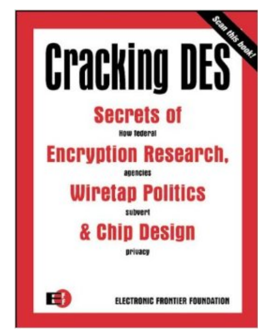
Introduction to Cyber Security

Fall 2017 | Sherman Chow | CUHK IERG 4130

Chapter 6
3DES, AES, Mode of Operations

Years go by…

- **DES Challenge**: find $k \in \{0,1\}^{56}$ s.t. $\mathrm{DES}(k, m_i) = c_i$ for $i = 1, 2, 3$
- ’94: DES was reaffirmed by NIST for US Federal Government use for another 5 years, i.e., due 1999.
- ’97: Internet search (cooperation of internet-connected computer): 3 months
- ’98: Electronic Frontier Foundation (EFF)’s “Deep Crack”: 3 days
  - Initial costs: US$ 220,000
  - US$ 150,000 to replicate a machine (in 1998)

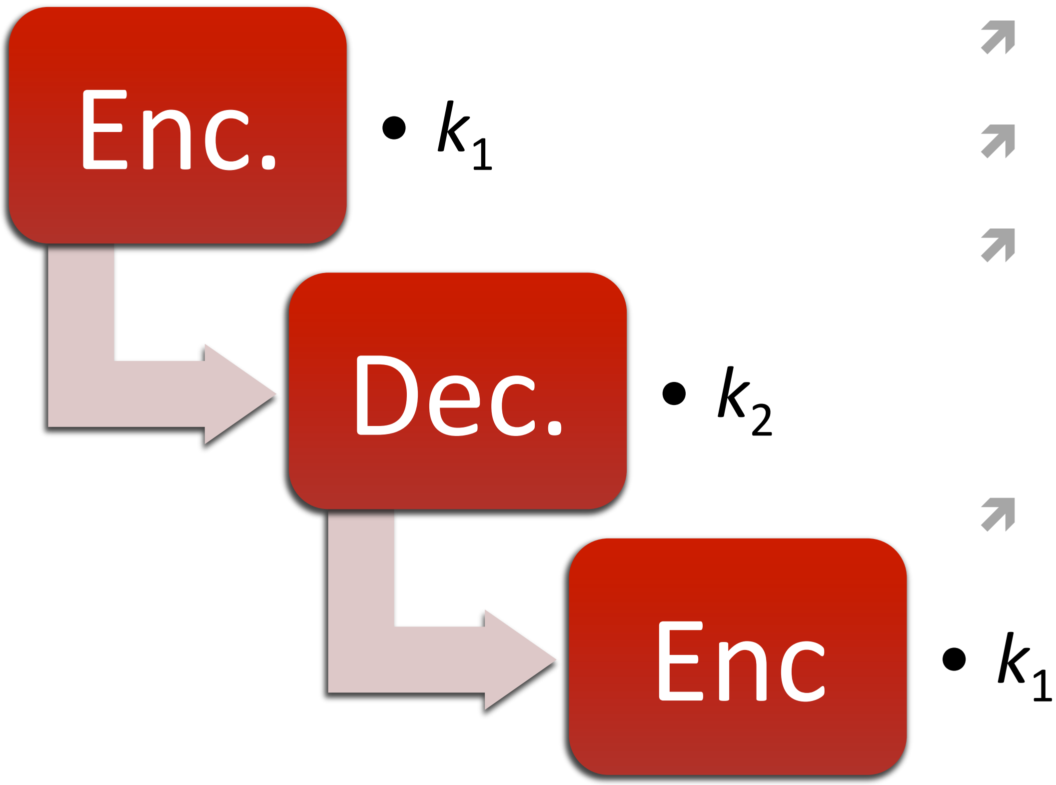


- ’99: Combined search (EFF + Distributed.net): 22 hours
- ’99: DES NIST issued a new standard requiring “Triple DES” to be used.

All these years go by…

- Moorse's Law: hardware price/performance improving 40% per year ➔ keys must grow by about 1-bit every 2 years
- DES was designed in 1979, if 56-bit key was just sufficient…
- 64-bit is about right in 1995
- 128 bits would suffice until 2123?
- 2006: COPACOBANA: 7 days at $10,000
  - Parallel architecture, based on 120 low-cost FPGAs

Triple-DES



- Triple WHY!
- Why two keys, not three?
- Why three encryptions, not less or more?
  - Well, the more stages the slower the whole process
- Why EDE, not EEE or EDD?
  - Backward compatibility
  - (imagine a legacy system only interface with DES)

Let's Meet in the Middle (MITM)

$$m \rightarrow E(k_1, \cdot\,) \rightarrow E(k_2, \cdot\,) \rightarrow c$$

X		X		X		X	
	X		X		X		X
X		X		X		X	
	X		X		X		X
X		X		X		X	
	X		X		X		X
X		X		X		X	
	X		X		X		X

- Given one (or several) pair(s) of (m, c), find the key (k_1, k_2)

MITM Attack

Key observation: $D(k_2, c) = E(k_1, m)$

- Make a Forward Table with 2^{56} entries
  - Each entry consists of a DES key k_1
  - and the result r of applying that key k_1 to encrypt m.
  - Sort the table in numerical order by r
- Make a Backward Table with 2^{56} entries
  - Each entry consists of a DES key k_2
  - and the result r of applying that key k_2 to decrypt c.
  - Sort the table in numerical order by r
- Match two tables and find the candidate key (k_1, k_2) such that $D(k_2, c) = E(k_1, m)$
  - Sorting is just for easy comparison
  - A naïve way is to cross check each pair of entries from both tables ($2^{56} \times 2^{56}$)

MITM Attack (cont.)

- What if multiple entries match?
- Try once more, with another pair <*m*’, *c*’>
- The “real” key-pair will always work
- The other “coincident” key-pairs will almost surely fail on at least one of the other <*m*, *c*> pairs

Analysis

- Known plaintext attack
- Space complexity
  - 2^{57}
- Time complexity
  - Forward (i.e., trial encryption + sorting) + Backward (same 2 stages)
  - $2(2^{56} + 2^{56}\log(2^{56}))$ // recall that sorting *n* numbers takes *n*lg(*n*)
  - $< 2^{63}$
  - $<< 2^{112}$
- How about MITM on 3DES?

Advanced Encryption Standard (AES)

- NIST had an open call for proposals (actually a contest) in 1997
- Resistance to known attacks and randomness tests
- Complexity
- Efficient hardware and software implementation
- Flexibility, i.e., can be parameterized easily
  - e.g., lengths for a key and a block
- 21 submissions from all over the world
- 15 fulfilled all the requirement
  - 8 from North America, 4 from Europe, 2 from Asia, 1 from Australia

AES Winner

- After testing and evaluation, shortlist in Aug ’99:

Algorithm	Complexity	Speed	Security
MARS (IBM)	Complex	Fast	High
RC6 (USA)	Very Simple	Very Fast	Low
Rijndael (Belgium)	Clean	Fast	Good
Serpent (Euro)	Slow	Slow	Very High
Twoflsh (USA)	Complex	Very Fast	High

- Rijndael wins in ’00, standardized as AES effective May ’02
- Contrast: few complex rounds verses many simple rounds

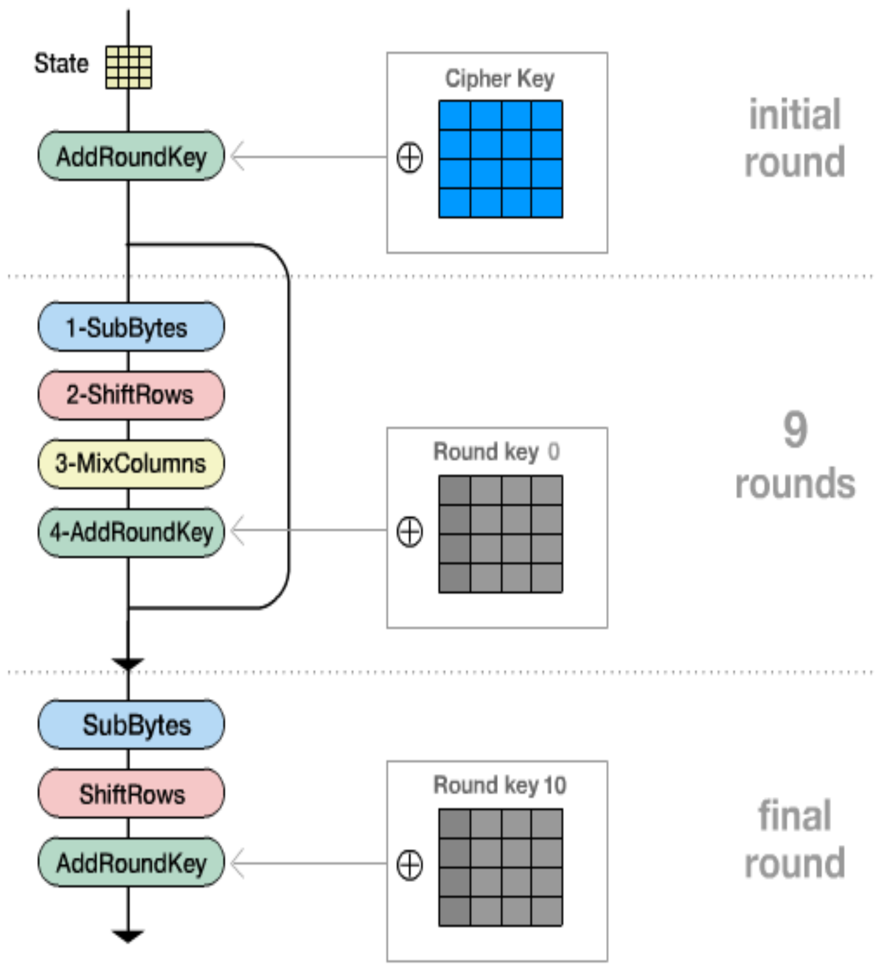
Properties of Rijndael

- Rijndael supports multiple block sizes that can be reconfigured to support key lengths of 128, 192, 256
- Number of Rounds depends on key length
- AES standard uses 128 bits per block
- Rijndael does not use the Feistel structure
  - Unlike several other NIST AES contest finalists
- It relies on special properties in “Generalized Field” Mathematics for computing the “inverse”, i.e., decryption in this context.

Implementations of Rijndael

- Unlike DES, the algorithm/hardware for Rijndael encryption and decryption process are not identical, but differ slightly.
- Unlike DES, Rijindael also features a fast software implementation.
- Software Implementation of Rijndael performs well across a wide range of platforms, from 8-bit (Smartcard like) to 64-bit CPU
  - 24+ Mbps enc/decryption on a 200MHz Pentium Pro, Borland C++
  - JavaScript AES (http://crypto.stanford.edu/sjcl)
- Fastest in Hardware amongst all the finalists
  - ASIC implementation by NSA demonstrated performance ranging from 443 to 606 Mbps, depending on key-length and mix of key scheduling

Overview of AES



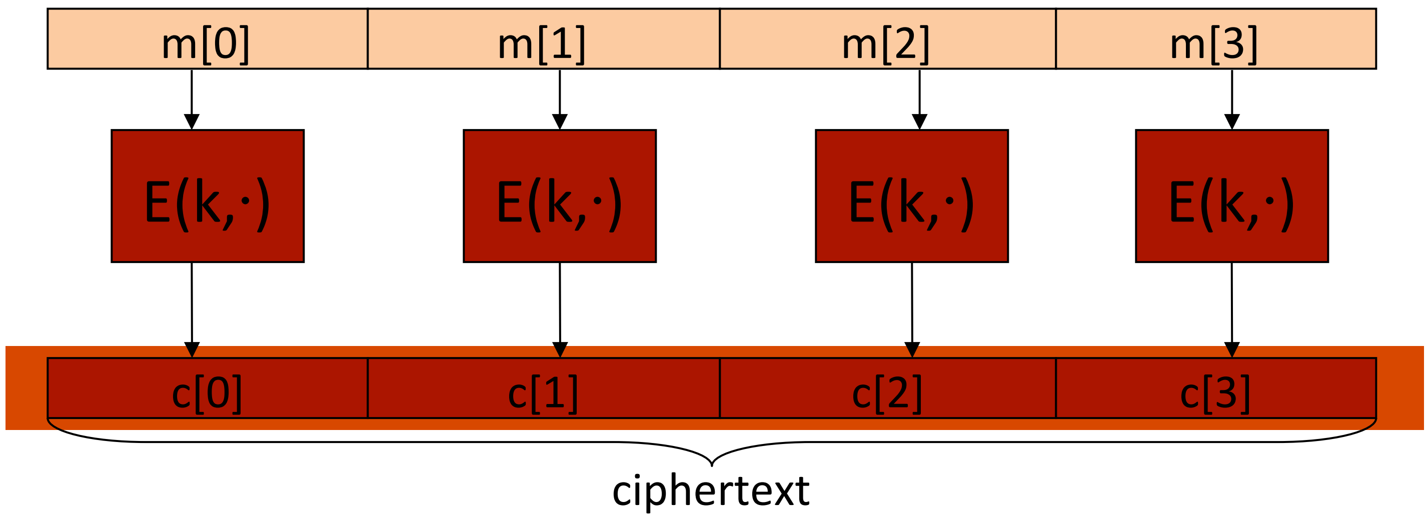
Key-Reusing, and Weak randomness

- One-time pad is perfectly secure.
- How about two-time pad?
- Never use random() for crypto!
  - E.g. Kerberos v. 4

Mode of Operations

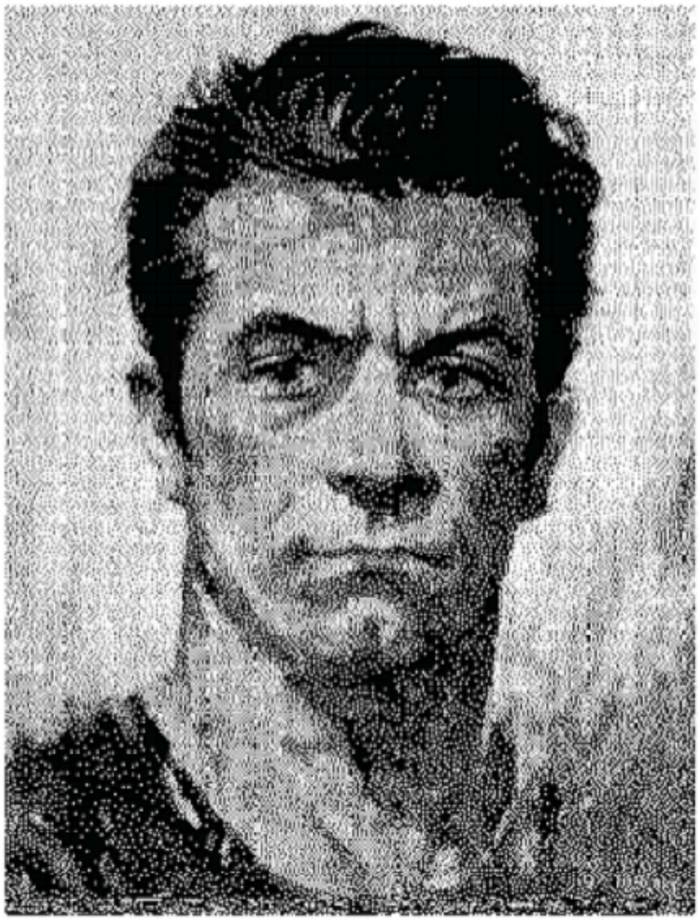
- What if I want to encrypt more than 1 block?
- Electronic Code Book (ECB mode)
- Using a block-cipher as a mean for authentication?
- What if I want to authenticate more than 1 block?
  - (You guessed correctly, our next chapter is on authentication.)
  - (Yes, we will see mode of operation in authentication again.)

ECB, depicted



- Throughout (most of) the slides, we use 4-block as example
- If you got more/less blocks, you have a wider/narrower circuit

ECB depicted (literally)

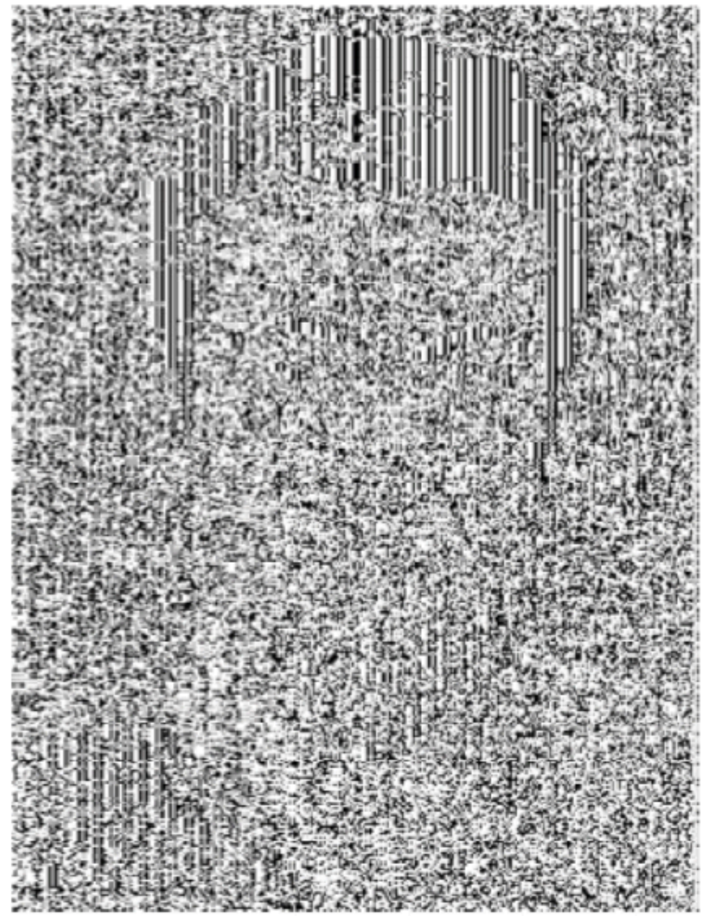


ECB-encrypted

with a

large AES-key

→



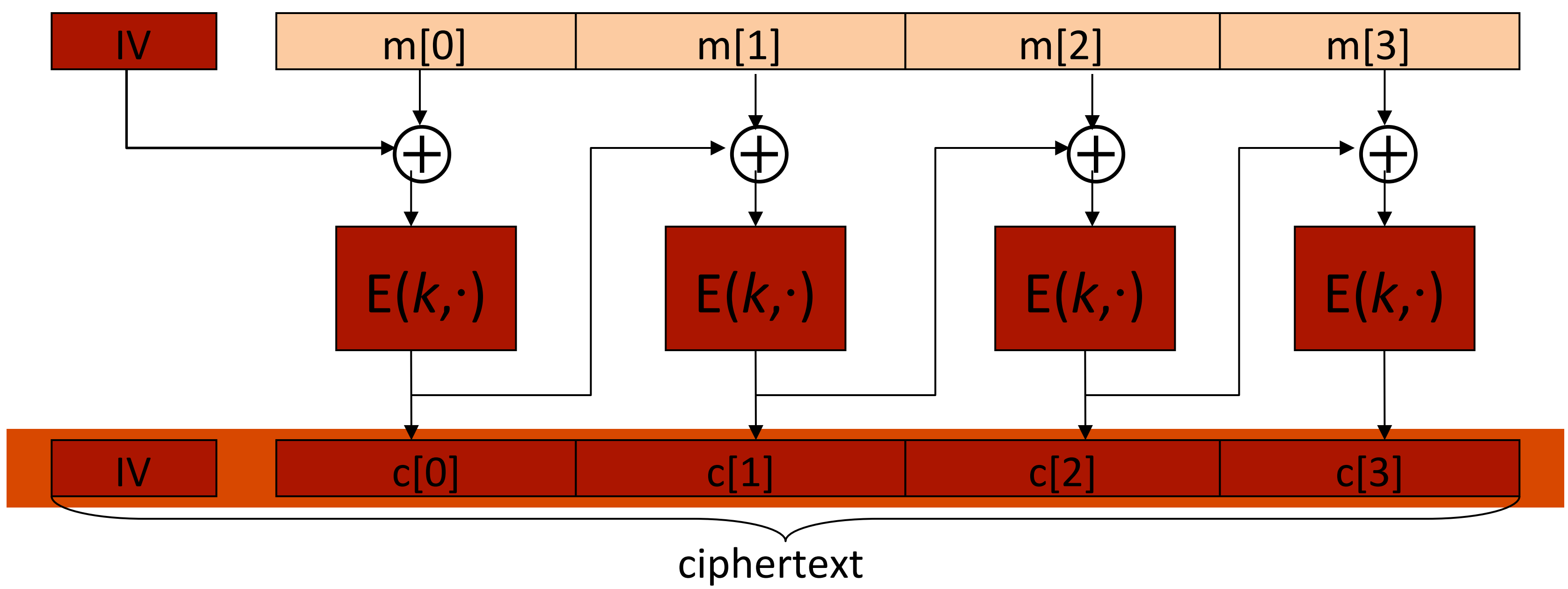
[Source: Bart Preneel]

Discussions of ECB and Randomization

- ECB is not "*semantically secure*" (More info at. IERG 5240 / ENGG 5383 :)
  - Deterministic encryption: Same plaintext -> Same ciphertext -> reveal "semantic"
  - E.g., consider you learn that E("yes") = "XYZ", you do not need to decrypt "XYZ" next time.
- Randomness is required, we call the randomness "*initialization vector*" (IV)
  - Wait, is it just like the secret key?
  - No, its secrecy is not needed for ciphertext's confidentiality
  - Yet, it may be required to be unpredictable during encryption c.f. TLS CBC IV Attack [**]
- Other modes of operation are invented, e.g., CBC by IBM '76
  - Only one IV, how to "randomize" the encryption of multiple blocks?
  - One way is to "chain" them together.
- Is security all we care about in communication?

Cipher-Block Chaining (CBC) w/ Random IV

- $c[0] = E\big(k, IV \oplus m[0]\big)$, $c[1] = E\big(k, c[0] \oplus m[1]\big)$, …

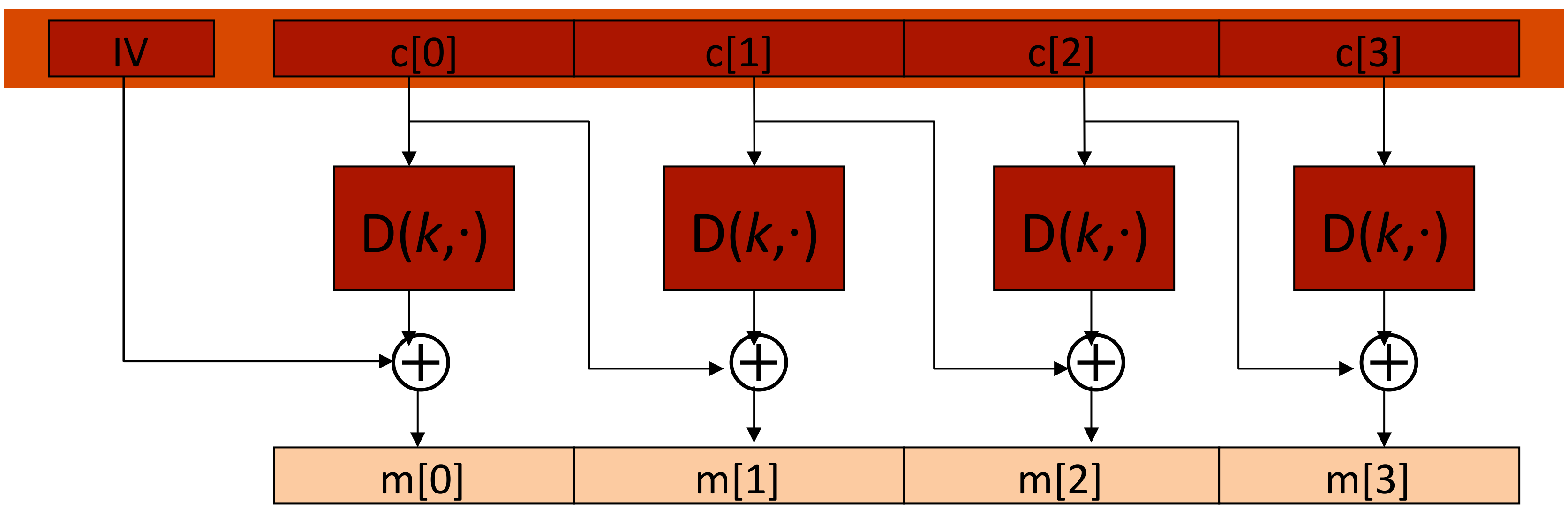


Discussion of CBC

- Encryption is sequential
  - Cipher (e.g., DES/AES) operations E() or D() is time-consuming
  - when compared with bitwise-XOR operation
- Message must be padded to a multiple of the block size
- One bit change in IV affects all subsequent ciphertext blocks
  - Good for security, since IV is our source of randomness
- One bit change in plaintext affects all subsequent ctxt. blocks

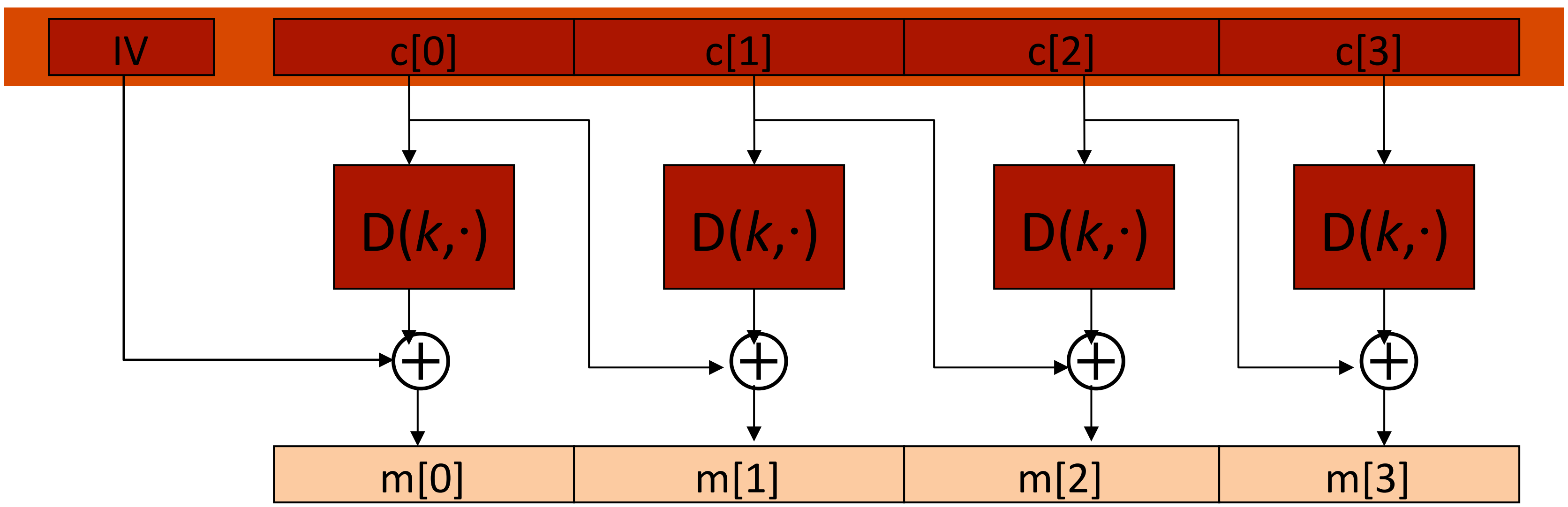
Decryption Circuit

- $c[0] = E\big(k, IV \oplus m[0]\big) \Rightarrow m[0] =$ [redacted]



Discussion of CBC

- Can decryption be parallelized?
- What if incorrect IV is used during decryption?
- What's the consequence of one bit change in a ctxt. block?

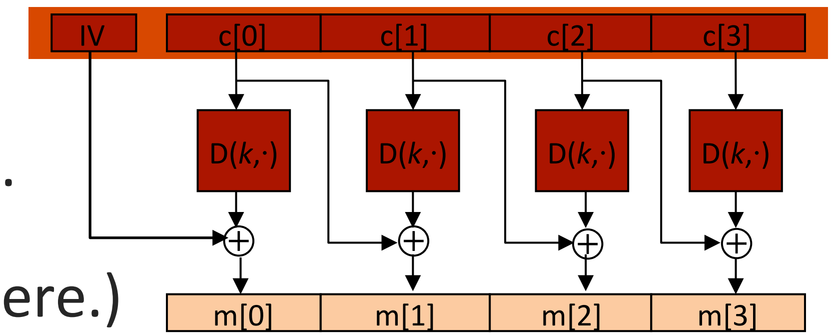


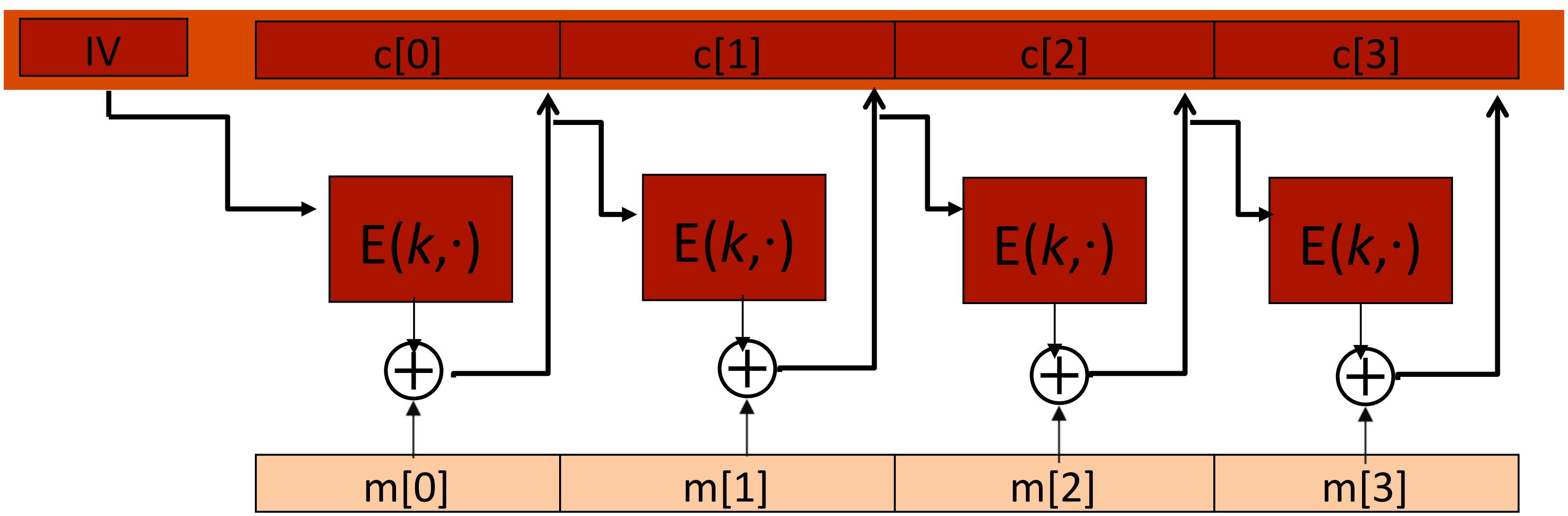
Ciphertext FeedBack (CFB) Mode

- A friend of CBC, recall:
  - $c[i] = E\big(k, c[i\text{-}1] \oplus m[i]\big)$ // c[-1] = IV
  - $m[i] = D\big(k, c[i]\big) \oplus c[i\text{-}1]$
- Now CFB is:
  - $c[i] = E(k, c[i\text{-}1]) \oplus m[i]$
  - $m[i] = \mathbf{E}(k, c[i\text{-}1]) \oplus c[i]$
- Is encryption/decryption parallelizable?
- Error propagation?
  - One bit change in IV?
  - One bit change in ctxt. block?

CFB Encryption

- Recall CBC-Decryption →
- And below is CFB-Encryption.
- (CFB-Decryption is omitted here.)



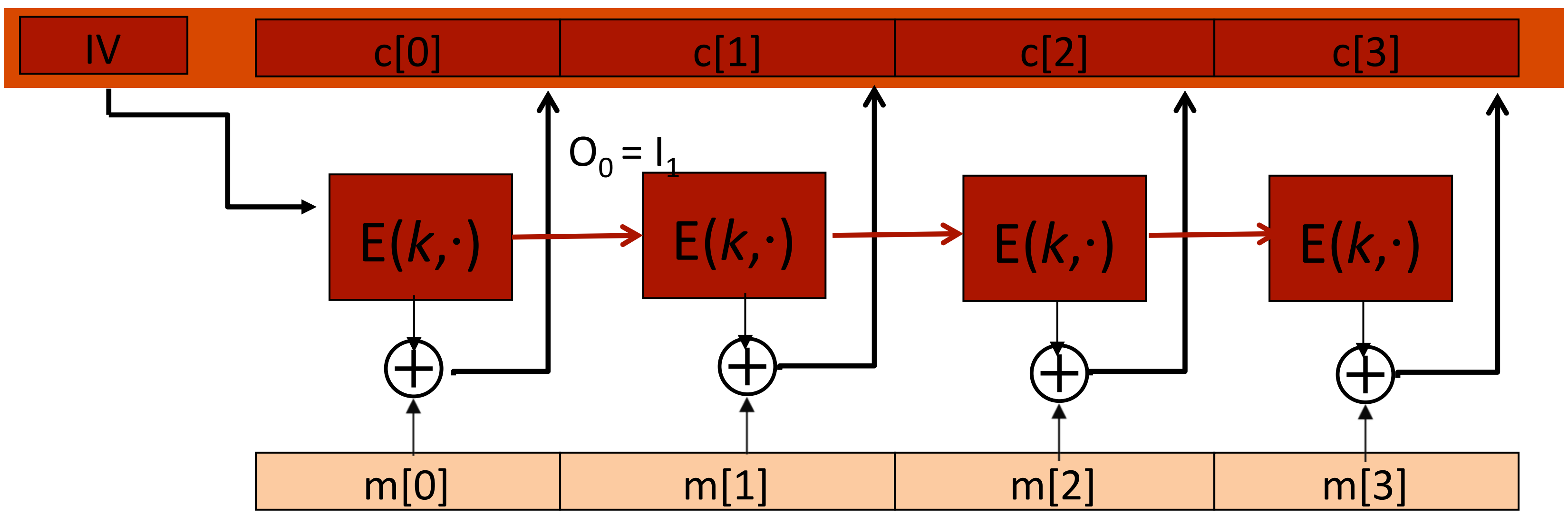


Discussion of CFB

- Result is "feedback" for next stage (hence name)
- Standard allows any number of bit to be fed back
  - e.g., CFB-1, CFB-8, CFB-64, etc.
  - of course, most efficient to use all 64 bits
- Turns a Block Cipher into a Stream Cipher (kind-of)
  - message is treated as a stream of bits
  - each character can be **encrypted** and **transmitted immediately**
  - operates in **real time**
  - not really, need to stall for every *s* (e.g., *s* = 8 for CFB-8) bits
- Can be made "self-synchronizing" [**]

Output FeedBack (OFB) Mode

- CFB = Ctxt FeedBack. Now we feedback (blockcipher's) output.
- I_0 = IV, $O_j = E(k, I_j)$, $I_j = O_{j-1}$,; $c_i = m_i \oplus O_i$;
- Executions of E() are independent of message (the red arrows)



Caveats of using OFB

- Must never re-use the same “sequence”/padding (i.e., same key & IV)
- Only OFB-64 (i.e., full-block) should ever be used
- If c[*i*] has error, all m[*j*]’s **other** than m[*i*] are not affected.
- If IV has error, total disaster
- Not parallelizable, but pre-computable
- Synchronous [**]
  - On the other hand, sender and receiver must remain in sync.
  - Some recovery method is needed to ensure this occurs

Counter (CTR) Mode

- Just counter, no feedback
- Again, what're "good"? And what're "bad"?
  - omitted but you should be able to answer by yourself