School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP3911 Secure Computing

4: Symmetric Ciphers

Nick Efford

https://comp3911.info

Slides are customised for XJCO3911 by Turgay Celik (Turgay.Celik@swjtu.edu.cn)

Objectives

UNIVERSITY OF LEEDS

- To appreciate how **symmetric ciphers** help us achieve the goal of **data confidentiality**
- To explore the resistance of ciphers to brute-force attack
- To consider the role played by cipher modes
- To learn the broad principles of how the Advanced Encryption Standard (AES) works
- To see how AES fits into cryptographic schemes such as AES-CBC-HMAC and AES-GCM

Basic Concepts

UNIVERSITY OF LEEDS

- Message to be encrypted is the **plaintext**
- Symmetric cipher uses a **secret key**, shared with recipient, to transform the plaintext
- Transformation performs multiple transpositions and substitutions of bits, to generate **ciphertext**
- Decryption inverts the transformation, **using same key**
- Transformation should be easily reversible if we know the key, nearly impossible to reverse if we don’t

Types of Symmetric Cipher

UNIVERSITY OF LEEDS

Block cipher

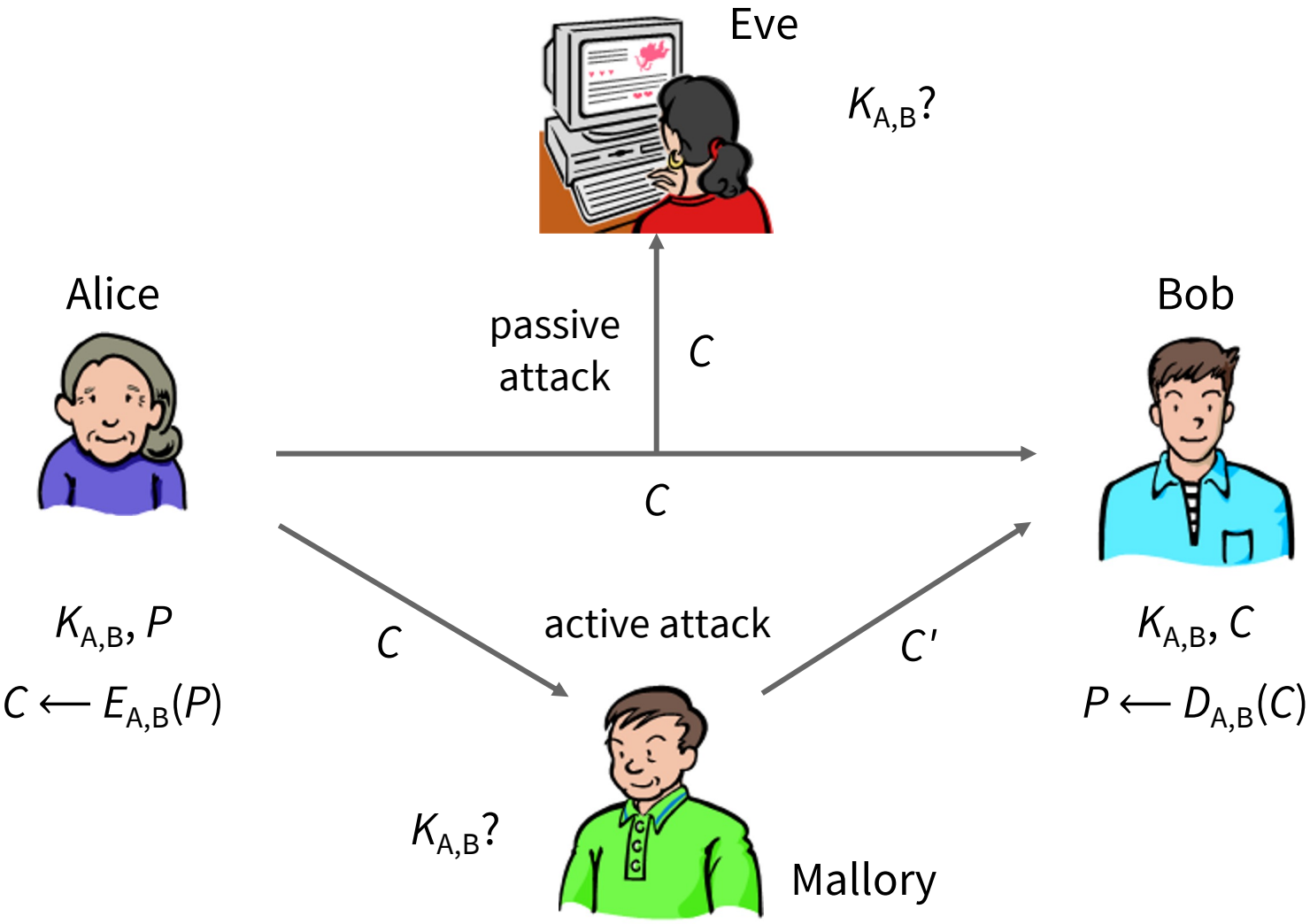
- Processes plaintext in fixed-size blocks
- Adds padding so plaintext size is a multiple of block size
- Examples: DES, RC2, Blowfish, Camellia, **AES**

Stream cipher

- Processes plaintext continuously
- Uses key to generate a stream of pseudorandom data, then XORs ($\oplus$) this with plaintext
- Examples: RC4, Rabbit, SNOW 3G

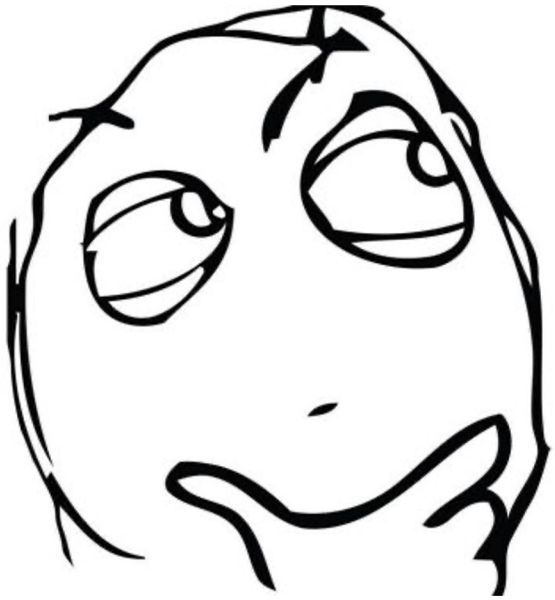
Scenarios

UNIVERSITY OF LEEDS



‘Security Through Obscurity’

UNIVERSITY OF LEEDS



Which is better?

A well-established cipher whose inner workings are widely known?

Or a custom cipher invented for a specific application by some software developers, whose details are kept hidden?

Kerckhoffs' Principle

UNIVERSITY OF LEEDS

"A cryptosystem should be secure even if everything about the system, **except the key**, is public knowledge"

See also **Shannon's maxim**:
"assume that the enemy knows the system"

Requirements For Key

UNIVERSITY OF LEEDS

- Large enough that it cannot be found by **brute-forcing** (i.e., trying all keys until decryption succeeds)
- Random enough that it can’t be predicted
- Stored and shared securely

Brute-Forcing of Keys

UNIVERSITY OF LEEDS

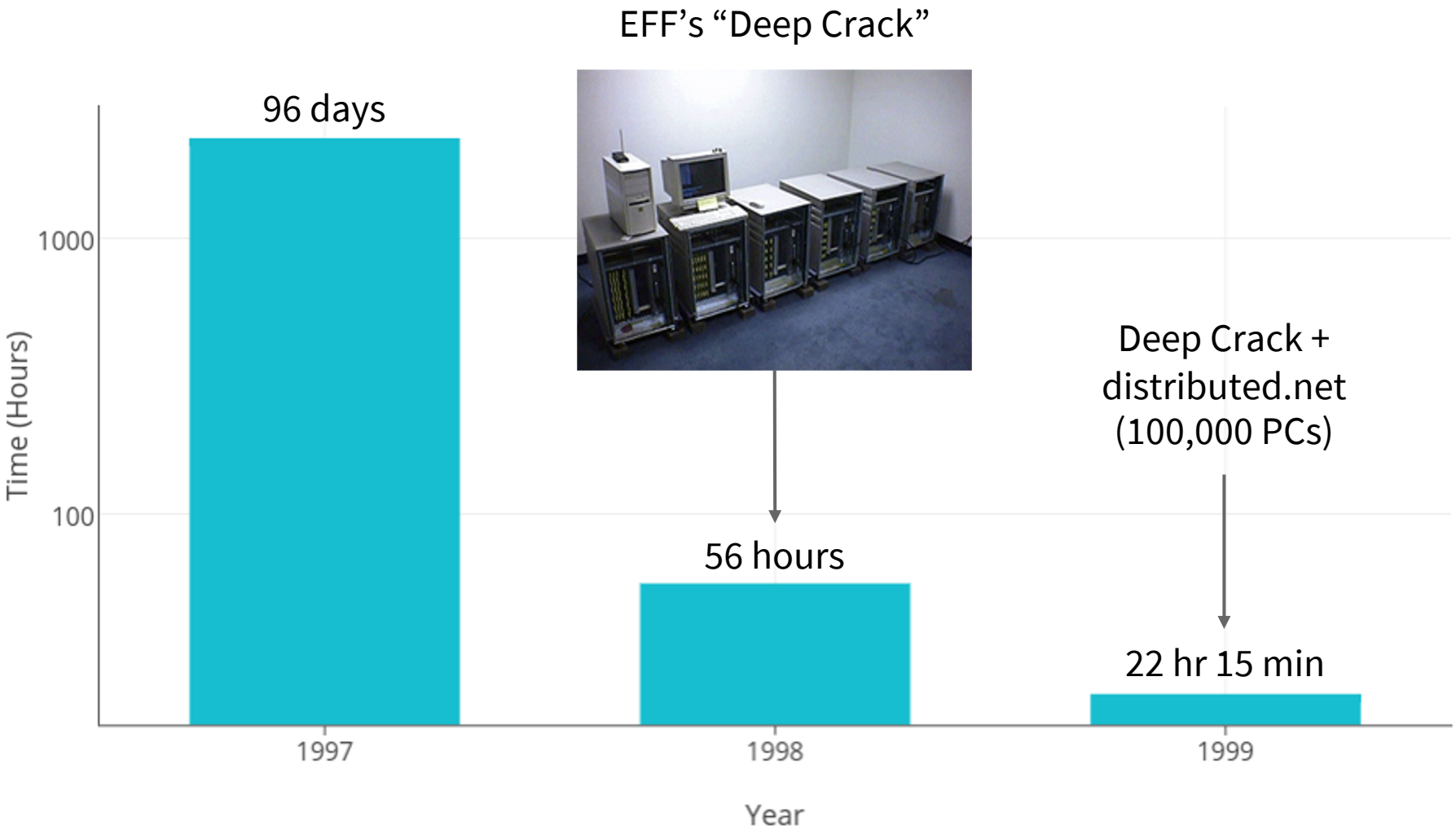
- How much time is required?
- How can we make brute-forcing less practical?

$T_{crack} = N_{trials} (T_{decrypt} + T_{check})$ where N_{trials} is average no. of trials

$= 2^{n-1} (T_{decrypt} + T_{check})$ where n is no. of bits in key

Cracking DES (56-bit Key)

UNIVERSITY OF LEEDS



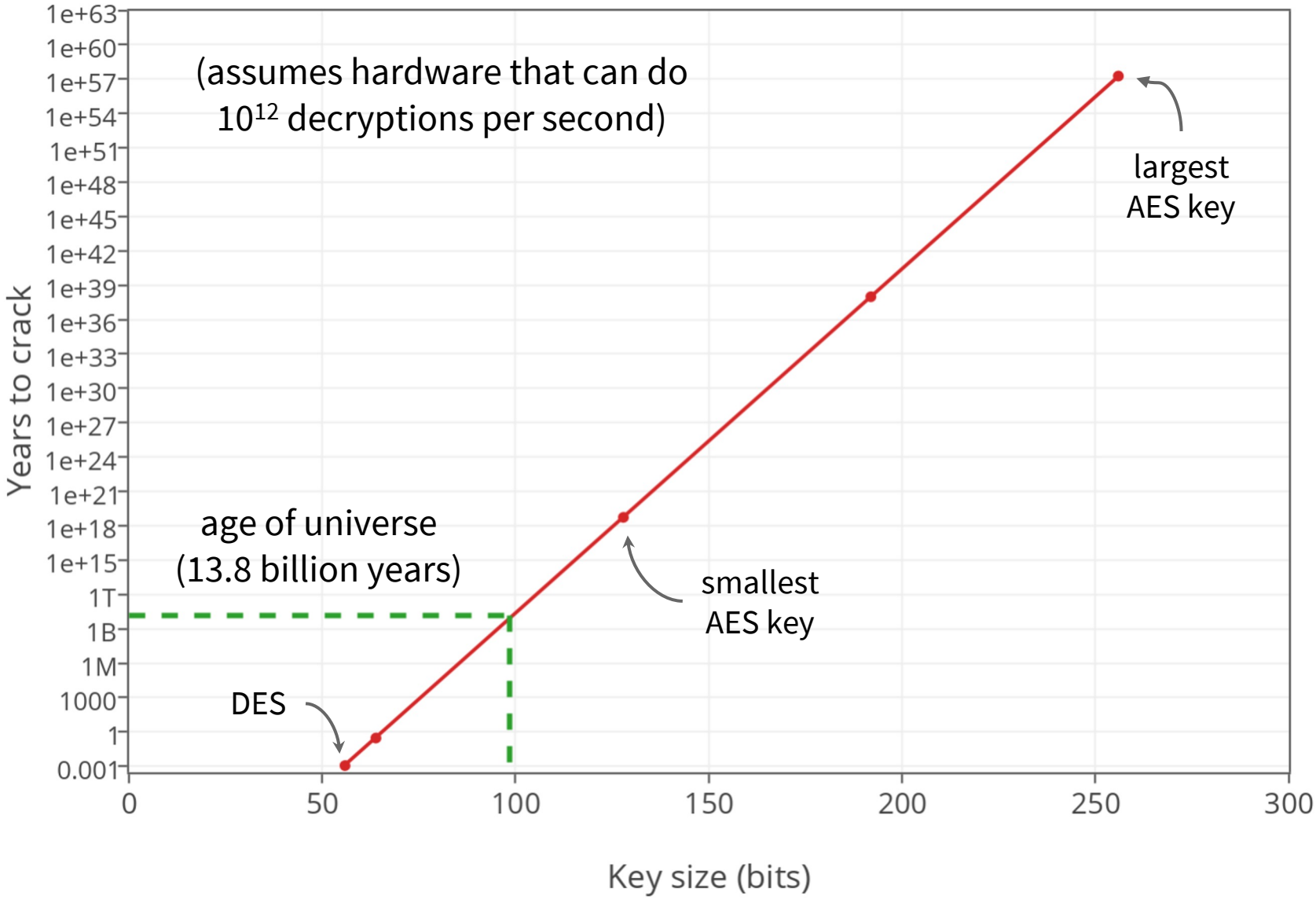
Today?

UNIVERSITY OF LEEDS

- GPUs
  - Massive parallelism from (up to) hundreds of cores
- Field-Programmable Gate Arrays (FPGAs)
  - Orders-of-magnitude better cost-performance ratio than using PC clusters for brute-forcing
  - Much lower power consumption than PCs
- The Cloud
  - Use of VMs and economies of scale in large data centres make compute power a (relatively) low-cost utility
  - Easy to create a large network dedicated to cracking without ever leaving your browser

Effect of Key Size

UNIVERSITY OF LEEDS



Caveats

UNIVERSITY OF LEEDS

- Graph assumes computing power is fixed; if Moore's Law continues indefinitely (unlikely) then 128-bit keys could be vulnerable within a century
- Ignores practical limitations on brute-forcing (economic cost, power requirements, etc)
- Ignores possibility of significant weaknesses being discovered in standard ciphers (unlikely)
- Ignores radical technology – e.g., quantum computing

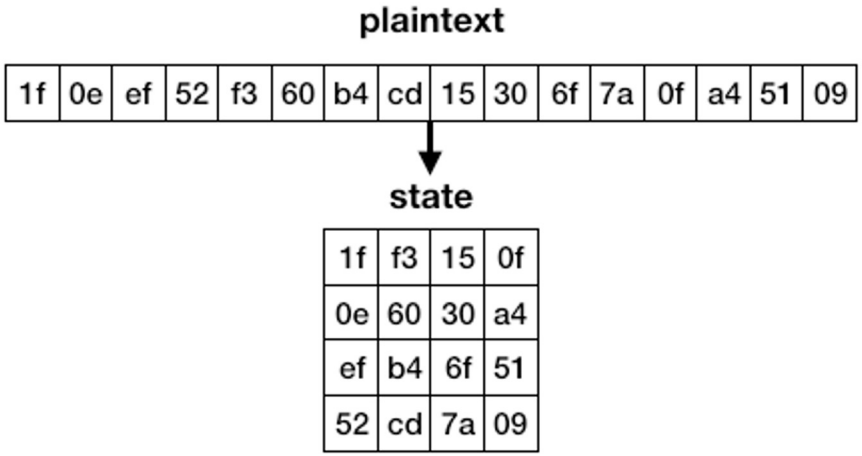
Advanced Encryption Standard (AES)

UNIVERSITY OF LEEDS

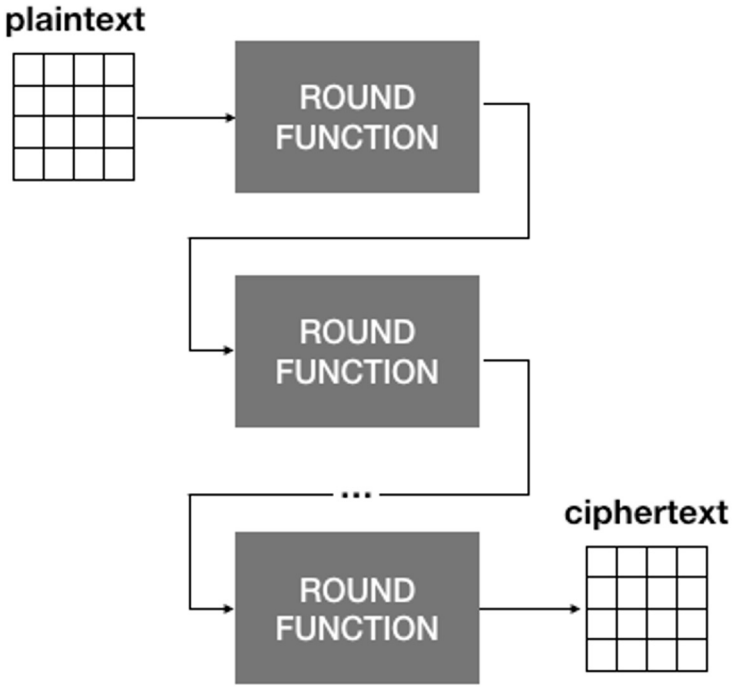
- Rijndael algorithm, standardised in 2001
- **Block cipher**, with 128-bit blocks
- Key sizes of 128, 192 or 256 bits
- Fast in software
- Has hardware support on Intel & AMD
    - AES-NI (‘New Instructions’) instruction set

AES Block Processing

UNIVERSITY OF LEEDS



16-byte blocks of plaintext are treated as 4 × 4 squares

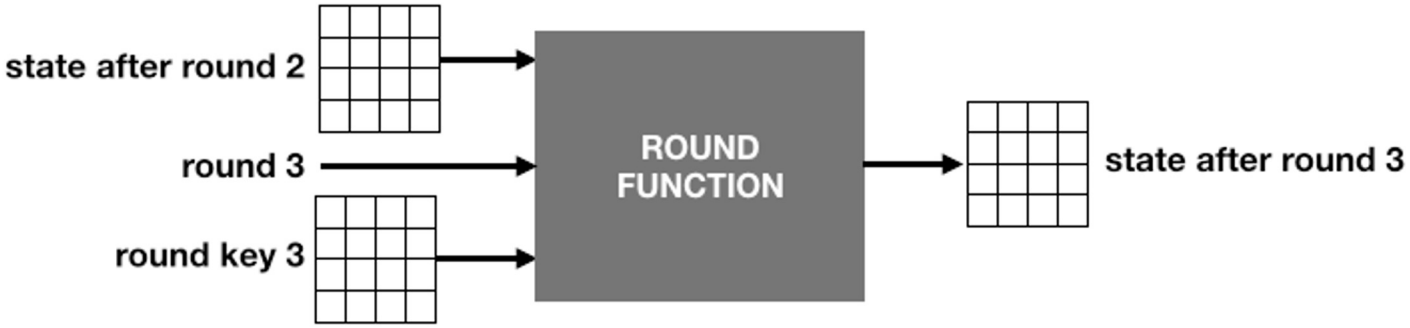


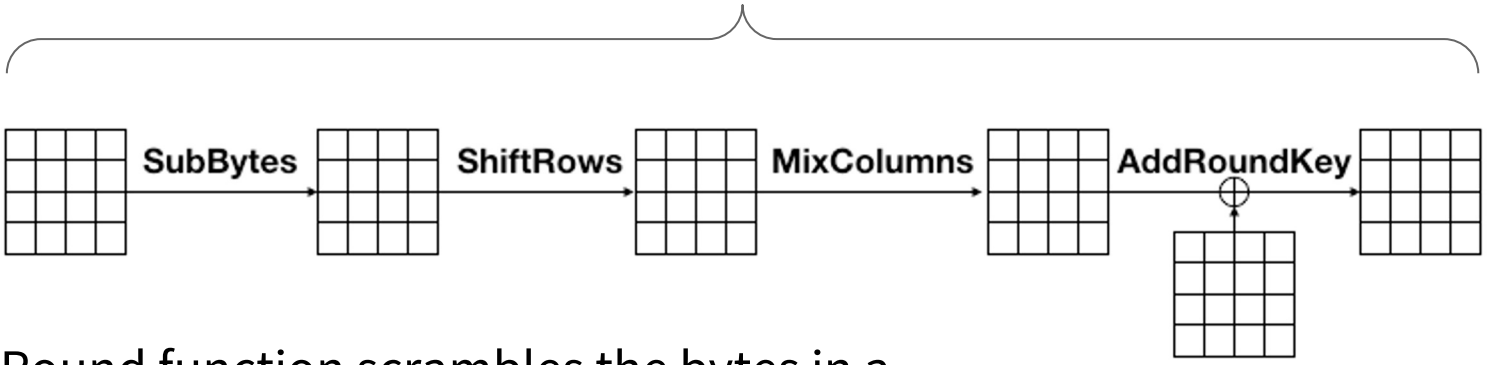
4 × 4 square is fed through multiple iterations of a **round function**, finally yielding ciphertext

AES Round Function

UNIVERSITY OF LEEDS

Round function takes three inputs: the 4 × 4 square output by the previous round, the round number, and a **round key**

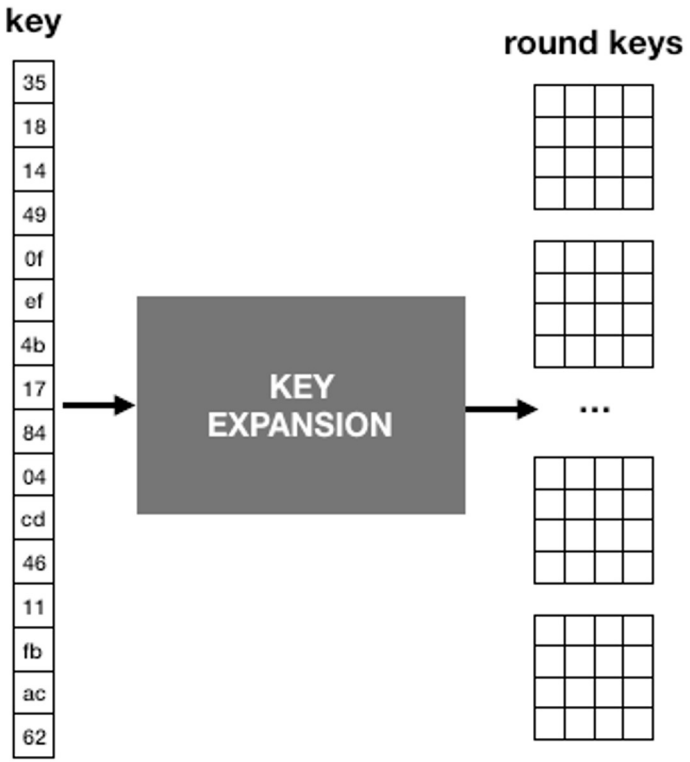




Round function scrambles the bytes in a complex way, combining four different operations

AES Round Key Derivation

UNIVERSITY OF LEEDS



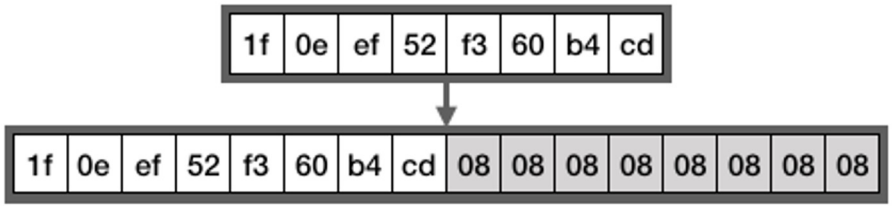
Each round uses a different round key, generated from the main key via a **key schedule**

AES-128 needs 11 round keys
AES-192 needs 13
AES-256 needs 15

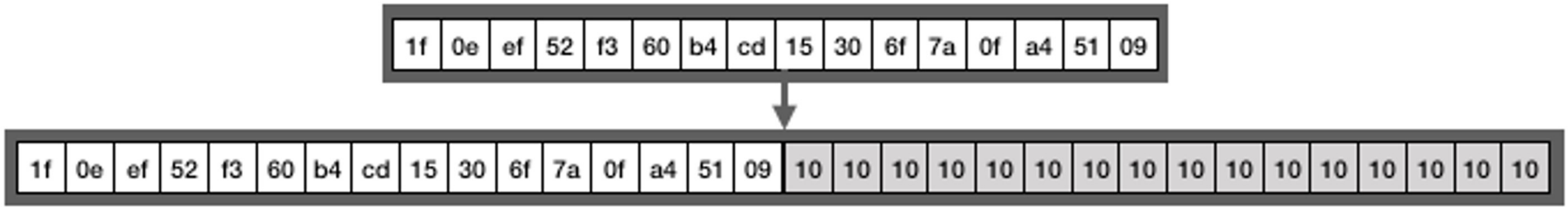
Padding

UNIVERSITY OF LEEDS

Need to pad out the plaintext to a multiple of the cipher block size



In the **PKCS#7** scheme, each padding byte has a value equal to the amount of padding required



If plaintext length is already a multiple of block size, we add an entire block of padding to the end, giving each byte a value of 16 (0x10)

ECB Mode

UNIVERSITY OF LEEDS

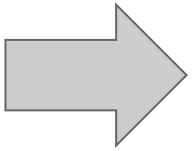
- ‘Electronic Code Book’ mode
- Each block is processed independently
- A particular block of plaintext always encrypts to the same block of ciphertext, regardless of its position
- … so if there is repetition in the plaintext, this same pattern will be visible in the ciphertext!
- **Never use a symmetric cipher in ECB mode!**

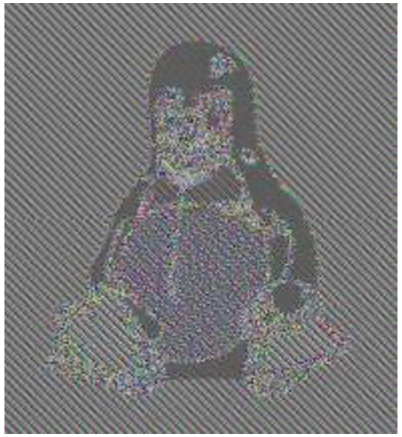
A More Graphic Example

UNIVERSITY OF LEEDS

original image

encrypt pixel values
in ECB mode



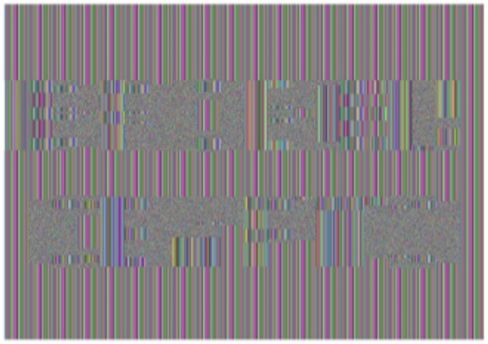


encrypted image

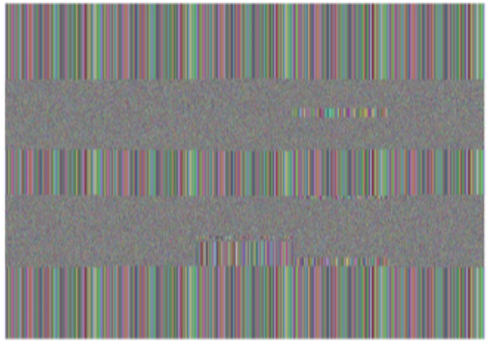
(a) Plaintext image, 2000 by 1400 pixels, 24 bit color depth.

(b) ECB mode ciphertext, 5 pixel (120 bit) block size.

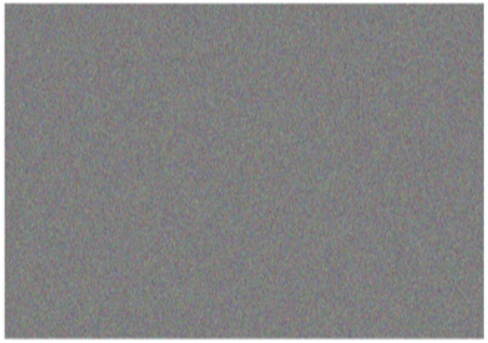
(c) ECB mode ciphertext, 30 pixel (720 bit) block size.



(d) ECB mode ciphertext, 100 pixel (2400 bit) block size.



(e) ECB mode ciphertext, 400 pixel (9600 bit) block size.



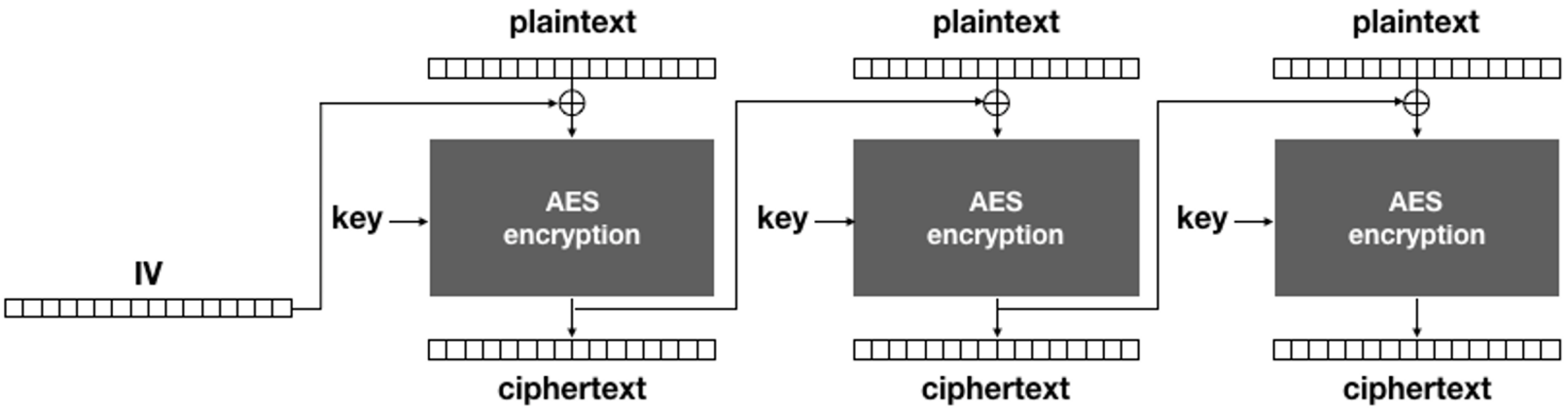
(f) Ciphertext under idealized encryption.

Encrypting over a larger blocks results in fewer identical blocks, therefore reduced opportunities for information leakage

But we can't change block size of a standard cipher like AES!

Cipher Block Chaining

UNIVERSITY OF LEEDS



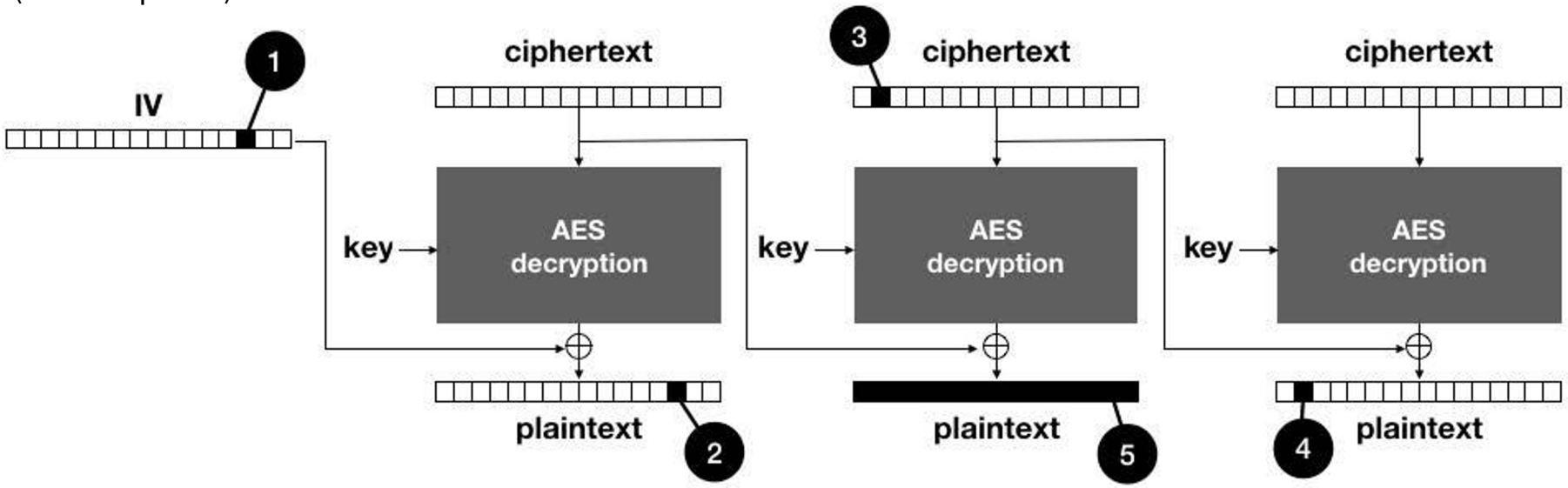
- $C_i \longleftarrow E_K(P_i \bigoplus C_{i-1})$
- Repeating plaintext blocks encrypt differently!
- IV **must be random** and be shared with recipient

Attacking CBC

UNIVERSITY OF LEEDS

Attacker flips a bit from 1 to 0 in IV (which is public)...

Attacker flips a bit in a block of ciphertext…



...this flips corresponding bit in first block of plaintext

But current block of plaintext will be corrupted as a result

...which will flip the corresponding bit in the next block of plaintext

So we need integrity checking – e.g, using HMAC

AES-CBC-HMAC

UNIVERSITY OF LEEDS

- Compute an HMAC over the IV and ciphertext
  - Use SHA-256 as the hash function
  - Preferably don’t use cipher key as the shared secret (domain separation)
- Resulting authentication tag is concatenated with the IV and ciphertext and transmitted
- Message recipient checks the auth tag using a constant time algorithm, to defeat timing attacks

AEAD

UNIVERSITY OF LEEDS

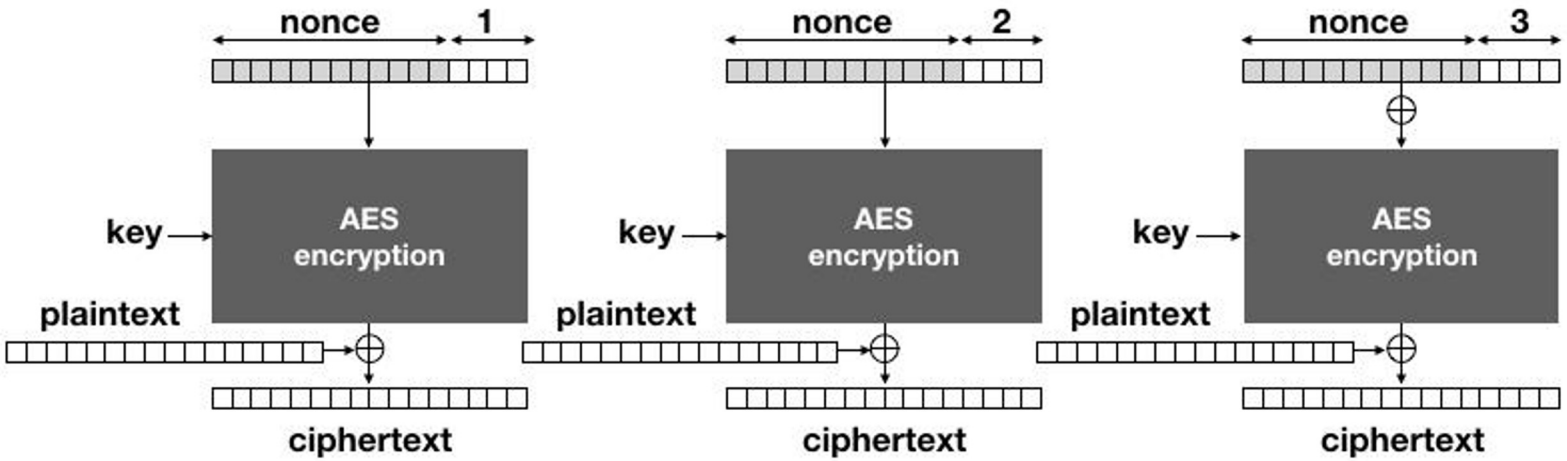
- ‘Authenticated Encryption with Additional Data’
- Broadly similar to AES-CBC-HMAC but includes optional additional data when computing auth tag
- Additional data is authenticated but not encrypted
- Most popular version of this is **AES with Galois/Counter Mode** (AES-GCM)
  - Uses CTR as cipher mode
  - Use GMAC algorithm to compute auth tag

CTR Mode

UNIVERSITY OF LEEDS

nonce ('number used once') should be unique but doesn't have to be random

96-bit **nonce** is concatenated with a 32-bit incrementing counter and encrypted



AES output, known as the **keystream**, is XORed with plaintext blocks to yield ciphertext

keystream is simply truncated if we don't have a full block of plaintext – i.e., **no padding**

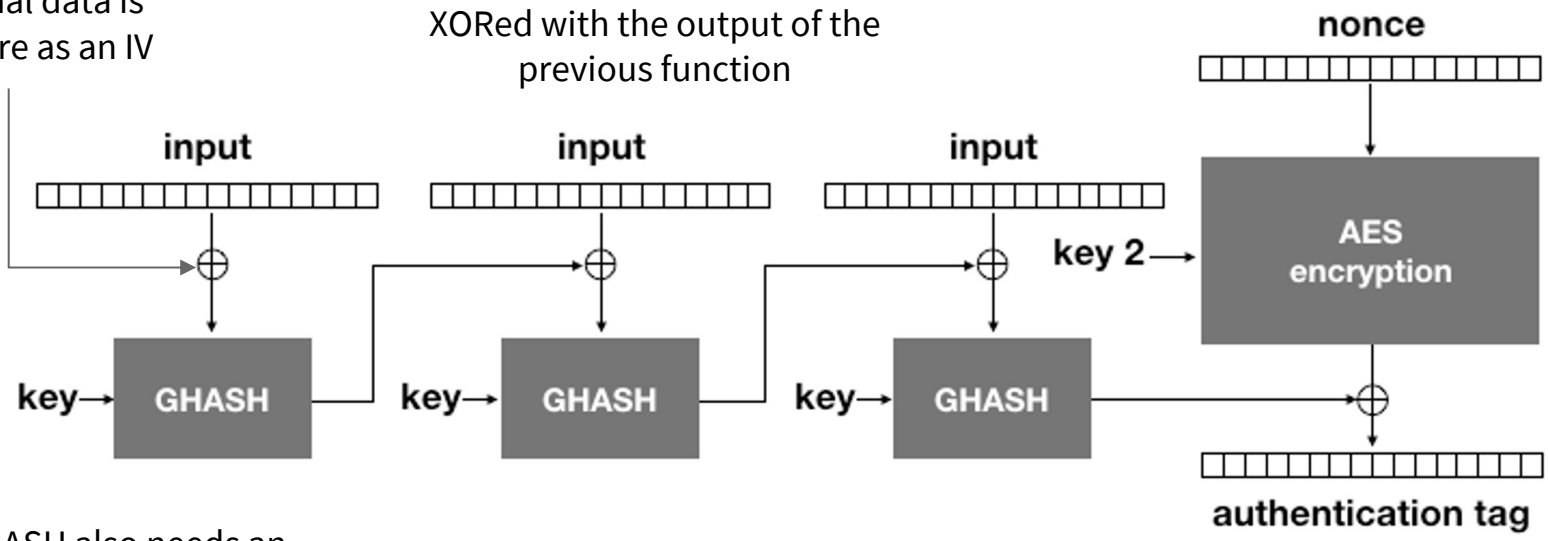
crucial that the nonce isn't reused

GMAC Algorithm

UNIVERSITY OF LEEDS

hash of the additional data is fed in here as an IV

input to GHASH function is the ciphertext generated by AES-CTR, XORed with the output of the previous function



GHASH also needs an **authentication key**, which is computed using the cipher key

nonce + counter of 0 is encrypted and then XORed with final GHASH output to produce the auth tag

Nonce Uniqueness

UNIVERSITY OF LEEDS

- Use a counter
  - 2^{96} values available, so we won't run out
  - But need to maintain state: what if machine crashes?
- Use a random value
  - 2^{96} possibilities, so initially unlikely to reuse a value
  - But using more than ~2^{30} nonces for a given key is inadvisable (birthday bound)
  - Example: Visa processes 150 million transactions per day, so would hit this limit after a week!

Summary

UNIVERSITY OF LEEDS

We have

- Discussed the basic principles of symmetric ciphers
- Explored the feasibility of brute-forcing and the significant weakness of ECB mode
- Investigated the inner workings of AES
- Seen how AES in practice requires message authentication
- Examined the most popular authenticated encryption algorithm, AES-GCM

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Code Examples
- Exercise 5 (AES, ECB mode, CBC mode)
- Exercise 6 (AES-GCM in Java)
- Paper on key cracking using FPGAs (PDF)
- Sweet32: ‘birthday attacks’ against 64-bit block ciphers