Lab 3a: Private Key Encryption

Details

Aim: To provide a foundation in data encryption.

Activities

If Visual Studio is installed on your machine, download the following solution [1]:

http://www.dcs.napier.ac.uk/~bill/encryption.zip

1. The .NET environment provides a number of cryptography classes. An excellent method is to use a code wrapper, which provides a simple method of accessing these classes [1]. It provides encryption algorithms such as DES, 3DES and BlowFish, and also to hash algorithms such as MD5 and SHA. The following is a simple example using the 3DES algorithm:

```
using System;
using XCrypt;
// Program uses XCrypt library from http://www.codeproject.com/csharp/xcrypt.asp
namespace encryption
{
   class MyEncryption
   {
      static void Main(string[] args)
      {
         XCryptEngine xe = new XCryptEngine();
         xe.InitializeEngine(XCryptEngine.AlgorithmType.TripleDES);
// Other algorithms are:
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.BlowFish);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.Twofish);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.DES);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.MD5);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.RC2);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.Rijndael);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.SHA);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.SHA256);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.SHA384);
//       xe.InitializeEngine(XCryptEngine.AlgorithmType.SHA512);
         xe.Key = "MyKey";

         Console.WriteLine("Enter string to encrypt:");
         string inText = Console.ReadLine();

         string encText = xe.Encrypt(inText);

         string decText = xe.Decrypt(encText);

         Console.WriteLine("Input: {0}\r\nEncr: {1}\r\nDecr: {2}",
                                                 inText,encText,decText);
         Console.ReadLine();
      }
   }
}
```

A sample run shows:

```
Enter string to encrypt:
test
Input: test
Encr: uVZLHJ3Wr8s=
Decr: test
```

By changing the method to SHA gives:

```
Enter string to hash:
test
Input: test
Hash: qUqP5cyxm6YcTAhz05Hph5gvu9M=
```

2. Implement a program for the MD5, SHA, SHA (256-bit), SHA (384-bit), SHA (512-bit) and complete the following table (for the first few characters of the signature):

Text	MD5	SHA	SHA (256)	SHA (384)	SHA (512)
apple					
Apple					
apples					
This is it.					
This is it					

How many characters does each of the types have?

3. Add the following method, and thus convert MD5 and SHA-1 Base-64 hash signatures to hex format:

```
public static string Base64ToHex(string input)
 {
   StringBuilder sb = new StringBuilder();
   byte [] inputBytes = Convert.FromBase64String(input);
   foreach(byte b in inputBytes)
   {
    sb.Append(string.Format("{0:x2}", b));
   }
   return sb.ToString();
 }
```

And change the main program so that it uses the method, such as:

```
xe.InitializeEngine(XCryptEngine.AlgorithmType.MD5);
Console.WriteLine("Enter string to encrypt:");
string inText = Console.ReadLine();

string encText = Base64ToHex(xe.Encrypt(inText));
```

Determine the hash signature for "hello", and check it again a standard MD5 program, such as from: http://pajhome.org.uk/crypt/md5/

4. Prove that the following program can decrypt an encrypted message with the correct encryption key, while an incorrect one does not. Change the program so that the user enters the encryption key, and also the decryption key:

```
xe.Key = "MyKey";

Console.WriteLine("Enter string to encrypt:");
```

```
        string inText = Console.ReadLine();

        string encText = xe.Encrypt(inText);

        xe.Key = "test"; // should not be able to decrypt as the key differs
        try
        {
           string decText = xe.Decrypt(encText);

           Console.WriteLine("Input: {0}\r\nEncr: {1}\r\nDecr: {2}",
                          inText,encText,decText);
        }
        catch { Console.WriteLine("Cannot decrypt");}  ;
        Console.ReadLine();
```

5. The following program uses a single character as an encryption key, and then searches for the encryption key, and displays it. Modify it so that it implements a 2-character encryption key, and then a 3-character one:

```
using System;
using XCrypt;
// Program uses XCrypt library from http://www.codeproject.com/csharp/xcrypt.asp
namespace encryption
{
   class MyEncryption
   {
      static void Main(string[] args)
      {
         XCryptEngine xe = new XCryptEngine();
         xe.InitializeEngine(XCryptEngine.AlgorithmType.TripleDES);
         // Other algorithms are:
         //       xe.InitializeEngine(XCryptEngine.AlgorithmType.BlowFish);
         //       xe.InitializeEngine(XCryptEngine.AlgorithmType.Twofish);
         //       xe.InitializeEngine(XCryptEngine.AlgorithmType.DES);
         //       xe.InitializeEngine(XCryptEngine.AlgorithmType.RC2);
         //       xe.InitializeEngine(XCryptEngine.AlgorithmType.Rijndael);
         xe.Key = "f";

         Console.WriteLine("Enter string to encrypt:");
         string inText = Console.ReadLine();

         string encText = xe.Encrypt(inText);
         for (char ch ='a'; ch<='z'; ch++)
         {
            try
            {
               xe.Key=ch.ToString();
               string decText = xe.Decrypt(encText);
               if (inText==decText) Console.WriteLine("Encryption key found {0}",xe.Key);
            }
            catch {}  ;
         }
         Console.ReadLine();

      }
   }
}
```

An example test run is:

```
Enter string to encrypt:
test
Encryption key found f
```

Note

C# programs can be created without the need for Visual Studio. To compile them, either go to the .NET framework directory, such as:

```
c:\> cd \WINDOWS\Microsoft.NET\Framework\v1.1.4322
```

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322> csc myprog.cs
```

which produces an executable file named **myprog.exe** or create a batch file, with the contents:
c:\windows\microsoft.net\framework\v1.1.4322\csc %1

and call it compile.bat, and then run compile myprog.cs, and it produces the exe.

[1] This code is based around the Xcrypt libraries provided at http://www.codeproject.com/csharp/xcrypt.asp.

Lab 3b: Public-Key Encryption

Details

Aim: To provide a foundation in asymmetric encryption, using the RSA method.

Activities

1. .NET provides us with an excellent foundation in creating applications in which we can view and log events, as well as monitoring for processes. Another key feature is that it supports many encryption and authentication standards. If Visual Studio is installed on your machine, download the following solution:

↳http://www.dcs.napier.ac.uk/~bill/eventLogNew.zip

It has a Windows interface, such as:

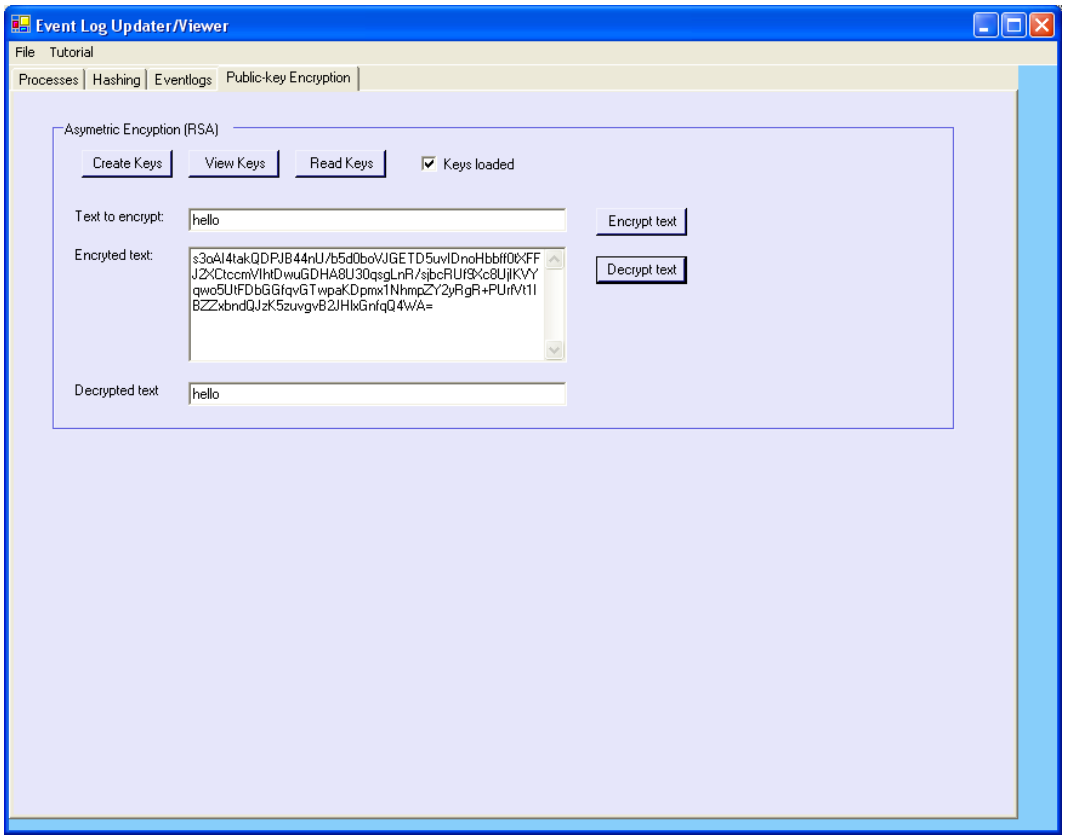


Figure 1: Public-key encryption

2. For the **Create Keys** button add the following code:

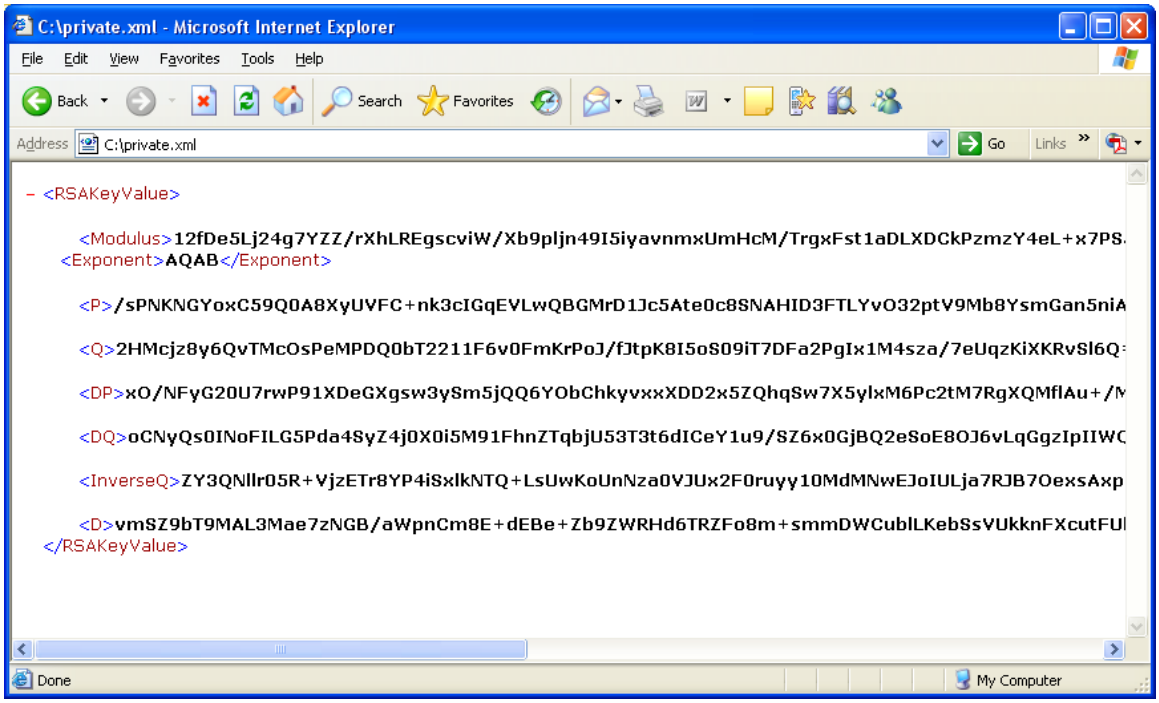
```
System.Security.Cryptography.RSACryptoServiceProvider RSAProvider;
RSAProvider = new System.Security.Cryptography.RSACryptoServiceProvider(1024);
publicAndPrivateKeys = RSAProvider.ToXmlString(true );
justPublicKey = RSAProvider.ToXmlString( false);
StreamWriter fs = new StreamWriter("c:\\public.xml");
fs.Write(justPublicKey);
fs.Close();
fs = new StreamWriter("c:\\private.xml");
fs.Write(publicAndPrivateKeys);
fs.Close();
checkBox2.Checked=true;
```

3. This creates two files on your disk. One contains your public key (**public.xml**) and the other contains both the private key and the public key (**private.xml**). Run the program, and using the View Keys button, view the keys.

What is the format of the keys:

View the files using Internet Explorer to see the XML format.

What are the XML tags in each of the files:



4. From the form, add the following code to the **Read Keys** button:

```
XmlTextReader xtr = new XmlTextReader("c:\\private.xml");
publicAndPrivateKeys=""; // reset keys
justPublicKey="";
while (xtr.Read())
{
   publicAndPrivateKeys += xtr.ReadOuterXml();
}
xtr.Close();
xtr = new XmlTextReader("c:\\public.xml");
while (xtr.Read())
{
   justPublicKey += xtr.ReadOuterXml();
}
xtr.Close();
checkBox2.Checked=true;
```

5. Now add the following code to the **Encrypt text** button:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
string txt=tbTxtEncrypt.Text;
rsa.FromXmlString(justPublicKey);
byte[] plainbytes = System.Text.Encoding.UTF8.GetBytes(txt);
byte[] cipherbytes = rsa.Encrypt(plainbytes,false);
this.tbTxtEncrypted.Text=Convert.ToBase64String(cipherbytes);
```

6. Now add the following code to the **Decrypt text** button:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
string txt=tbTxtEncrypted.Text;
rsa.FromXmlString(publicAndPrivateKeys);
byte[] cipherbytes = Convert.FromBase64String(txt);
byte[] plainbytes = rsa.Decrypt(cipherbytes,false);
System.Text.ASCIIEncoding enc = new System.Text.ASCIIEncoding();
this.tbTxtDecrypt.Text = enc.GetString(plainbytes);
```

7. Now run the program and add some text to the Text to encrypt box, and see if the program encrypts the text, and correctly decrypts it.

Did the program encrypt and decrypt correctly:

8. Now get your give your neighhour your public key file (**public.key**), and get them to encrypt a message. Now take the encrypted message (pass it through copy and paste, and then email the cipertext, or put it on a shared folder), and see if can decrypt it.

Did the program decrypt correctly:

Lab 3c: Hash Methods

You will be assigned a folder in the DFET Cloud. The hashcat version has a time-out, so enter the following command:

date -s "1 OCT 2015 18:00:00"

Demo: http://youtu.be/Xvbk2nSzEPk

1 Hashing

No	Description	Result

1	Using (either on your Windows desktop or on Kali):
	
	`http://asecuritysite.com/encryption/md5`
	
	Match the hash signatures with their words ("Falkirk", "Edinburgh", "Glasgow" and "Stirling").
	
	`03CF54D8CE19777B12732B8C50B3B66F`
	`D586293D554981ED611AB7B01316D2D5`
	`48E935332AADEC763F2C82CDB4601A25`
	`EE19033300A54DF2FA41DB9881B4B723`
		`03CF5:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]?
		
		`D5862:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]?
		
		`48E93:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]?
		
		`EE190:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]?

2	Repeat Part 1, but now use openssl, such as: echo -n 'Falkirk' | openssl md5	`03CF5:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]? `D5862:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]? `48E93:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]? `EE190:` Is it [Falkirk][Edinburgh][Glasgow][Stirling]?

3	Using (either on your Windows desktop or on Kali): `http://asecuritysite.com/encryption/md5` Determine the number of hex characters in the following hash signatures.	MD5 hex chars: SHA-1 hex chars: SHA-256 hex chars: SHA-384 hex chars: SHA-512 hex chars: How does the number of hex characters relate to the length of the hash signature:

3	From your Windows desktop or Kali, for the following /etc/shadow file, determine the matching password:
	
	`bill:$apr1$waZS/8Tm$jDZmiZBct/c2hysERcZ3m1`
	`mike:$apr1$mKfrJquI$Kx0CL9krmqhCu0SHKqp5Q0`
	`fred:$apr1$Jbe/hCIb$/k3A4kjpJyC06BUUaPRKs0`
	`ian:$apr1$0GyPhsLi$jTTzW0HNS4Cl5ZEoyFLjB.`
	`jane: $1$rqOIRBBN$R2pOQH9egTTVN1Nlst2U7.`
		The passwords are password, napier, inkwell and Ankle123.
		[Hint: openssl passwd -apr1 -salt ZaZS/8TF napier]
		
		Bill's password:
		
		Mike's password:
		
		Fred's password:
		
		Ian's password:
		
		Jane's password:

4	From your Windows desktop or Kali, download the following: `http://asecuritysite.com/files02.zip`	Which file(s) have been modified:

	and the files should have the following MD5 signatures:
	
	`MD5(1.txt)= 5d41402abc4b2a76b9719d911017c592`
	`MD5(2.txt)= 69faab6268350295550de7d587bc323d`
	`MD5(3.txt)= fea0f1f6fede90bd0a925b4194deac11`
	`MD5(4.txt)= d89b56f81cd7b82856231e662429bcf2`	

5	From your Windows desktop or Kali, download the following ZIP file:
	
	`http://asecuritysite.com/letters.zip`
	
	View the Postscript files using:
	
	`http://view.samurajdata.se/`
		Outline what the letters contain:
		
		Now determine the MD5 signature for them. What can you observe from the result?

6	Select either Windows or Kali for this part:
	
	On Kali, download the following ZIP file and run the two programs, and run them in a command console:
	
	`http://asecuritysite.com/files01u.zip`
	
	Or on Windows, download the following ZIP file and run the two programs, and run them in a command console:
	
	`http://asecuritysite.com/files01.zip`
		What do the programs do?
		
		Now determine the MD5 signature for them. What can you observe from the result?

2 Hashing Cracking (MD5)

No	Description	Result
1	On Kali, next create a words file (**words**) with the words of "napier", "password" "Ankle123" and "inkwell" Using hashcat crack the following MD5 signatures (hash1):	`232DD...634C` Is it [napier][password][Ankle123][inkwell]? `5F4DC...CF99` Is it [napier][password][Ankle123][inkwell]?

	232DD5D7274E0D662F36C575A3BD634C 5F4DCC3B5AA765D61D8327DEB882CF99 6D5875265D1979BDAD1C8A8F383C5FF5 04013F78ACCFEC9B673005FC6F20698D Command used: `hashcat -m 0 hash1 words`	6D587...5FF5 Is it [napier][password][Ankle123][inkwell]? 04013...698D Is it [napier][password][Ankle123][inkwell]?
2	Using the method used in the first part of this tutorial, find crack the following for names of fruits (the fruits are all in lowercase): FE01D67A002DFA0F3AC084298142ECCD 1F3870BE274F6C49B3E31A0C6728957F 72B302BF297A228A75730123EFEF7C41 8893DC16B1B2534BAB7B03727145A2BB 889560D93572D538078CE1578567B91A	FE01D: 1F387: 72B30: 8893D: 88956:

3 Hashing Cracking (LM Hash/Windows)

All of the passwords in this section are in lowercase.

No	Description	Result
1	On Kali, and using John the Ripper, and using a word list with the names of fruits, crack the following pwdump passwords: fred:500:E79E56A8E5C6F8FEAAD3B435B51404EE:5EBE7DFA074DA8EE8AEF1FAA2BBDE876::: bert:501:10EAF413723CBB15AAD3B435B51404EE:CA8E025E9893E8CE3D2CBF847FC56814:::	Fred: Bert:
2	On Kali, and using John the Ripper, the following pwdump passwords (they are names of major Scottish cities/towns): Admin:500:629E2BA1C0338CE0AAD3B435B51404EE:9408CB400B20ABA3DFEC054D2B6EE5A1::: fred:501:33E58ABB4D723E5EE72C57EF50F76A05:4DFC4E7AA65D71FD4E06D061871C05F2::: bert:502:BC2B6A869601E4D9AAD3B435B51404EE:2D8947D98F0B09A88DC9FCD6E546A711:::	Admin: Fred: Bert:

3	On Kali, and using John the Ripper, crack the following pwdump passwords (they are the names of animals): fred:500:5A8BB08EFF0D416AAAD3B435B51404EE:85A2ED1CA59D0479B1E3406972AB1928::: bert:501:C6E4266FEBEBD6A8AAD3B435B51404EE:0B9957E8BED733E0350C703AC1CDA822::: admin:502::333CB006680FAF0A417EAF50CFAC29C3:D2EDBC29463C40E76297119421D2A707:::	Fred: Bert: Admin:

Repeat all 3.1, 3.2 and 3.3 using **Ophcrack**, and the rainbow table contained on the instance (rainbow_tables_xp_free).

Lab 3d: Digital Certificates

1 Introduction

No	Description	Result
1	From: `http://asecuritysite.com/encryption/digitalcert` Open up certificate 1 and identify the following.	Serial number: Effective date: Name: Issuer: What is CN used for: What is ON used for: What is O used for: What is L used for:
2	Now open-up the ZIP file for the certificate, and view the CER file.	What other information can you gain from the certificate: What is the size of the public key: Which hashing method has been used: Is the certificate trusted on your system: [Yes][No]

3	For Example 2 to Example 6. Complete the following table:	

Cert	**Organisation (Issued to)**	**Date range when valid**	**Size of public key**	**Issuer**	**Root CA**	**Hash method**	**Is it trusted?**
1							
2							
3							
4							
5							
6							

2 PFX files

We have a root certificate authority of My Global Corp, which is based in Washington, US, and the administrator is admin@myglobalcorp.com and we are going to issue a certificate to My Little Corp, which is based in Glasgow, UK, and the administrator is admin@mylittlecorp.com.

No	Description	Result
1	We will now view some PFX certificate files, and which are protected with a password: `http://asecuritysite.com/encryption/digitalcert2`	For Certificate 1, can you open it in the Web browser with an incorrect password: Now enter "apples" as a password, and record some of the key details of the certificate: Now repeat for Certificate 2:
2	Now with the PFX files (contained in the ZIP files from the Web site), try and import them onto your computer. Try to enter an incorrect password first, and observe the message.	Was the import successful? If successful, outline some of the details of the certificates:

3 Creating certificates

Now we will create our own self-signed certificates.

No	Description	Result
1	Create your own certificate from: `http://asecuritysite.com/encryption/createcert` Add in your own details.	View the certificate, and verify some of the details on the certificate. Can you view the DER file?

4 Creating a self signed certificate

You will be assigned a folder in vCentre. Navigate to Production->crypto->netxx and then startup your Kali instance.

We have a root certificate authority of My Global Corp, which is based in Washington, US, and the administrator is admin@myglobalcorp.com and we are going to issue a certificate to My Little Corp, which is based in Glasgow, UK, and the administrator is admin@mylittlecorp.com.

No	Description	Result
1	On Kali, login and get an IP address using: `sudo dhclient eth0`	

2	Create your RSA key pair with:
	
	`openssl genrsa -out ca.key 2048`
	
	Next create a self-signed root CA certificate ca.crt for My Little Corp:
	
	`openssl req -new -x509 -days 1826 -key ca.key -out ca.crt`
		How many years will the certificate be valid for?
		
		Which details have you entered:

3	Next go to Places, and from your Home folder, open up ca.crt and view the details of the certificate.	Which Key Algorithm has been used: Which hashing methods have been used: When does the certificate expire: Who is it verified by: Who has it been issued to:

4	Next we will create a subordinate CA (My Little Corp), and which will be used for the signing of the certificate. First, generate the key:
	
	`openssl genrsa -out ia.key 2048`
	
	Next we will request a certificate for our newly created subordinate CA:
	
	`openssl req -new -key ia.key -out ia.csr`
	
	We can then create a **certificate from the subordinate CA** certificate and **signed by the root CA**.
	
	`openssl x509 -req -days 730 -in ia.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out ia.crt`
		View the newly created certificate.
		
		When does it expire:
		
		Who is the subject of the certificate:
		
		Which is their country:
		
		Who signed the certificate:
		
		Which is their country:
		
		What is the serial number of the certificate:
		
		Check the serial number for the root certificate. What is its serial number:

5	If we want to use this certificate to digitally sign files and verify the signatures, we need to convert it to a PKCS12 file:	Can you view ia.p12 in a text edit?

	`openssl pkcs12 -export -out ia.p12 -inkey ia.key -in ia.crt -chain -CAfile ca.crt`	

6	The crt format is in encoded in binary. If we want to export to a Base64 format, we can use DER:
	
	`openssl x509 -inform pem -outform pem -in ca.crt -out ca.cer`
	
	and for My Little Corp:
	
	`openssl x509 -inform pem -outform pem -in ia.crt -out ia.cer`
		View each of the output files in a text editor (ca.cer and then ia.cer). What can you observe from the format:
		
		Which are the standard headers and footers used:

Lab 3e Hashing Implementation

This lab outlines some implementations for hashing methods.

A LM Hash

The LM Hash is used in Microsoft Windows. For example for LM Hash:

hashme gives: FA-91-C4-FD-28-A2-D2-57-**AA-D3-B4-35-B5-14-04-EE**
network gives: D7-5A-34-5D-5D-20-7A-00-**AA-D3-B4-35-B5-14-04-EE**
napier gives: 12-B9-C5-4F-6F-E0-EC-80-**AA-D3-B4-35-B5-14-04-EE**

Notice that the right-most element of the hash are always the same, if the password is less than eight characters. With more than eight characters we get:

networksims gives: D7-5A-34-5D-5D-20-7A-00-38-32-A0-DB-BA-51-68-07
napier123 gives: 67-82-2A-34-ED-C7-48-92-B7-5E-0C-8D-76-95-4A-50

For "hello" we get:

LM: FD-A9-5F-BE-CA-28-8D-44-AA-D3-B4-35-B5-14-04-EE
NTLM: 06-6D-DF-D4-EF-0E-9C-D7-C2-56-FE-77-19-1E-F4-3C

We can check these with a Python script:

```
import passlib.hash;
string="hello"
print "LM Hash:"+passlib.hash.lmhash.encrypt(string)
print "NT Hash:"+passlib.hash.nthash.encrypt(string)
```

which gives:

```
LM Hash:fda95fbeca288d44aad3b435b51404ee
NT Hash:066ddfd4ef0e9cd7c256fe77191ef43c
```

📖 **Web link (Prime Numbers):** http://asecuritysite.com/encryption/lmhash

No	Description	Result
1	Create a Python script to determine the LM hash and NTLM hash of the following words:	"Napier" "Foxtrot"

B APR1

The Apache-defined APR1 format addresses the problems of brute forcing an MD5 hash, and basically iterates over the hash value 1,000 times. This considerably slows an intruder as they

try to crack the hashed value. The resulting hashed string contains $apr1$" to identify it and uses a 32-bit salt value. We can use both htpassword and Openssl to compute the hashed string (where “bill” is the user and “hello” is the password):

```
# htpasswd -nbm bill hello
bill:$apr1$PkWj6gM4$XGWpADBVPyypjL/cL0XMc1

# openssl passwd -apr1 -salt PkWj6gM4 hello
$apr1$PkWj6gM4$XGWpADBVPyypjL/cL0XMc1
```

We can also create a simple Python program with the passlib library, and add the same salt as the example above:

```
import passlib.hash;

salt="PkWj6gM4"
string="hello"
print "APR1:"+passlib.hash.apr_md5_crypt.encrypt(string, salt=salt)
```

We can created a simple Python program with the passlib library, and add the same salt as the example above:

```
APR1:$apr1$PkWj6gM4$XGWpADBVPyypjL/cL0XMc1
```

Refer to: http://asecuritysite.com/encryption/apr1

No	Description	Result
1	Create a Python script to create the APR1 hash for the following: Prove them against on-line APR1 generator (or from the page given above).	“changeme”: “123456”: “password”

C SHA

While APR1 has a salted value, the SHA has for storing passwords does not have a salted value. It produces a 160-bit signature, thus can contain a larger set of hashed value, but because there is no salt it can be cracked to rainbow tables, and also brute force. The format for the storage of the hashed password on Linux systems is:

```
# htpasswd -nbs bill hello
bill:{SHA}qvTGHdzF6KLavt4PO0gs2a6pQ00=
```

We can also generate salted passwords, and can use the Python script of:

```
import passlib.hash;
salt="8sFt66rZ"
string="hello"
print "SHA1:"+passlib.hash.sha1_crypt.encrypt(string, salt=salt)
print "SHA256:"+passlib.hash.sha256_crypt.encrypt(string, salt=salt)
```

```
print "SHA512:"+passlib.hash.sha512_crypt.encrypt(string, salt=salt)
```

SHA-512 salts start with 6 and are up to 16 chars long.
SHA-256 salts start with 5 and are up to 16 chars long

Which produces:

```
SHA1:$sha1$480000$8sFt66rZ$klAZf7IPWRN1ACGNZIMxxuVaIKRj
SHA256:$5$rounds=535000$8sFt66rZ$.YYuHL27JtcOX8WpjwKf2VM876kLTGZHsHwCBbq9x
TD
SHA512:$6$rounds=656000$8sFt66rZ$aMTKQHl60VXFjiDAsyNFxn4gRezZOZarxHaK.TcpV
YLpMw6MnX0lyPQU06SSVmSdmF/VNbvPkkMpOEONvSd5Q1
```

No	Description	Result
1	Create a Python script to create the SHA hash for the following: Prove them against on-line SHA generator (or from the page given above).	"changeme": "123456": "password"

D PHPass

phpass is used as a hashing method by WordPress and Drupal. It is public domain software and used with PHP applications. The three main methods used are:

- CRYPT_BLOWFISH. This is the most secure method is known as, and related to the bcrypt method. Another method (CRYPT_EXT_DES) uses the DES encryption method.
- CRYPT_EXT_DES. This method uses DES encryption.
- MD5. This is the least preferred method and simply uses an MD5 digest.

The output uses the following to identify the differing types:

- P". Standard.
- H. Phpbb.
- S"; Drupal. SHA-512-like digests.

A sample run with "password" and salt of "ZDzPE45C" for seven rounds gives:

```
$P$5ZDzPE45Ci.QxPaPz.03z6TYbakcSQ0
```

Where it can be see that the salt value is paced after "P5" ("ZDzPE45C"), and after that there are 22 Base-64 characters (giving 128-bit hash signature).

We can check the output against the following Python code:

```
import passlib.hash;
string = "password"
salt="ZDzPE45C"
print passlib.hash.phpass.encrypt(string, salt=salt,rounds=7)
```

which should give: P5**ZDzPE45C**i.QxPaPz.03z6TYbakcSQ0

The code uses 7 rounds - to save processor time - but it can vary between 7 and 30.The number of iterations is 2^rounds. In this case the hash is "i.QxPaPz.03z6TYbakcSQ0" which is a 128-bit hash signature.

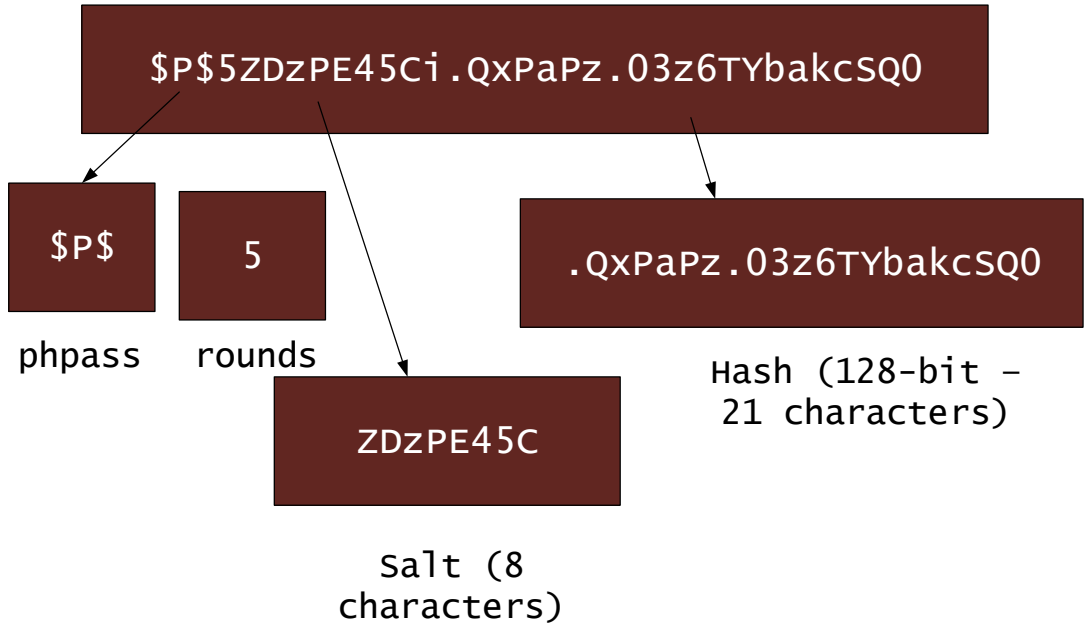


Figure 3.1 phpass hashing

🕮 **Web link (phpass):** http://asecuritysite.com/encryption/phpass

No	Description	Result
1	Create a Python script to create the PHPass hash for the following: Prove them against on-line PHPass generator (or from the page given above). Just note the first five characters of the hashed value.	"changeme": Salt: Hash: "123456": Salt: Hash: "password" Salt: Hash:

E PBKDF2

PBKDF2 (Password-Based Key Derivation Function 2) is defined in RFC 2898 and generates a salted hash. Often this is used to create an encryption key from a defined password, and where it is not possible to reverse the password from the hashed value. It is used in TrueCrypt to generate the key required to read the header information of the encrypted drive, and which stores the encryption keys.

PBKDF2 is used in WPA-2 and TrueCrypt (Figugre 1). Its main focus is to produced a hashed version of a password, and includes a salt value to reduce the opportunity for a rainbow table attack. It generally uses over 1,000 iterations in order to slow down the creation of the hash, so that it can overcome brute force attacks. The generalise format for PBKDF2 is:

DK = PBKDF2(Password, Salt, MInterations, dkLen)

where Password is the pass phrase, Salt is the salt, MInterations is the number of iterations, and dklen is the length of the derived hash.

In WPA-2, the IEEE 802.11i standard defines that the pre-shared key is defined by:

PSK = PBKDF2(PassPhrase, ssid, ssidLength, 4096, 256)

In TrueCrypt we use PBKDF2 to generate the key (with salt) and which will decrypt the header, and reveal the keys which have been used to encrypt the disk (using AES, 3DES or Twofish). We use:

```
byte[] result = passwordDerive.GenerateDerivedKey(16,
                    ASCIIEncoding.UTF8.GetBytes(message), salt, 1000);
```

which has a key length of 16 bytes (128 bits - dklen), uses a salt byte array, and 1000 iterations of the hash (Minterations). The resulting hash value will have 32 hexadecimal characters (16 bytes).

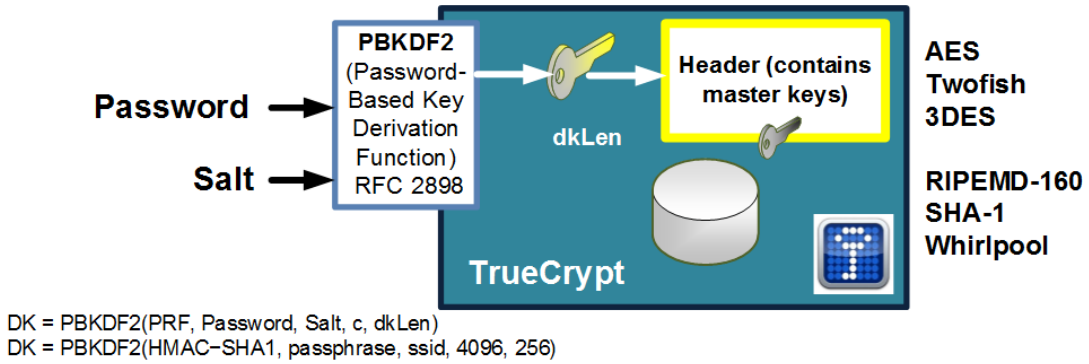


Figure 1 PBKDF2

🕮 **Web link (PBKDF2):** http://www.asecuritysite.com/encryption/PBKDF2

```
import hashlib;
import passlib.hash;
import sys;

salt="ZDzPE45C"
```

```
string="password"

if (len(sys.argv)>1):
       string=sys.argv[1]

if (len(sys.argv)>2):
       salt=sys.argv[2]

print "PBKDF2 (SHA1):"+passlib.hash.pbkdf2_sha1.encrypt(string, salt=salt)
print "PBKDF2 (SHA256):"+passlib.hash.pbkdf2_sha256.encrypt(string, salt=salt)
```

No	Description	Result
1	Create a Python script to create the PBKDF2 hash for the following (uses a salt value of "ZDzPE45C"). You just need to list the first six hex characters of the hashed value.	"changeme": "123456": "password"

F Bcrypt

MD5 and SHA-1 produce a hash signature, but this can be attacked by rainbow tables. Bcrypt (Blowfish Crypt) is a more powerful hash generator for passwords and uses salt to create a non-recurrent hash. It was designed by Niels Provos and David Mazières, and is based on the Blowfish cipher. It is used as the default password hashing method for BSD and other systems.

Overall it uses a 128-bit salt value, which requires 22 Base-64 characters. It can use a number of iterations, which will slow down any brute-force cracking of the hashed value. For example, "Hello" with a salt value of "$2a$06$NkYh0RCM8pNWPaYvRLgN9." gives:

$2a$**06$NkYh0RCM8pNWPaYvRLgN9.**LbJw4gcnWCOQYIom0P08UEZRQQjbfpy

As illustrated in Figure 2, the first part is "$2a$" (or "$2b$"), and then followed by the number of rounds used. In this case is it **6 rounds** which is 2^6 iterations (where each additional round doubles the hash time). The 128-bit (22 character) salt values comes after this, and then finally there is a 184-bit hash code (which is 31 characters).

The slowness of Bcrypt is highlighted with an AWS EC2 server benchmark using hashcat:

- Hash type: MD5 Speed/sec: 380.02M words
- Hash type: SHA1 Speed/sec: 218.86M words
- Hash type: SHA256 Speed/sec: 110.37M words
- Hash type: bcrypt, Blowfish(OpenBSD) Speed/sec: 25.86k words
- Hash type: NTLM. Speed/sec: 370.22M words

You can see that Bcrypt is almost 15,000 times slower than MD5 (380,000,000 words/sec down to only 25,860 words/sec). With John The Ripper:

- md5crypt [MD5 32/64 X2] 318237 c/s real, 8881 c/s virtual
- bcrypt ("$2a$05", 32 iterations) 25488 c/s real, 708 c/s virtual
- LM [DES 128/128 SSE2-16] 88090K c/s real, 2462K c/s virtual

where you can see that Bcrypt over 3,000 times slower than LM hashes. So, although the main hashing methods are fast and efficient, this speed has a down side, in that they can be cracked easier. With Bcrypt the speed of cracking is considerably slowed down, with each iteration doubling the amount of time it takes to crack the hash with brute force. If we add one onto the number of rounds, we double the time taken for the hashing process. So to go from 6 to 16 increase by over 1,000 (2^{10}) and from 6 to 26 increases by over 1 million (2^{20}).

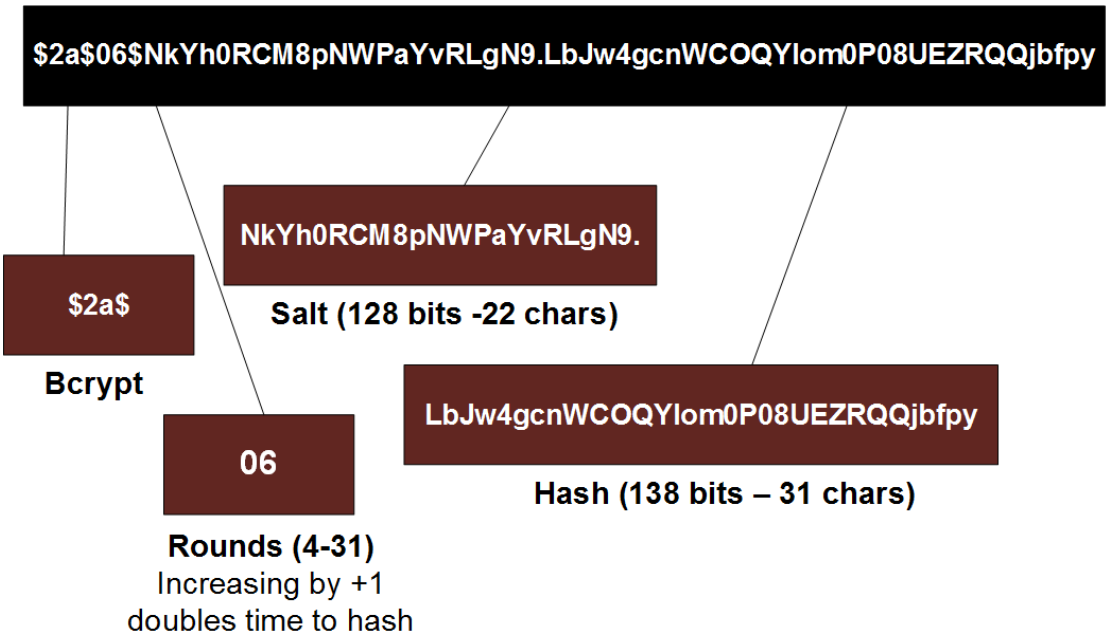


Figure 2 Bcrypt

The following defines a Python script which calculates a whole range of hashes:

```
import hashlib;
import passlib.hash;

salt="ZDzPE45C"
string="password"
salt2="11111111111111111111"

print "General Hashes"
print "MD5:"+hashlib.md5(string).hexdigest()
print "SHA1:"+hashlib.sha1(string).hexdigest()
print "SHA256:"+hashlib.sha256(string).hexdigest()
print "SHA512:"+hashlib.sha512(string).hexdigest()

print "UNIX hashes (with salt)"
print "DES:"+passlib.hash.des_crypt.encrypt(string, salt=salt[:2])
print "MD5:"+passlib.hash.md5_crypt.encrypt(string, salt=salt)
print "Bcrypt:"+passlib.hash.bcrypt.encrypt(string, salt=salt2[:22])
print "Sun MD5:"+passlib.hash.sun_md5_crypt.encrypt(string, salt=salt)
print "SHA1:"+passlib.hash.sha1_crypt.encrypt(string, salt=salt)
print "SHA256:"+passlib.hash.sha256_crypt.encrypt(string, salt=salt)
print "SHA512:"+passlib.hash.sha512_crypt.encrypt(string, salt=salt)
```

No	Description	Result
1	Create the hash for the word “hello” for the different methods (you only have to give the first six hex characters for the hash):	MD5: SHA1: SHA256: SHA512: DES:

	Also note the number hex characters that the hashed value uses:	MD5: Sun MD5: SHA-1: SHA-256: SHA-512:

G A more complete set of hashes

In this final exercise, we will attempt to hash most of the widely used hashing method. For this enter the code of:

```
import hashlib;
import passlib.hash;
import sys;


salt="ZDzPE45C"
string="password"
salt2="11111111111111111111"


if (len(sys.argv)>1):
	string=sys.argv[1]

if (len(sys.argv)>2):
	salt=sys.argv[2]

print "General Hashes"
print "MD5:"+hashlib.md5(string).hexdigest()
print "SHA1:"+hashlib.sha1(string).hexdigest()
print "SHA256:"+hashlib.sha256(string).hexdigest()
print "SHA512:"+hashlib.sha512(string).hexdigest()

print "UNIX hashes (with salt)"
print "DES:"+passlib.hash.des_crypt.encrypt(string, salt=salt[:2])
print "MD5:"+passlib.hash.md5_crypt.encrypt(string, salt=salt)
print "Bcrypt:"+passlib.hash.bcrypt.encrypt(string, salt=salt2[:22])
print "Sun MD5:"+passlib.hash.sun_md5_crypt.encrypt(string, salt=salt)
print "SHA1:"+passlib.hash.sha1_crypt.encrypt(string, salt=salt)
print "SHA256:"+passlib.hash.sha256_crypt.encrypt(string, salt=salt)
print "SHA512:"+passlib.hash.sha512_crypt.encrypt(string, salt=salt)

print "APR1:"+passlib.hash.apr_md5_crypt.encrypt(string, salt=salt)
print "PHPASS:"+passlib.hash.phpass.encrypt(string, salt=salt)
print "PBKDF2 (SHA1):"+passlib.hash.pbkdf2_sha1.encrypt(string, salt=salt)
print "PBKDF2 (SHA256):"+passlib.hash.pbkdf2_sha256.encrypt(string, salt=salt)
#print "PBKDF2 (SHA512):"+passlib.hash.pbkdf2_sha512.encrypt(string, salt=salt)
#print "CTA PBKDF2:"+passlib.hash.cta_pbkdf2_sha1.encrypt(string, salt=salt)
#print "DLITZ PBKDF2:"+passlib.hash.dlitz_pbkdf2_sha1.encrypt(string, salt=salt)

print "MS Windows Hashes"
print "LM Hash:"+passlib.hash.lmhash.encrypt(string)
print "NT Hash:"+passlib.hash.nthash.encrypt(string)
print "MS DCC:"+passlib.hash.msdcc.encrypt(string, salt)
print "MS DCC2:"+passlib.hash.msdcc2.encrypt(string, salt)

#print "LDAP Hashes"
#print "LDAP (MD5):"+passlib.hash.ldap_md5.encrypt(string)
#print "LDAP (MD5 Salted):"+passlib.hash.ldap_salted_md5.encrypt(string, salt=salt)
#print "LDAP (SHA):"+passlib.hash.ldap_sha1.encrypt(string)
#print "LDAP (SHA1 Salted):"+passlib.hash.ldap_salted_sha1.encrypt(string,
salt=salt)
#print "LDAP (DES Crypt):"+passlib.hash.ldap_des_crypt.encrypt(string)
#print "LDAP (BSDI Crypt):"+passlib.hash.ldap_bsdi_crypt.encrypt(string)
#print "LDAP (MD5 Crypt):"+passlib.hash.ldap_md5_crypt.encrypt(string)
#print "LDAP (Bcrypt):"+passlib.hash.ldap_bcrypt.encrypt(string)
#print "LDAP (SHA1):"+passlib.hash.ldap_sha1_crypt.encrypt(string)
#print "LDAP (SHA256):"+passlib.hash.ldap_sha256_crypt.encrypt(string)
#print "LDAP (SHA512):"+passlib.hash.ldap_sha512_crypt.encrypt(string)
```

```
print "LDAP (Hex MD5):"+passlib.hash.ldap_hex_md5.encrypt(string)
print "LDAP (Hex SHA1):"+passlib.hash.ldap_hex_sha1.encrypt(string)
print "LDAP (At Lass):"+passlib.hash.atlassian_pbkdf2_sha1.encrypt(string)
print "LDAP (FSHP):"+passlib.hash.fshp.encrypt(string)

print "Database Hashes"
print "MS SQL 2000:"+passlib.hash.mssql2000.encrypt(string)
print "MS SQL 2000:"+passlib.hash.mssql2005.encrypt(string)
print "MS SQL 2000:"+passlib.hash.mysql323.encrypt(string)
print "MySQL:"+passlib.hash.mysql41.encrypt(string)
print "Postgres (MD5):"+passlib.hash.postgres_md5.encrypt(string, user=salt)
print "Oracle 10:"+passlib.hash.oracle10.encrypt(string, user=salt)
print "Oracle 11:"+passlib.hash.oracle11.encrypt(string)

print "Other Known Hashes"
print "Cisco PIX:"+passlib.hash.cisco_pix.encrypt(string, user=salt)
print "Cisco Type 7:"+passlib.hash.cisco_type7.encrypt(string)
print "Dyango DES:"+passlib.hash.django_des_crypt.encrypt(string, salt=salt)
print "Dyango MD5:"+passlib.hash.django_salted_md5.encrypt(string, salt=salt[:2])
print "Dyango SHA1:"+passlib.hash.django_salted_sha1.encrypt(string, salt=salt)
print "Dyango Bcrypt:"+passlib.hash.django_bcrypt.encrypt(string, salt=salt2[:22])
print "Dyango PBKDF2 SHA1:"+passlib.hash.django_pbkdf2_sha1.encrypt(string,
salt=salt)
print "Dyango PBKDF2 SHA1:"+passlib.hash.django_pbkdf2_sha256.encrypt(string,
salt=salt)
```

No	Description	Result
1	In the code, what does the modifier of "`[:22]`" do? In running the methods, which of them take the longest time to compute? Of the methods used, outline how you would identify some of the methods. For APR1 has an identifier of $apr1$.	

For the following identify the hash methods used:

- 5f4dcc3b5aa765d61d8327deb882cf99
- 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
- $apr1$ZDzPE45C$y372GZYCbB1WYtOkbm4/u.
- PHZDzPE45Ch4tvOeT9mhtu3i2G/JybR1
- b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86
- 1ZDzPE45C$EEQHJaCXI6yInV3FnskmF1
- $2a$12$1111111111111111111111uAQxS9vJNRtBb6zeFDV6k7tyB0DZJF0a

H Reflective statements

1. Why might increasing the number of iterations be a better method of protecting a hashed password than using a salted version?

2. Why might the methods BCrypt, Phpass and PBFDK2 be preferred for storing passwords than MD5, SHA?

Possible references:

- https://www.linkedin.com/pulse/swimming-molases-days-salting-password-may-passing-william-buchanan
- https://www.linkedin.com/pulse/when-slow-good-great-slowcoach-bcrypt-william-buchanan

Lab 3f: DLP - Disk/Email Encryption

Outline demo: https://youtu.be/dob5emKJa3o

1 PGP Encryption

An import element in data loss prevention is encrypted emails. In this part of the lab we will use an open source standard: PGP.

On your main desktop, install GPG from:

http://gpg4win.org/download.html

No	Description	Result

1	From the console on your **Windows desktop** (C:\Program Files (x86)\GNU\GnuPG), create a key pair with (RSA and 2,048 bit keys):
	
	`gpg --gen-key`
	
	Now export your public key using the form of:
	
	`gpg --export -a "Your name" > mypub.key`
	
	Now export your private key using the form of:
	
	`gpg --export-secret-key -a "Your name" > mypriv.key`
		How is the randomness generated?
		
		Outline the contents of your key file:

2	Now send your lab partner your public key in the contents of an email, and ask them to import it onto their key ring: `gpg --import` *`theirpubickey.key`*	Which keys are stored on your key ring and what details do they have:

	Now list your keys with: `gpg --list-keys`	
3	Create a text file, and save it. Next encrypt the file with their public key: `gpg -e -a -u "Your Name" -r "Your Lab Partner Name" hello.txt`	What does the –a option do: What does the –r option do: What does the –u option do: Which file does it produce and outline the format of its contents:
4	Send your encrypted file in an email to your lab partner, and get one back from them. Now create a file (such as myfile.asc) and decrypt the email using the public key received from them with: `gpg –d myfile.asc > myfile.txt`	Can you decrypt the message:

5	Next using this public key file, send Bill (w.buchanan@napier.ac.uk) a question (http://asecuritysite.com/public.txt):
	
	`-----BEGIN PGP PUBLIC KEY BLOCK-----`
	`Version: BCPG C# v1.6.1.0`
	
	`mQENBFTvF+gBCACkpcMPybSe1NTE1hDg86gPcQqoT8kD9oS/ankGwbB4R5zT+3Ny`
	`MZWZwT43lL99R7sfkluglwVkqko74Lemy9pBF/rbWeWev6mCR3z1V3yTTv3zP1V5`
		Did you receive a reply:

	tLcz3K65flRHPQU/FzxqH1T4kaH6dDiL/UuKKcyYMxXNnqERitJPU7ZJVhqeM3gi 4cG4znKY5fw8bdSpNC//pgkDzEaWYJFdyq/KqCwRK5r/Egj7FVHaLGC37lDgZKR5 dBoIVaOTfXykJLe3Vc3dIv9LU58U3YHqsc/w6X4E5R/eEnp0IwKyb7oXdrFOM5ud DSoJ7aT24IqZW678vNtufGdr4OD+BF5r2UZpABEBAAG0GHcuYnVjaGFuYW5hQG5h cGllci5hYy51a4kBHAQQAQIABgUCVO8X6AAKCRBOV4Uk9xMsXJgNB/4jfAnXLHjZ +I4z3Hhqn9UMokU6Q4cQtrGX0OhelymKZTMXNoSKhT5fB9GBlIIbwMkZHxcUNmUB PuAwq+RAhFqtRkcH3x1a5eNBhEvcfi9hS2ls43gfsrXjMzekY6dyzD/ePM7HvihJ vrsQNZNI7ZIaP5viCZFgQqmwYQA1LCrEy/xpSXBNrqrOwuti+2+xeZsswitYLAzA ryDMgCG9GPuSfkmvatYJJrl5QAhj1p0FKERhL1/h3bh18i8Llh1K9tEBxIJf4ZIy ivV1bX5G36jciOrKCLi7/m6xhHh86brRQA++qwUdXU/3MMqvRwuinSO9NYeVCf6Y V66cJqTgdR1F =uiw7 -----END PGP PUBLIC KEY BLOCK-----	
6	Next send your public key to Bill (w.buchanan@napier.ac.uk).	

2 TrueCrypt

Now go to the DFET Napier Cloud, and you should have a Kali instance.

No	Description	Result
1	Go to your Kali instance. Now **Create a new volume** and use an **encrypted file container** (use tc_*yourname*) with a Standard TrueCrypt volume. When you get to the Encryption Options, run the tests and outline the results:	CPU (Mean) AES: AES-Twofish: AES-Two-Seperent Serpent -AES Serpent: Serpent-Twofish-AES Twofish: Twofish-Serpent: Which is the fastest:

		Which is the slowest:
2	Select AES and RIPMD-160 and create a 100MB file. Finally select your password and use FAT for the file system.	What does the random pool generation do, and what does it use to generate the random key?
3	Now mount the file as a drive.	Can you view the drive on the file viewer and from the console? [Yes][No]
4	Create some files your TrueCrypt drive and save them. Next dismount your drive, and copy the file to the provided USB stick. Give the USB stick to your neighbour, and see if they can view the file contents.	Without giving them the password, can they read the file? With the password, can they read the files?

3 TrueCrypt Volumes

The following files have the passwords of "Ankle123", "foxtrot", "napier123", "password" or "napier". Determine the properties of the files defined in the table:

File	**Size**	**Encryption type**	**Key size**	**Files/folders on disk**	**Hidden partition (y/n)**	**Hash method**
http://asecuritysite.com/tctest01.zip						
http://asecuritysite.com/tctest02.zip						
http://asecuritysite.com/tctest03.zip						

Now with **truecrack** see if you can determine the password on the volumes. Which TrueCrypt volumes can truecrack?

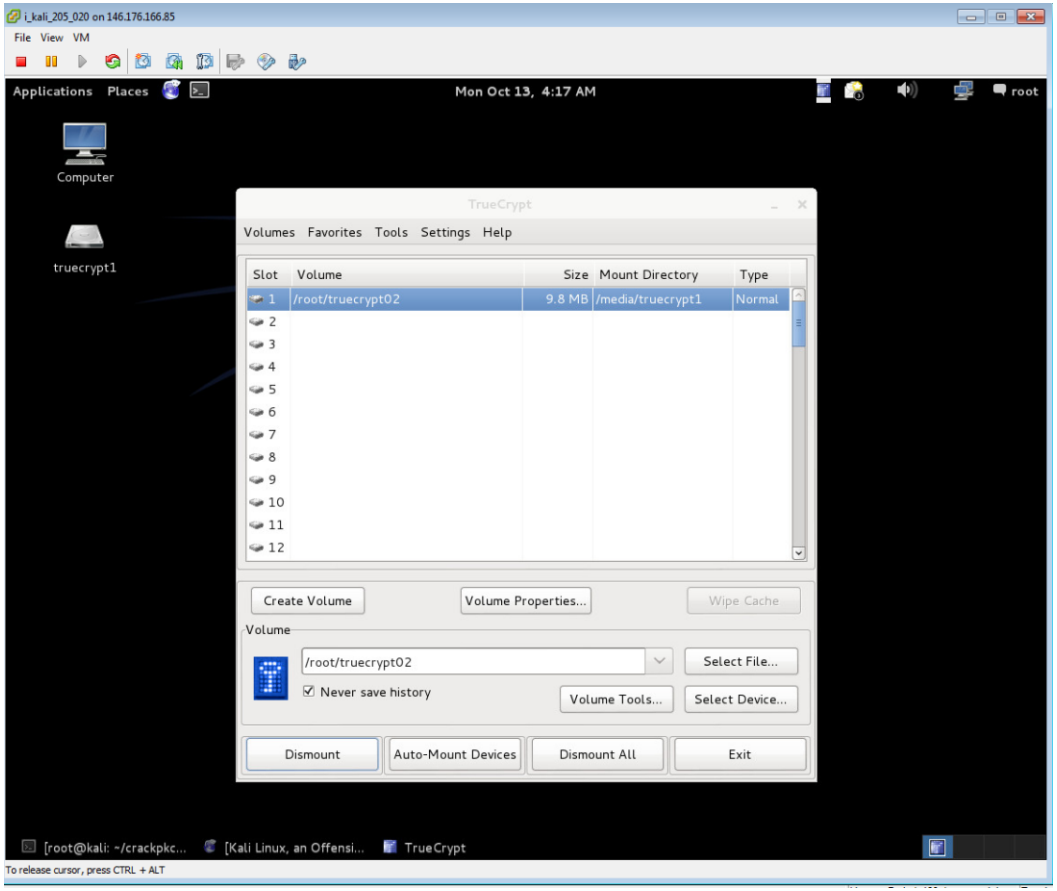


Figure 1: Kali mount

4 EFS

On your desktop, undertake the following.

No	Description	Result
1	Go to your Windows 7 instance. Now create a folder named: **My_Enc_*yourname*** Add some files to the folder, and then right click on the folder and encrypt it.	How does the name of the folder change when it is encrypted?
2	Now use: Cipher /u	Which files are encrypted on your drive:
3	Using: Cipher /c *filename*	Which encryption type and key size has been used for the file encryption?
4	Make sure you can view your files. Now export your certificates with: Cipher /r:*filename*	Which are the names of the files created? View the CER and PFX file. What is the difference between the two files?
5	Go to Control Panel -> Internet Options Then click on the Content tab and select the Certificates button. Now view your EFS certificate.	Outline some of the details of the EFS certificate. When does it expire? What type of encryption does it use? What is the length of the encryption key?

		Who has signed it?
6	Now delete the EFS certificate from the store and reboot your instance.	After reboot, can you access your files? [Yes][No]
7	Now import the PFX certificate that you created.	Can you access your files? [Yes][No]

5 EFS (with USB)

Undertake the following, but this time mount a USB stick, and encrypt on the USB device. First delete your existing EFS certificate.

No	Description	Result
1	Go to your Windows 7 instance. Now create a folder named: **My_Enc_*yourname*** Add some files to the folder, and then right click on the folder and encrypt it.	How does the name of the folder change when it is encrypted?
2	Now use: Cipher /u	Which files are encrypted on your USB disk:
3	Using: Cipher /c *filename*	Which encryption type and key size has been used for the file encryption?

4	Make sure you can view your files. Now export your certificates with: Cipher /r:*filename*	Which are the names of the files created? View the CER and PFX file. What is the difference between the two files?
5	Go to Control Panel -> Internet Options Then click on the Content tab and select the Certificates button. Now view your EFS certificate.	Outline some of the details of the EFS certificate. When does it expire? What type of encryption does it use? What is the length of the encryption key? Who has signed it?
6	Now delete the EFS certificate from the store and reboot your instance.	After reboot, can you access your files? [Yes][No]
7	Now import the PFX certificate that you created.	Can you access your files? [Yes][No]
8	Now dismount your drive, and give the USB stick to your neighbour. 1. Ask them to access the files on the USB disk without importing the certificate. 2. Ask them to access the files on the USB disk after importing the certificate.	Can they access your files before certificate import? [Yes][No] Can they access your files after certificate import? [Yes][No]

6 Cracking digital certificates and file types

Undertake the following.

No	Description	Result
1	Run Networksims. Now run Toolkit client (Figure 2). Goto the Encryption tab and select Digital Certificate from the left-hand menu. Next click on the Dictionary Search button, and load each of the following files (remember to extract to PFX): http://asecuritysite.com/log/fred.zip http://asecuritysite.com/log/sample01.zip	What are the passwords for the PFX files?

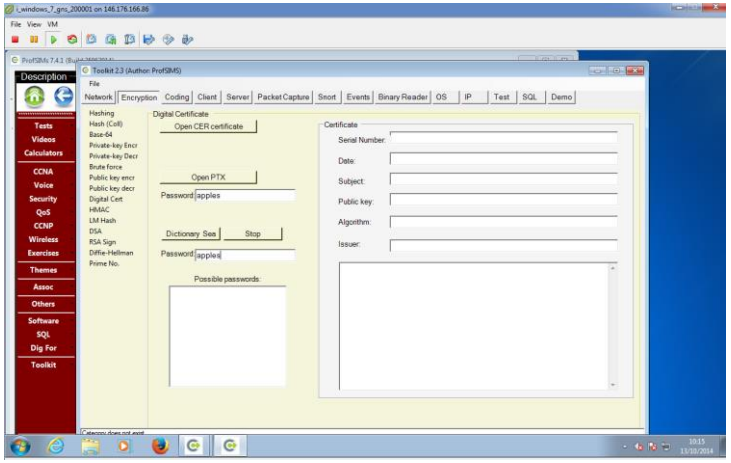


Figure 2: Dictionary search