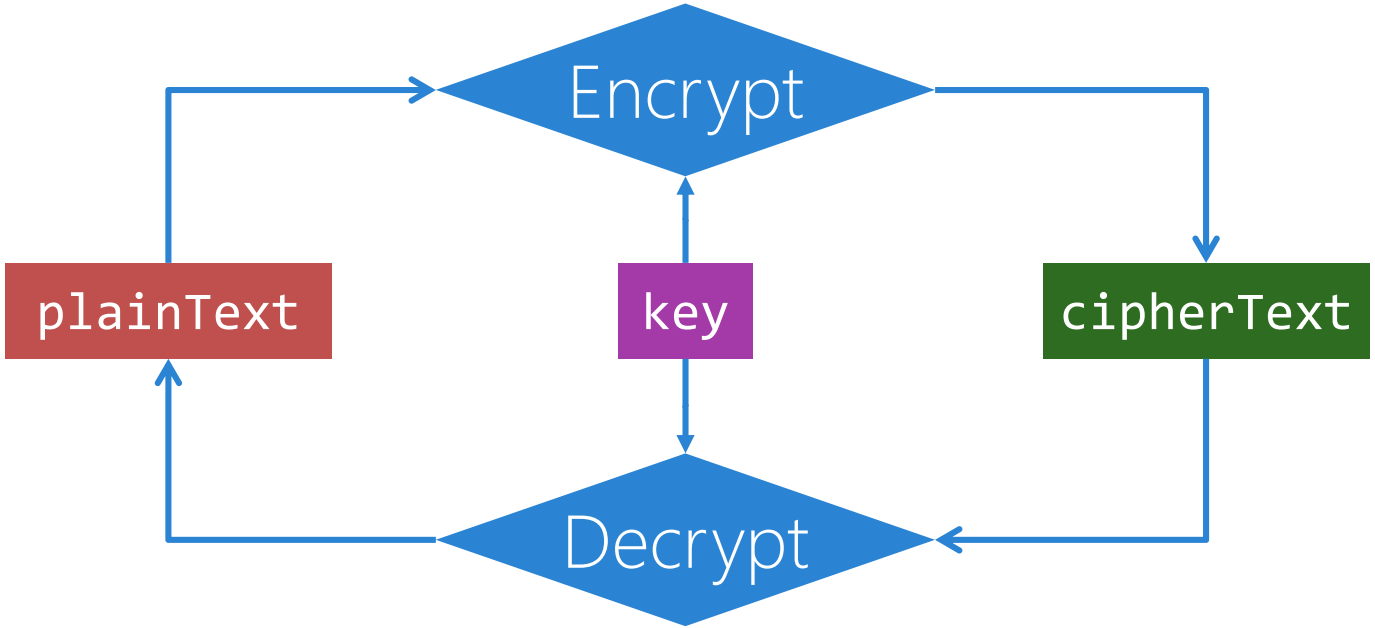
Motivation

- Might need to secure large personal or sensitive data like images, files, etc. on device

Xamarin.Auth stores strings and is not intended for large and/or diverse data types

What is symmetric-key cryptography?

- Symmetric-key cryptography uses a single key to encrypt and decrypt



Symmetric algorithms

- PCLCrypto lists many symmetric algorithms; however, not every algorithm is available on every platform; should verify in docs or source

```
namespace PCLCrypto
{
  public enum SymmetricAlgorithm
  {
    AesCbc, AesCbcPkcs7, AesCcm, AesEcb, AesEcbPkcs7, AesGcm,
    DesCbc, DesCbcPkcs7, DesEcb, DesEcbPkcs7,
    TripleDesCbc, TripleDesCbcPkcs7, TripleDesEcb, TripleDesEcbPkcs7,
    Rc2Cbc, Rc2CbcPkcs7, Rc2Ecb, Rc2EcbPkcs7, Rc4,
  }
}
```

PCLCrypto source

Symmetric algorithm choice

- Consider using the most secure algorithm available on your target platform(s)

```
namespace PCLCrypto
{
  public enum SymmetricAlgorithm
  {
    AesCbc, AesCbcPkcs7, AesCcm, AesEcb, AesEcbPkcs7, AesGcm,
    DesCbc, DesCbcPkcs7, DesEcb, DesEcbPkcs7,
    TripleDesCbc, TripleDesCbcPkcs7, TripleDesEcb, TripleDesEcbPkcs7,
    Rc2Cbc, Rc2CbcPkcs7, Rc2Ecb, Rc2EcbPkcs7, Rc4,
  }
```

Common wisdom says **AES** provides
a good balance of algorithm speed and data security

PCLCrypto source

Symmetric key [key material]

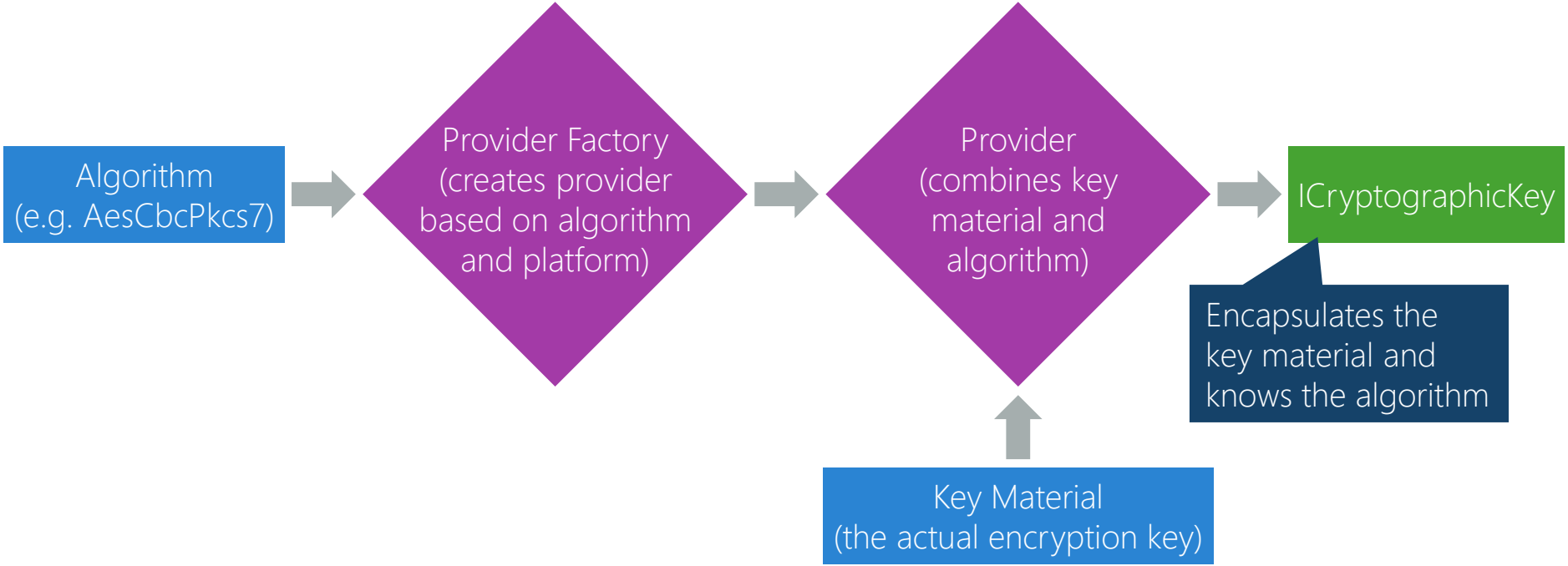
- A symmetric key is generated from *key material*, which can be **random** or derived from a user-entered value such as a password

```
byte[] keyMaterial = WinRTCrypto.CryptographicBuffer.GenerateRandom(32);
```

For AES, the key can be **16**, **24**, or **32** bytes (128, 192, or 256 bits), **prefer 32-byte keys** for best protection

How to generate an encryption key

- PCLCrypto uses a factory/provider pattern to create cryptographic keys



Symmetric key [generation]

- In PCLCrypto, you use a **provider** to generate a key from the key material

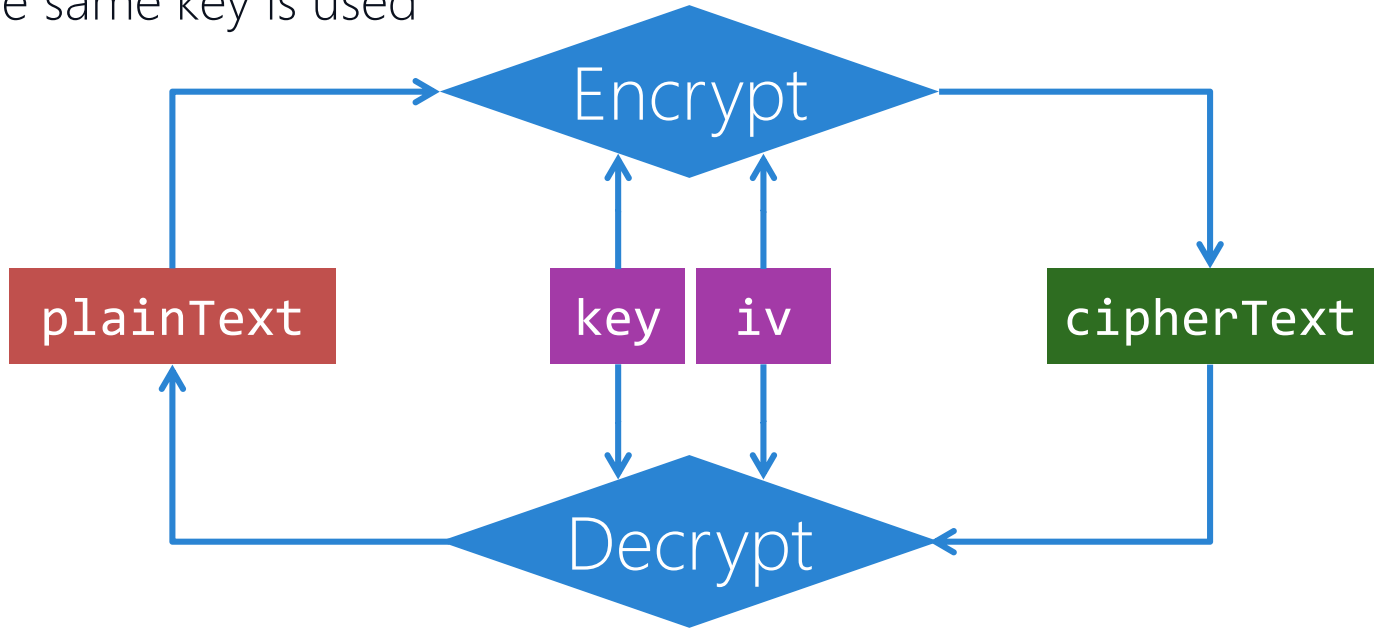
```
byte[] keyMaterial = WinRTCrypto.CryptographicBuffer.GenerateRandom(32);

ISymmetricKeyAlgorithmProvider aes;
aes = WinRTCrypto.SymmetricKeyAlgorithmProvider
                 .OpenAlgorithm(SymmetricAlgorithm.AesCbcPkcs7);

ICryptographicKey key = aes.CreateSymmetricKey(keyMaterial);
```

What is the IV?

- Some algorithms (e.g. AES CBC) use an *initialization vector* (IV) to help ensure that two inputs with the same prefix have different outputs when the same key is used



How to generate an IV

- Typical to use cryptographically random data for the initialization vector, should generate a new IV for every encryption operation when reusing a key

```
byte[] iv = WinRTCrypto.CryptographicBuffer.GenerateRandom(16);
```

Length must match the block size, e.g. 16 bytes (128 bits) for AES CBC

How to do symmetric encryption

- In PCLCrypto, you use **`CryptographicEngine`** with your key and IV to encrypt

```
byte[] MyEncryptFunction(ICryptographicKey key, byte[] iv, byte[] plainText)
{
  return WinRTCrypto.CryptographicEngine.Encrypt(key, plainText, iv);
}
```

Passed **`ICryptographicKey`** encapsulates both the key and the algorithm

How to do symmetric decryption

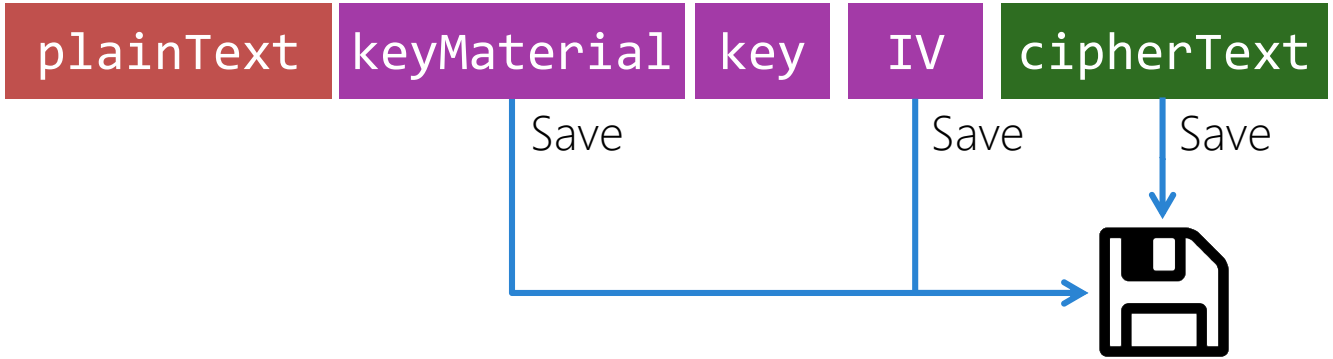
- In PCLCrypto, you use **`CryptographicEngine`** with your key and IV to decrypt

```
byte[] MyDecryptFunction(ICryptographicKey key, byte[] iv, byte[] cipherText)
{
  return WinRTCrypto.CryptographicEngine.Decrypt(key, cipherText, iv);
}
```

The IV must be identical to the one used for encryption

What to persist to decrypt?

- You need to store the key material and the IV so you can decrypt the cipher text



Individual Exercise

Encrypt/decrypt data

Xamarin University

Summary

1. Create a symmetric key
2. Encrypt data
3. Decrypt data

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft