Crypto implementation crash course

Tonimir Kišasondi, mag.inf, EUCIP

$ whois tkisason

- Junior researcher @ FOI
- Crypto, network security, penetration testing, linux, open systems

- Topic of this talk:
- How to avoid common crypto implementation fails

Let's start off on the right foot...

- One nice snippet of code i found on Twitter

- http://windows.xiangnen.com/how-to-encrypt-string-in-c-windows-application/

- C#

How many problems do you see here?

```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.Mode = CipherMode.ECB;
tripleDES.Padding = PaddingMode.PKCS7;
ICryptoTransform cTransform = tripleDES.CreateEncryptor();
byte[] resultArray = cTransform.TransformFinalBlock(inputArra…
0, inputArray.Length);
tripleDES.Clear();
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

How many problems do you see here?

```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.Mode = CipherMode.ECB;
tripleDES.Padding = PaddingMode.PKCS7;
ICryptoTransform cTransform = tripleDES.CreateEncryptor();
byte[] resultArray = cTransform.TransformFinalBlock(inputArra…
0, inputArray.Length);
tripleDES.Clear();
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

How many problems do you see here?

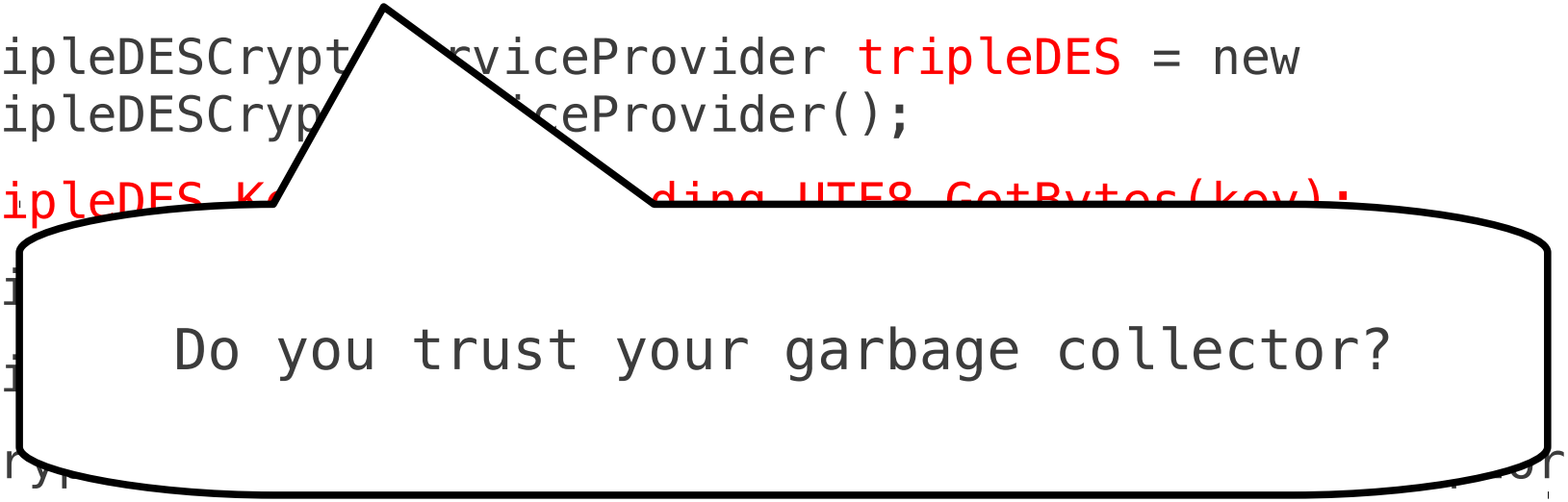
```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.Mode = CipherMode.ECB;
tripleDES.Padding = PaddingMode.PKCS7;
ICryptoTransform cTransform = tripleDES.CreateEncryptor();
byte[] resultArray = cTransform.TransformFinalBlock(inputArray,
0, inputArray.Length);
tripleDES.Clear();
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

UTF8 for "PASSWORD"

0050 0041 0053 0053 0057 004F 0052 0044

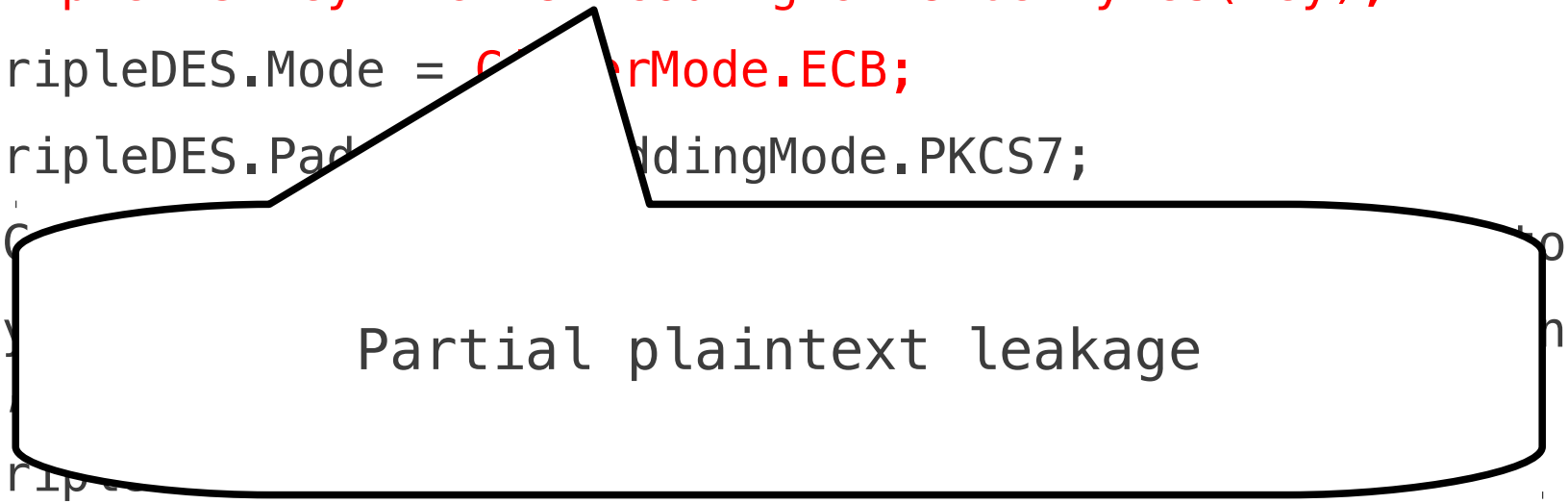
How many problems do you see here?

```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCrypt   rviceProvider tripleDES = new
TripleDESCryp      ceProvider();
tripleDES.K                    ding.UTF8.GetBytes(key);
tri
tri
ICry                                                    r();
byte[] resultArray = cTransform.TransformFinalBlock(inputArra…
0, inputArray.Length);
tripleDES.Clear();
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```



How many problems do you see here?

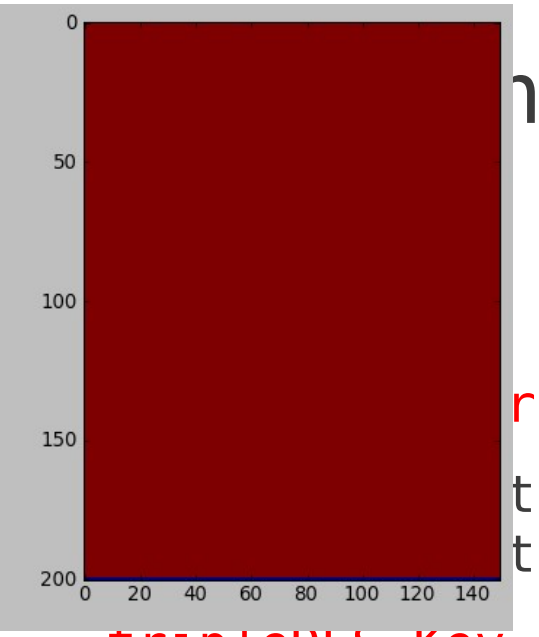
```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.Mode = CipherMode.ECB;
tripleDES.Padding = PaddingMode.PKCS7;
ICryptoTransform cTransform = tripleDES.CreateEncryptor();
byte[] resultArray = cTransform.TransformFinalBlock(inputArray, 0,
inputArray.Length);
tripleDES.Clear();
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

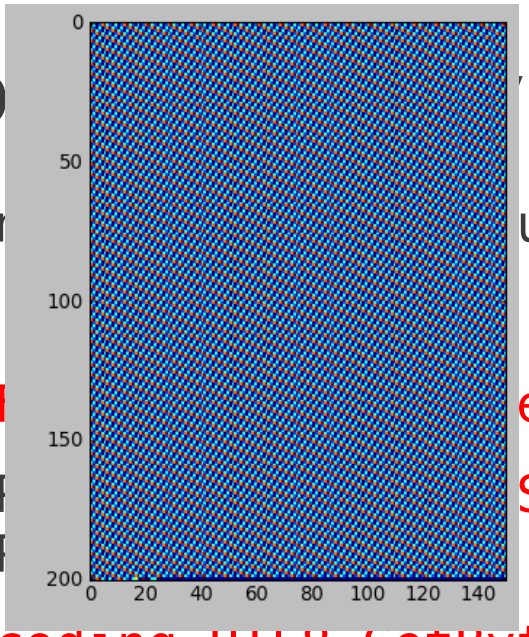


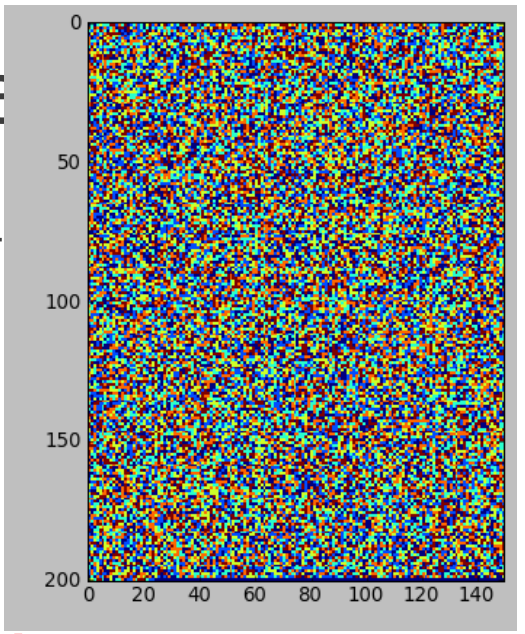
How many problems do you see here?

```
public static string Encrypt(string input, string key)
{
byte[] inputArray = UTF8Encoding.UTF8.GetBytes(input);
TripleDESCryptoServiceProvider tripleDES = new
TripleDESCryptoServiceProvider();
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.      = CipherMode.ECB;
tripleDES.              PaddingMode.PKCS7;
IC                                                    ();
by                                                    tArra…
0, Use current standards, don't rely on old ones
trip
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

3DES -> Fail








```
ny pro                    ou se
    string Er                 ut, stri

rray = UTF                    etBytes(
toServiceP                    S = new
toServiceP
tripleDES.Key = UTF8Encoding.UTF8.GetBytes(key);
tripleDES.Mode = CipherMode.ECB;
tripleDES.Padding             ngMode.PKCS7;
IC                                                  ();
by                                                  tArra…
0,
t
return Convert.ToBase64String(resultArray, 0,
resultArray.Length);
}
```

ECB is extremely bad
More possible plaintext leakage
Plaintext structure leakage
Replay attacks! Since we can :)

Whats the lesson behind all this?

- You can easily fail while implementing crypto
- Some implementations are less vulnerable, some are catastrophically vulnerable
- Do you feel lucky?

Lesson 1:

- "If you write the letters AES in your source code, you're doing it wrong"
  - Thomas Ptacek

- If you don't know what to do:
  - GPG for static content
  - TLS 1.2 for traffic
  - Be smart with key management
- Use cryptographic libraries:
  - Google Keyczar
  - BouncyCastle

Lesson 1:

- OpenSSL is really cool!
  - Do you know what bf-ofb means?
  - OpenSSL is here to ensure that you have sane and secure primitives
  - OpenSSL doesn't protect you against yourself
- Again, think as an attacker...
- You won't break AES, but you might just find a design vulnerability

Lesson 2: Primitives

- Should i even say that you should not implement your own crypto primitives?
- Using a CRC checksum with "secret" factors is not a digital signature
  - Collect enough messages and you can calculate the factors.
- Use standardised algorithms and libraries
  - No, the code you found on planetsourcecode or pastebin is not OK.
  - Wikipedia HMAC, SHA1 debacle

Lesson 3: EAI

- Know the difference:
  - Encryption
  - Authentication
  - Identification
- Don't encrypt when you want to authenticate, don't authenticate when you want to identify.
- You should: Identify, Authenticate and then Decrypt

Lesson 4: Keys

- Keys should be random:
  - I should not note this holds for passwords
- 10 character password for 128bit key:
  - 80 bit key, ASCII isn't true random
    - Even less, you can drop below 50 bit strength!
- Use PBKDF2 (Password based key derivation functions)

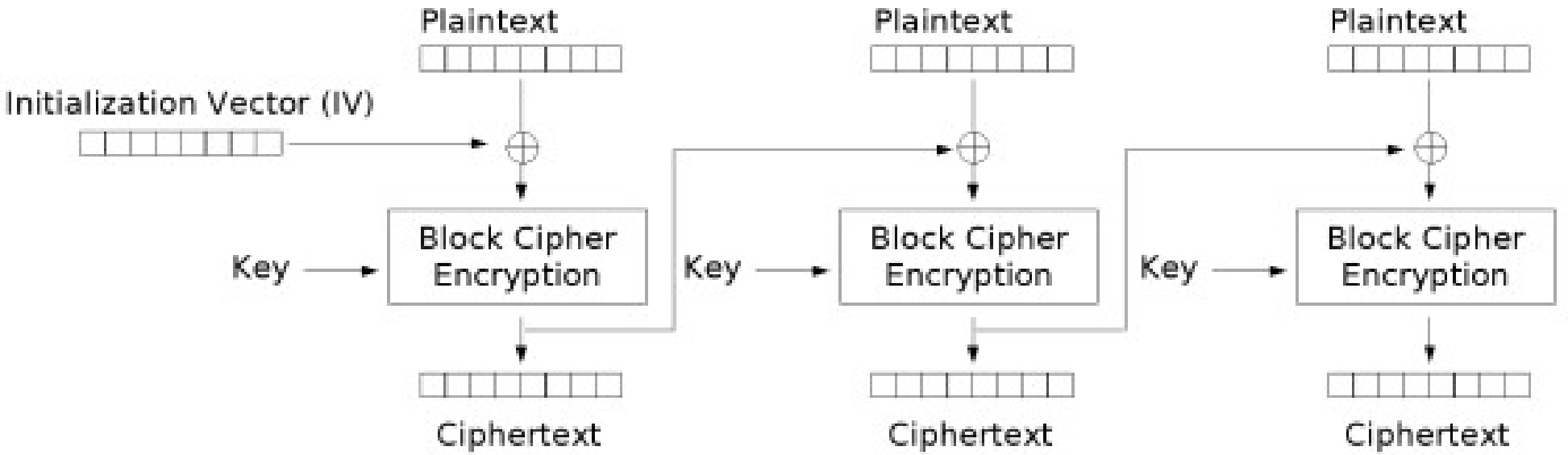
Lesson 5: Passwords

- If you store passwords:
  - Salt them
  - PBKDF2
  - Don't use weak hashes (MD5,SHA1)
  - Use strong hashes (SHA256)
  - Scrypt is great
- Zeroize keys and passwords when you don't need them!
- WPA does it right almost right...

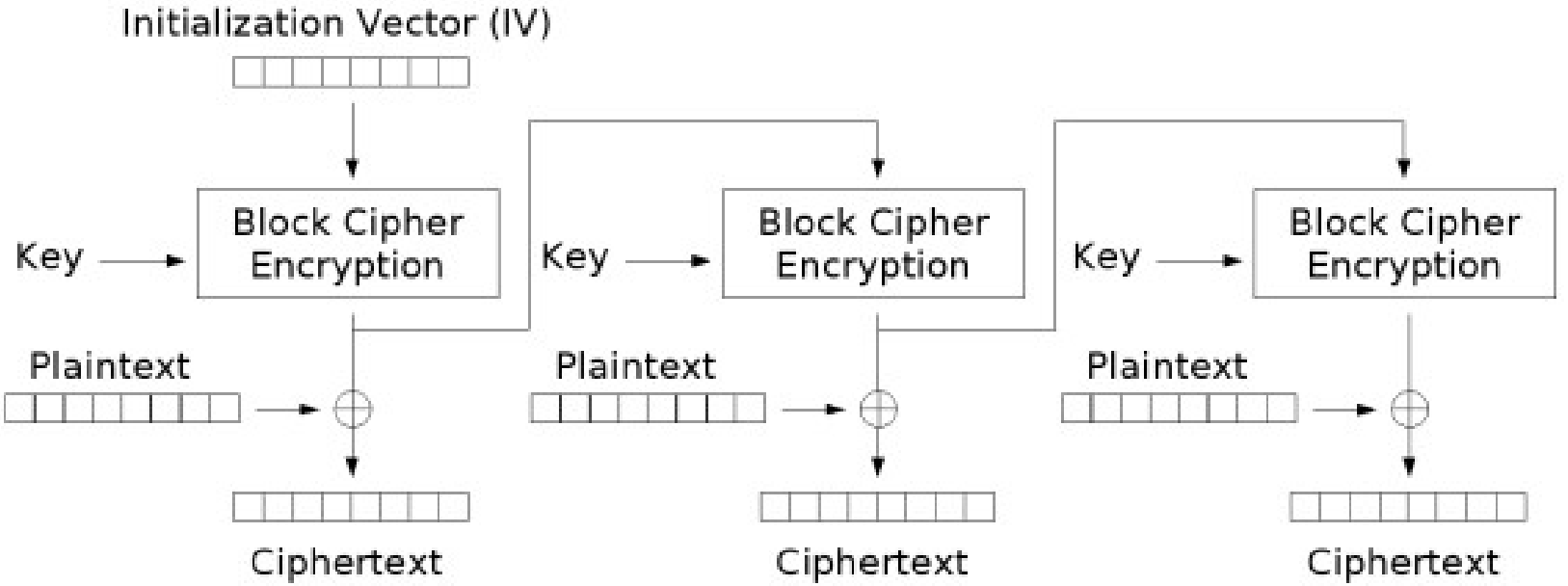
l0-structure	Occ	%	C%
a	13733915	41,85563%	41,86%
a0	9105511	27,75006%	69,61%
0	5227172	15,93039%	85,54%
0a	834095	2,54200%	88,08%
a0a	538871	1,64227%	89,72%
A	490396	1,49454%	91,21%
A0	441712	1,34617%	92,56%
Aa0	309847	0,94429%	93,51%
Aa	214160	0,65268%	94,16%
a#	212808	0,64856%	94,81%
a#a	164871	0,50246%	95,31%
a#0	161504	0,49220%	95,80%
a0a0	140140	0,42709%	96,23%

Lesson 6: Simmetric encryption

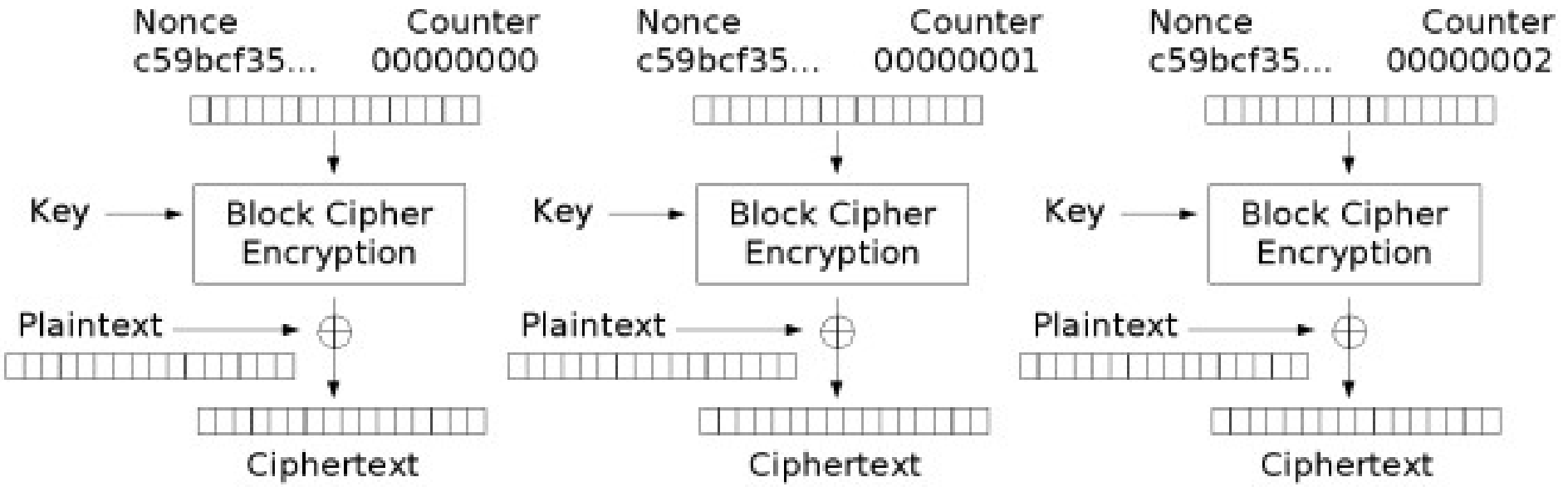
- Avoid 3DES, DES..
- Use AES256 or AES128
  - 256 is better in case someone nicks 128 bits
- Use known modes of operation
  - Raw mode / ECB is bad / Concat Raw is pure stupid
  - CBC is ok if you don't reuse IVs, and use random ones
  - GCM/XTR is ok for FDE
  - Avoid OFB/CFB, if you don't know proper stream implementations / syncronizations
  - OCB is patented, watch out for patents
  - CTR is OK
  - Authenticate messages before decryption
  - Don't reuse IV's, counters or outputs!



Cipher Block Chaining (CBC) mode encryption



Output Feedback (OFB) mode encryption



Counter (CTR) mode encryption

Lesson 7: Message authentication

- Symmetric signatures
- HMAC -> Hashed Message Authentication Code
- Helps you authenticate messages and prevents replay/forgery attacks
- Did i say? Authenticate before decryption?
- MACs are relatively less used, they should be mandatory.

Lesson 8:

- Avoid DSA
- Use at least RSA2048
  - Pad messages
- Avoid ECC if you don't want to get sued / be careful about your curves
  - Curve25519?
- If you can, use different keys for signing and encryption

Breaking SSL and TLS

- BEAST (Browser exploit against SSL/TLS)
- Malicious javascript inside SSL/TLS session == Profit!
- Blockwise chosen plaintext attack
  - Regarded some time ago as "theoretical at best"
- Allegedly breaks everyting except TLS 1.1 and 1.2
  - Guess who uses TLS 1.1 and 1.2
    - Thats right, nobody...

Conclusion

- Think critically about crypto implementations
- Use Google, ask a cryptographer
- Read a book, or 10
- "The devil is in the details"

Questions?

THANK YOU!