Review of last time

- Security in symmetric encryption:
  - The IND-CPA security game
  - A bit of PRF
  - How to prove OTP + PRG secure

- Proof techniques
  - Game hopping
  - Game equivalence by indistinguishability of games

PRPs and PRFs

Block ciphers, cryptanalysis, symmetric encryption

PRG IN OTP

- Recall the OTP
  - Traditional OTP for $\mathcal{K} = \mathcal{M} = \{0,1\}^m$
    - Choose random $k \overset{\$}{\leftarrow} \mathcal{K}$
    - Encrypt message m to : $c \coloneqq k \oplus m$
    - Decrypt ciphertext c as: $\hat{m} \coloneqq c \oplus k$

- Now replace random key generation by PRG:
  - OTP for $\mathcal{M} = \{0,1\}^m$ with $\mathcal{K} = \{0,1\}^n$ and $n < m$
  - Use a bounded-secure PRG $G: \{0,1\}^n \rightarrow \{0,1\}^m$
    - KeyGen: choose (once) $k \overset{\$}{\leftarrow} \mathcal{K}$
    - Encrypt message m as $c \coloneqq G(k) \oplus m$
    - Decrypt message as: $\hat{m} \coloneqq c \oplus G(k)$

Stream and Block Ciphers

Stream ciphers

- Based on pseudorandom generators
  - Usually in the PRG + OTP structure, encrypting traffic as it is sent
  - Note: symmetric in nature, and require synhronization for the masking string (output of PRG)

- Some examples: SEAL, A5, RC4
  - If PRG is efficient (it usually is), the construction is very fast
  - RC4 is probably the most often used stream cipher today, but some of its output bytes are biased, leading to breaking WEP and TLS + RC4

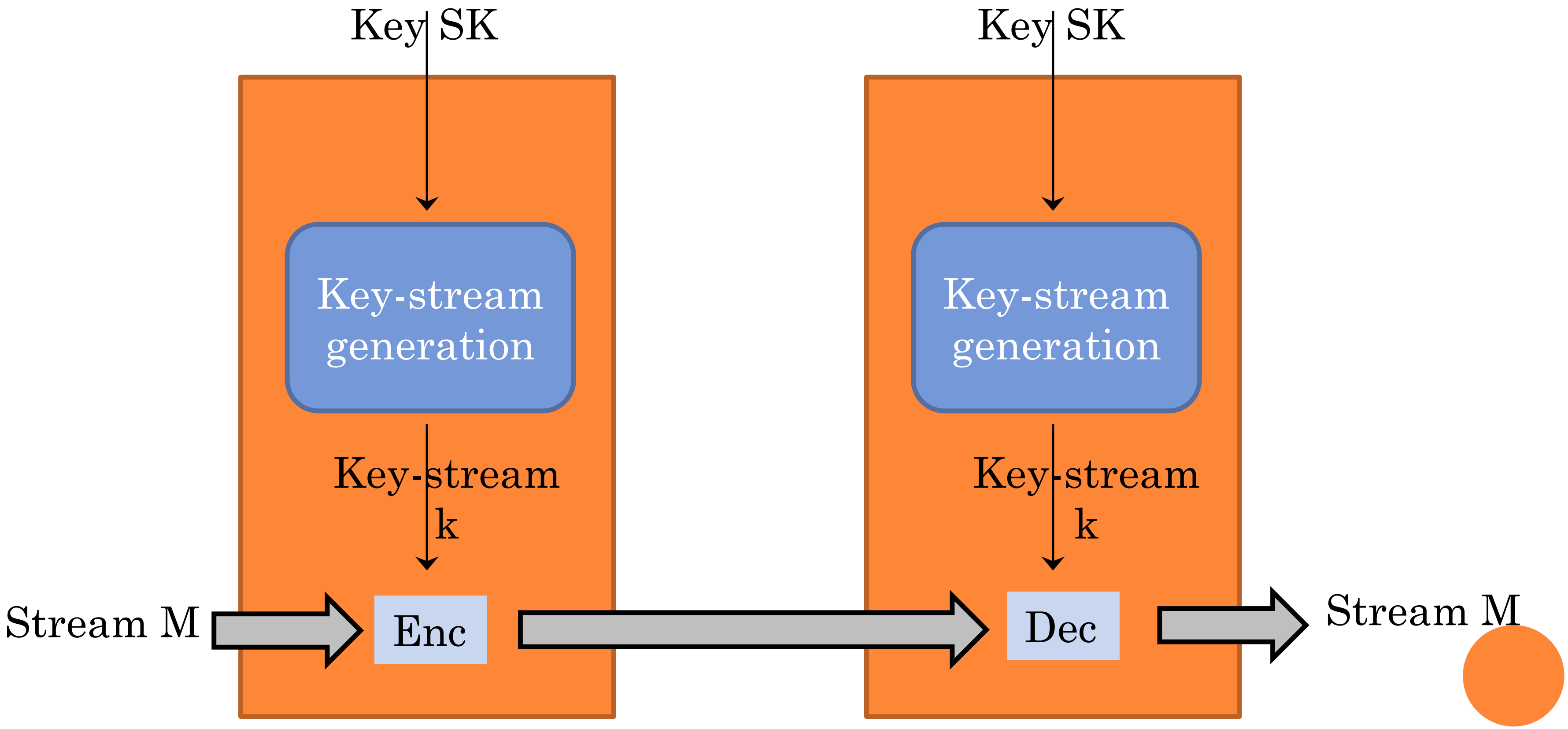
RC4

- Designed by Ron Rivest in 1987
- Used in protocols like TLS/SSL, WEP, etc.

- Starts with a key of 256 bytes: $k_0, \dots\ k_{255}$ (if not long enough, we pad it with itself)
- Also need permutation on (byte) positions $0, \dots, 255$, denoted S, which is shuffled at each round

Generic structure

- Stream ciphers must generate "pad" as we go



Generic structure

- Stream ciphers must generate “pad” as we go

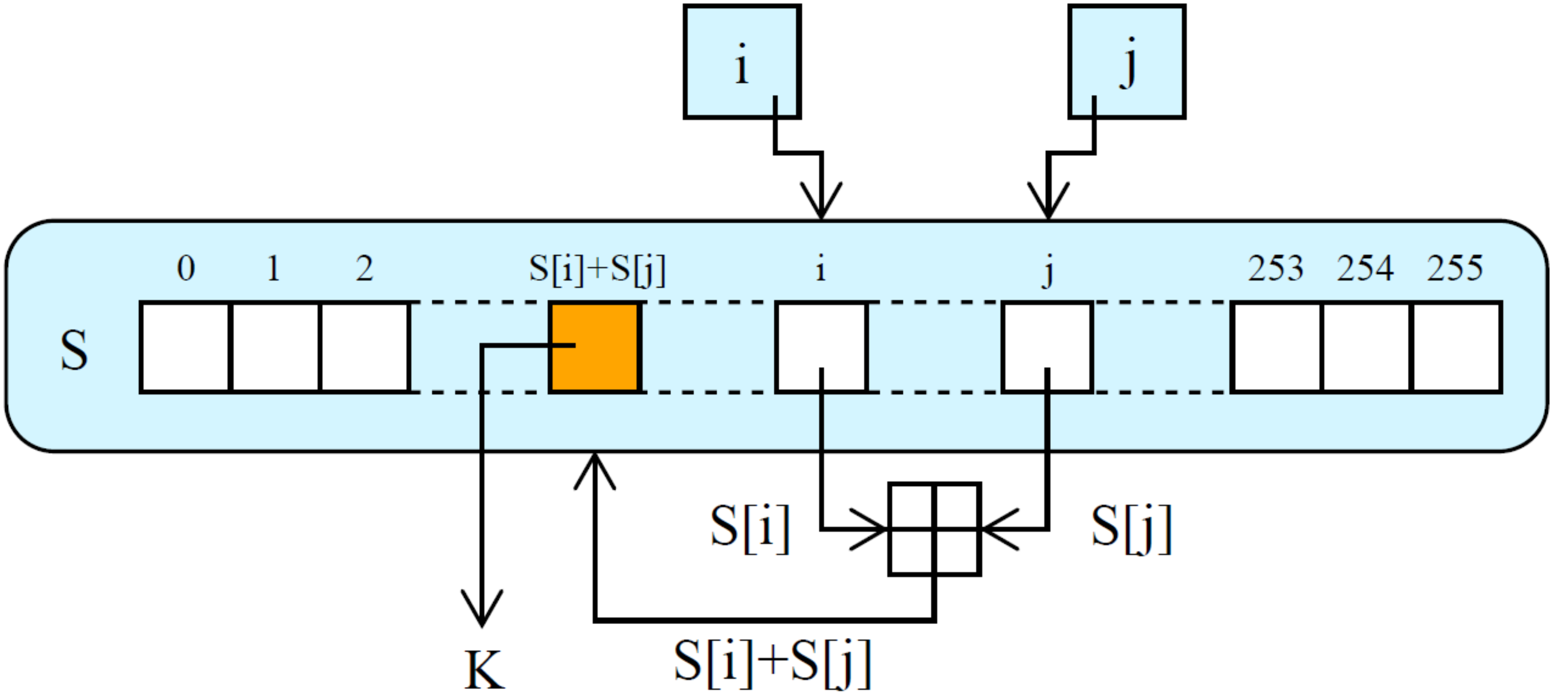
- Start with key K and a permutation S
- Do Key-Scheduling (KSA): use the key to initiate permutation
- Do PR generating algorithm (PRGA) to generate the key-stream

- Main problem: key-streams will eventually repeat themselves, and that’s where cryptanalysis strikes

RC4 DESCRIPTION

- Initialization:
  - $S_0 = 0;\ S_1 = 1; \ldots\ S_{255} = 255$
  - Key $K_0; \ldots\ K_{255}$
  - Current index $j = 0$
- KSA (instantiate S)
  - For $i = 1\ to\ 255$:
    - $j \coloneqq (j + S_i + K_i) \mod 256$
    - Swap S_i and S_j
- PRGA (use S to get key-byte)
  - Update: $i = i + 1 \mod 256$ and $j = j + S_i \mod 256$
  - Swap S_i and S_j
  - Output S_r with $r = S_i + S_j \mod 256$

Outputting the key stream



Source: Wikipedia.org

RC4 PROBLEMS

- Ideally:
  - We want that the output bytes be uniformly random
  - Or at least, that they are indistinguishable from uniformly random, by a poly-time distinguisher

- Bias in some of the bits:
  - Probability that first two bytes are 0 is $2^{-16} + 2^{-32}$
  - More attacks were recently published by Paterson et al.
  - At the moment RC4 is discouraged by TLS/SSL (but because it’s efficient, it’s still being used a lot)

Block Ciphers

- Stream ciphers pad plaintext with PRG output
  - Principle usually follows OTP

- Block ciphers act like a symmetric encryption on plaintext blocks
  - Idea: plaintext is a string of n bits, e.g. 64, or 128
  - A good permutation of the bits makes the output look unrelated to the input

- Given key K and message M of size n:
  - Encryption Enc_K maps M to a ciphertext C
  - Decryption Dec_K maps ciphertext to plaintext

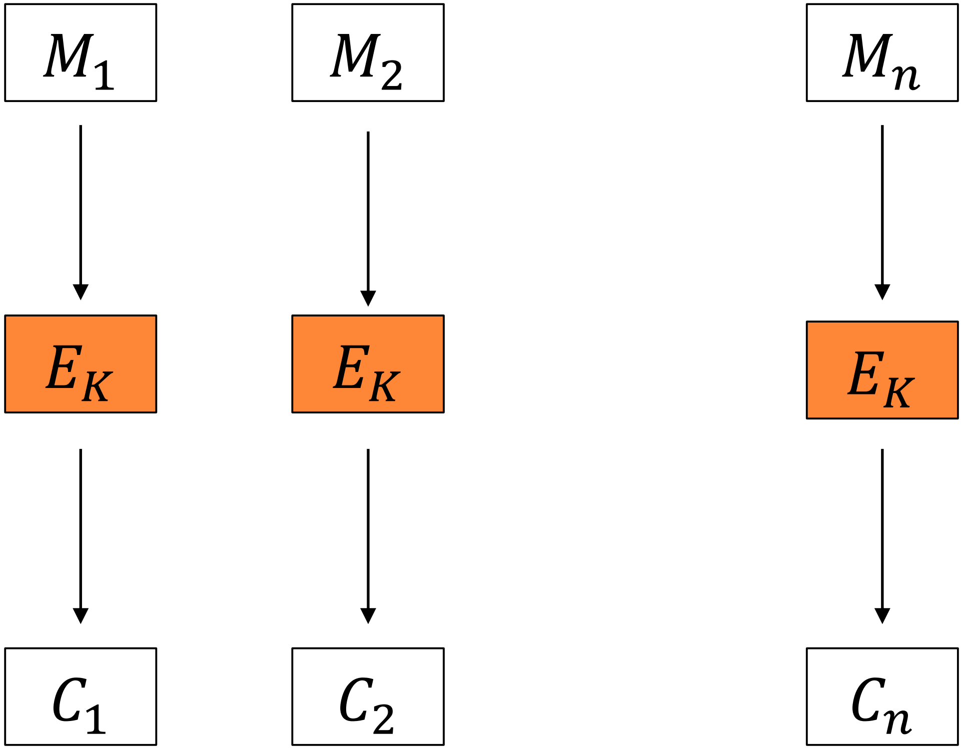
PERMUTATIONS AND PRPS

- Ideally:
  - Use a truly random permutation on the input domain
  - However, that means we need a key as large as the message

- In practice:
  - Use a pseudorandom permutation (PRP)
  - Then rely on indistinguishability of PRPs from RPs
  - The block cipher takes inputs of size n and returns output of same size
    - If we need to encrypt bigger texts, use one of several modes

ECB MODE

- Very simple: encrypt each block separately:



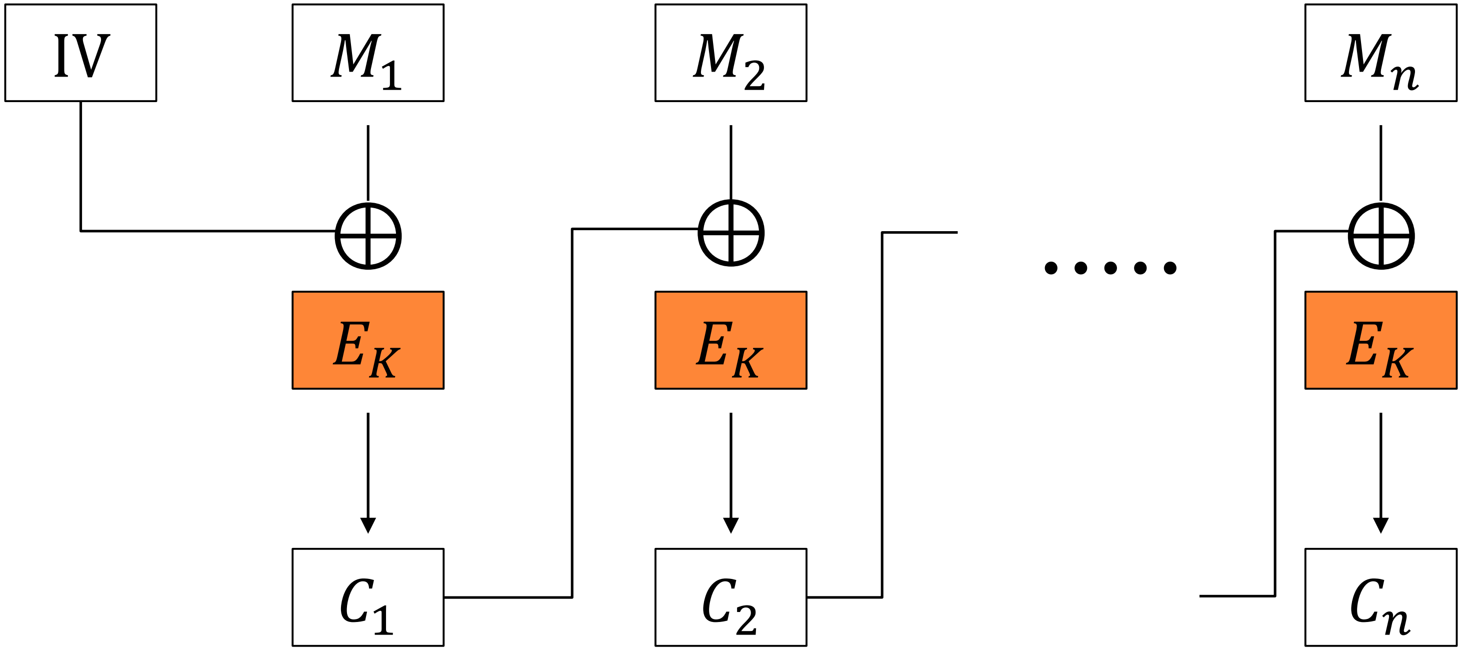
ECB Properties

- Advantages
  - Highly efficient and not harder to implement securely than the single-block encryption method
  - Parallelizable

- Security:
  - What happens if we have repetition in the input message? $(M_1, M_2 = M_1, M_3 \ldots)$
  - How about substitution/addition of message blocks?
  - Known for being insecure against active attackers

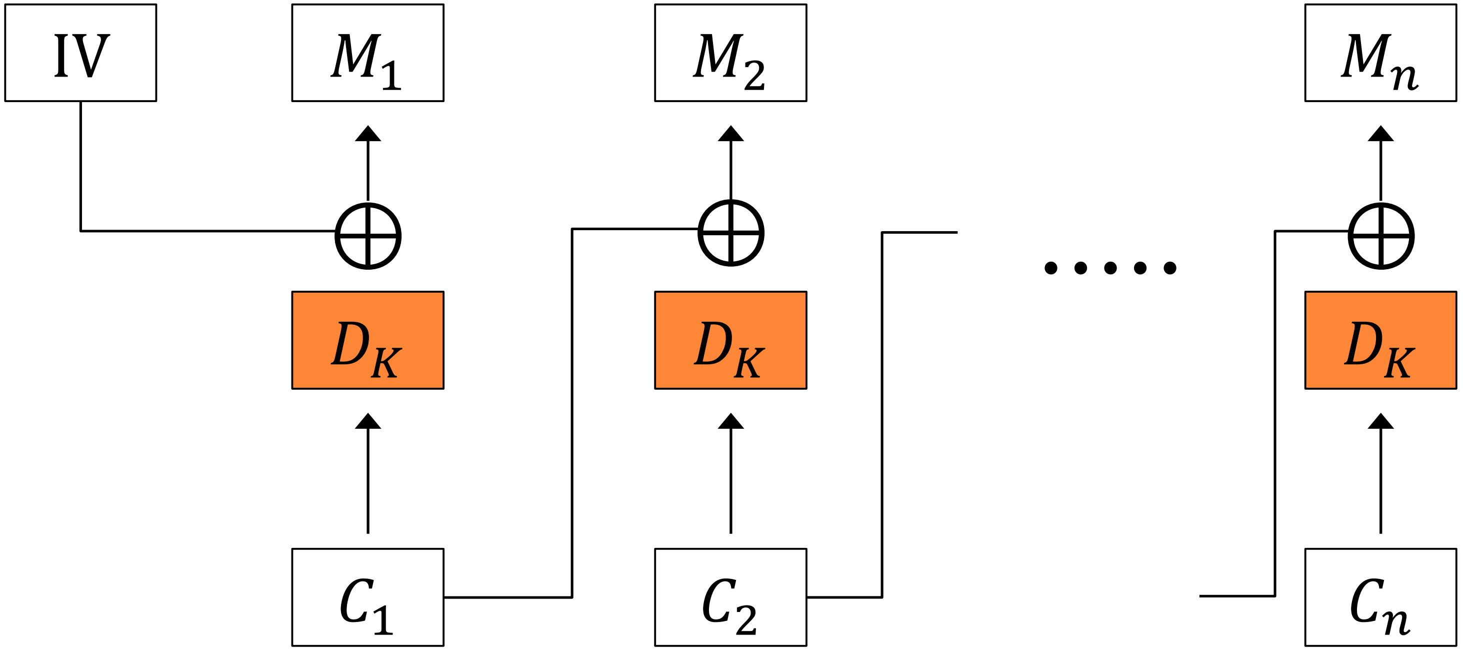
CBC MODE

- Link blocks together by using output blocks in the encryption of the following blocks
- An IV is used as a "seed", but can be sent in clear



CBC Properties

- Error handling:
  - Say one ciphertext block is corrupted
  - This only affects the decryption of the next block



CBC Security

- Not easy to insert messages
- Plaintext patterns (repetitions, etc.) not detectable

- The IV:
  - If IV is chosen uniformly at random and the encryption algorithm is a "good" permutation, then CBC encryption is a "good" encryption scheme
  - If IV is constant, CBC encryption does not hide prefixes

- You will often hear "do not use CBC modes in TLS/SSL". This is sound advice, but not because of weaknesses in the design of encryption

RECALL: GOOD SYMMETRIC ENCRYPTION

- $k \overset{\$}{\leftarrow} \mathrm{KGen}(1^{\gamma})$

  $b \overset{\$}{\leftarrow} \{0,1\}$

  $(\mathrm{m}_0, \mathrm{m}_1) \leftarrow A^{\mathrm{Enc}(\,)}(\gamma)$ with $|m_0| = |m_1|$

  $c \leftarrow \mathrm{Enc}(k, m_b)$

  $d \leftarrow A^{\mathrm{Enc}(\,)}(\gamma, c)$

  A wins iff $d = b$

- **(q, ϵ)-secure Symmetric Encryption**:

  A symmetric-key encryption scheme SEnc is (q, ϵ)-secure if, and only if, an adversary making at most q queries to Enc wins w.p. at most $\frac{1}{2} + \epsilon$

IND-CPA and Deterministic Encryption

- A generic IND-CPA attack:
  - $\mathcal{C}$ chooses K by running Key Generation
  - $\mathcal{A}$ picks M_0, M_1 and sends them to the Enc_K oracle:

    $$C_i \coloneqq \text{Enc}_K(M_i) \quad \text{for} \quad i = 0,1$$

  - $\mathcal{A}$ sends M_0, M_1 to $\mathcal{C}$, who encrypts M_b for $b \overset{\$}{\leftarrow} \{0,1\}$:

    $$\text{If } b = 0, \text{ then } C \coloneqq Enc_K(M_0)$$
    $$\text{Else, } C \coloneqq Enc_K(M_1) \qquad .$$

  - When $\mathcal{A}$ receives C, it compares it with $C_{0,}, C_1$, then returns $d = i$ if $C = C_i$; $i \in \{0,1\}$; else $\mathcal{A}$ sets $d \overset{\$}{\leftarrow} \{0,1\}$

- This always works if the encryption is deterministic. Why?

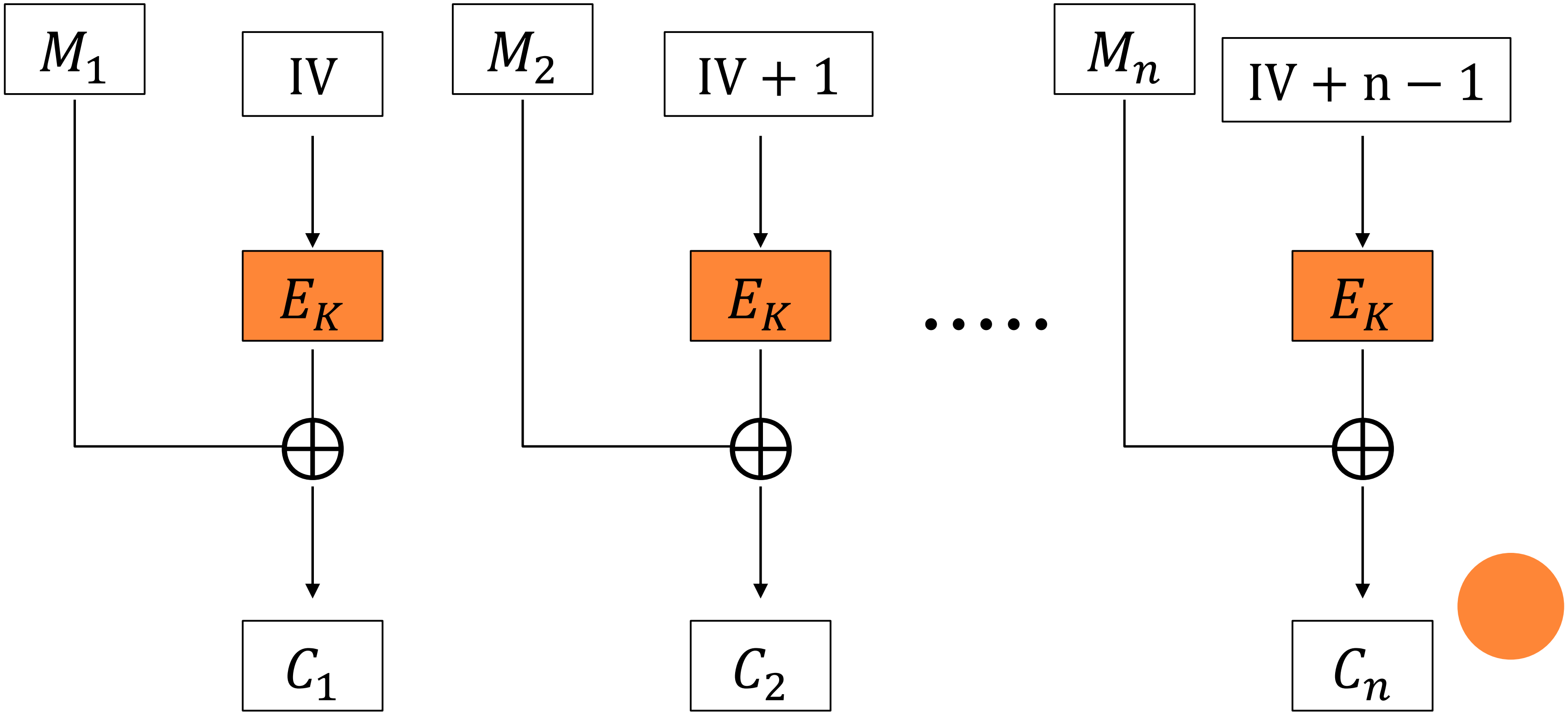
CBC WITH PREDICTALE IV

- Bug in TLS 1.0: IV for message M' is last ciphertext block of previous message M

- Attack:
  - First ask encryption of 0, receiving $(IV, \mathrm{Enc}_K(IV))$
  - Remember last ciphertext block, call it IV'
    - This is the IV for the next ciphertext
  - Submit $M_0 = IV \oplus IV'$ and a random M_1 to challenger
    - Now, if $b = 0$, then $\mathrm{Enc}_K\big(IV' \oplus (IV \oplus IV')\big) = \mathrm{Enc}_K(IV)$

CTR Mode Encryption

- Different IVs rather than a single one
- Parallelizable; IVs link ciphertext blocks together



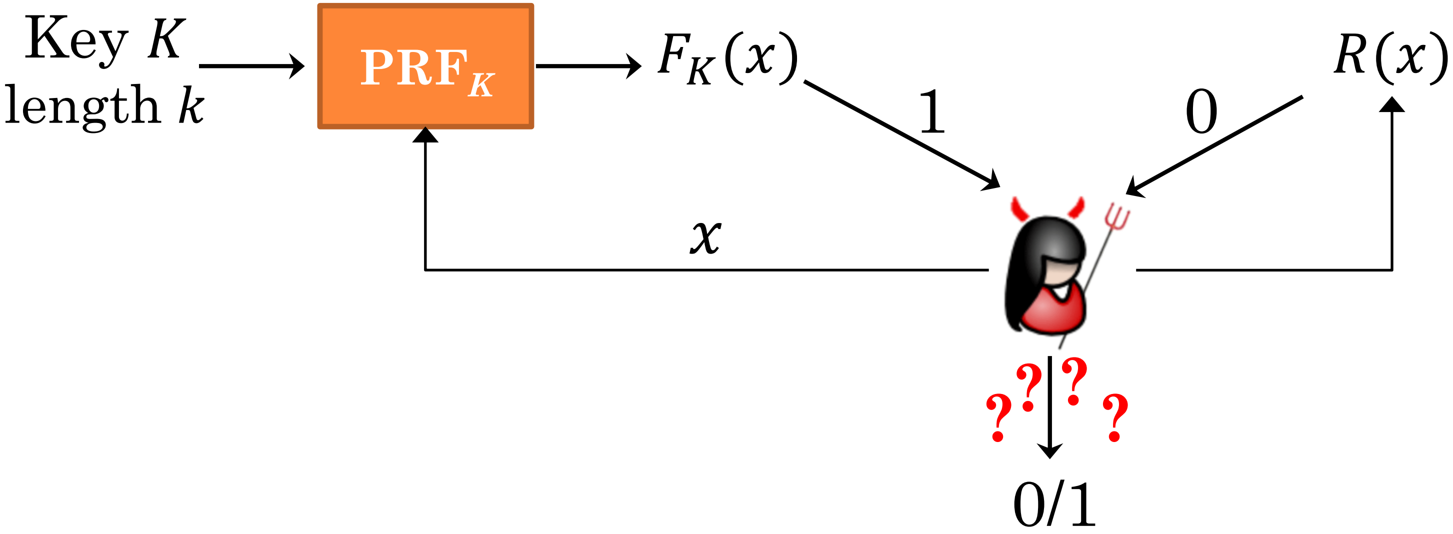
CTR Mode properties

- Efficiency and implementation:
  - Fully parallelizable once IV known
  - Some pre-processing can be done (such as encryption of all vectors from IV to IV+n-1)

- Security:
  - Note that this time, the length of IV need not be exactly equal to n
  - Hence, the symmetric encryption scheme is a function, rather than a permutation
  - In CTR mode, if encryption scheme is a PRF, then in CTR mode it has IND-CPA security

What is a PRF?

- Family of functions $F: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^m$
- First parameter is the key, chosen only once, so we regard the function as $F_k: \{0,1\}^n \rightarrow \{0,1\}^m$
- Notion of PRF (indistinguishability from random):



What is a PRF?

- Family of functions $F: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^m$
- First parameter is the key, chosen only once, so we regard the function as $F_k: \{0,1\}^n \rightarrow \{0,1\}^m$
- Notion of PRF (indistinguishability from random):

$$k \overset{\$}{\leftarrow} \{0,1\}^k$$

$$d \leftarrow \mathcal{A}^{G_b(*)}$$

$\mathcal{A}$ wins iff. $d = b$

$G_b(x)$

If $b = 0$, return $R(x)$
Else, return $F_K(x)$

  - (k, ε)-PR-ness: k queries to G_b, A wins w.p. at most $\frac{1}{2} + \varepsilon$

PRFs and PRPs

- For a keyed function $F_K\colon \{0,1\}^n \rightarrow \{0,1\}^n$,we may also speak of permutations
  - Permutation: domain and range are the same
  - Bijection: F_K is keyed permutation if for all K, F_K is 1-to-1 (bijective; thus invertible)

- Pseudo-random permutation:
  - Keyed Permutation
  - Indistinguishability from a random permutation: akin to PRF game, but with equal domain/range, and the bijective property

IND-CPA security from PRF

- Assumption:
  - Use PR function $F: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$
  - Choose secret key K of length k as output of Kgen
  - Both encryptor and decryptor know F and key K

- Encryption of some message $M \in \{0,1\}^n$:
  - Pick random number $r \overset{\$}{\leftarrow} \{0,1\}^n$
  - Encrypt M to $(r; M \oplus F_K(r))$

- Decryption of ciphertext $C = (C_1;\ C_2)$:
  - Decrypt C to $\widehat{M} \coloneqq\ C_2 \oplus F_K(C_1)$

Security of This Construction

- IND-CPA security:
  - For any adversary $\mathcal{A}$ against the IND-CPA security of the encryption scheme, making k queries to the encryption oracle and winning w.p. $\frac{1}{2} + \varepsilon_A$ …
  - … There exists an adversary B against the pseudo-randomness of the function F, which makes k queries to its generation oracle, and wins with probability:

$$P_b \geq \frac{1}{2} + \varepsilon_A + \frac{k}{2^n}$$

Why the additional term?

- Proof: in TDs

Message Authentication Codes (MACs)

Unforgeability and MACs

- Message Authentication Codes prove message integrity and indicate its provenance (sender)

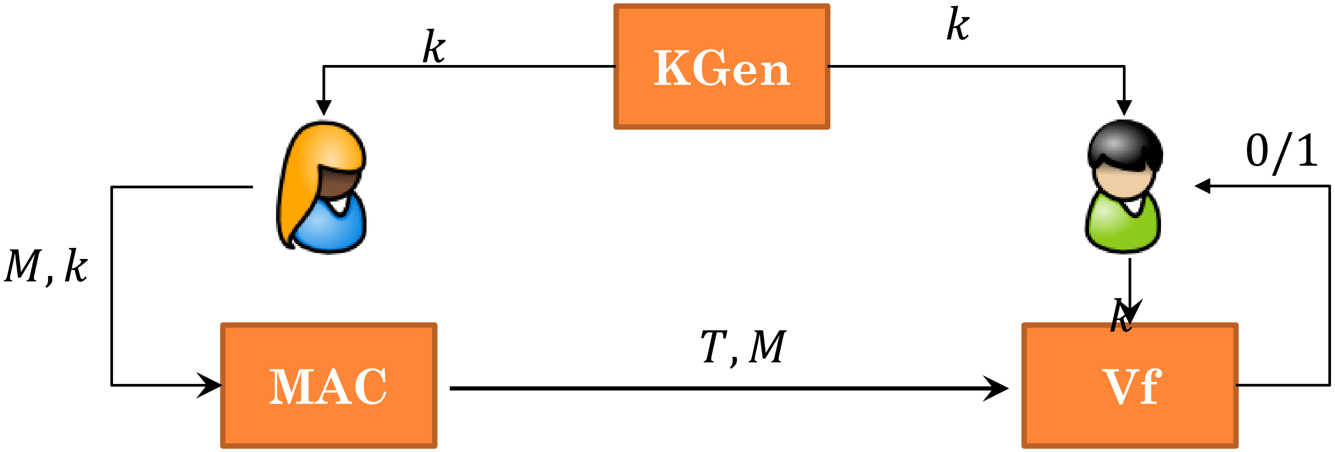
- MACs do not hide the message they authenticate
  - Quite the opposite: often you would send M along

- MACs do not entirely hide the key either
  - They can reveal a part of the key, as long as it is still hard to recover the other part (say a half)

- Their purpose is to authenticate, not to hide

MAC Scheme syntax

- Tuple of algorithms (KGen, MAC, Vf) s.t.:
  - $\mathrm{KGen}(1^{\gamma})$ outputs symmetric key k
  - MAC(k, M) outputs tag T for message M
  - Vf(k, M, T) outputs 1 if T verifies for M and 0 otherwise



- Correctness (of MAC and Vf)
  - For any K, M, if $T = \mathrm{MAC}(K; M)$, it holds $\mathrm{Vf}(K; M, T) = 1$

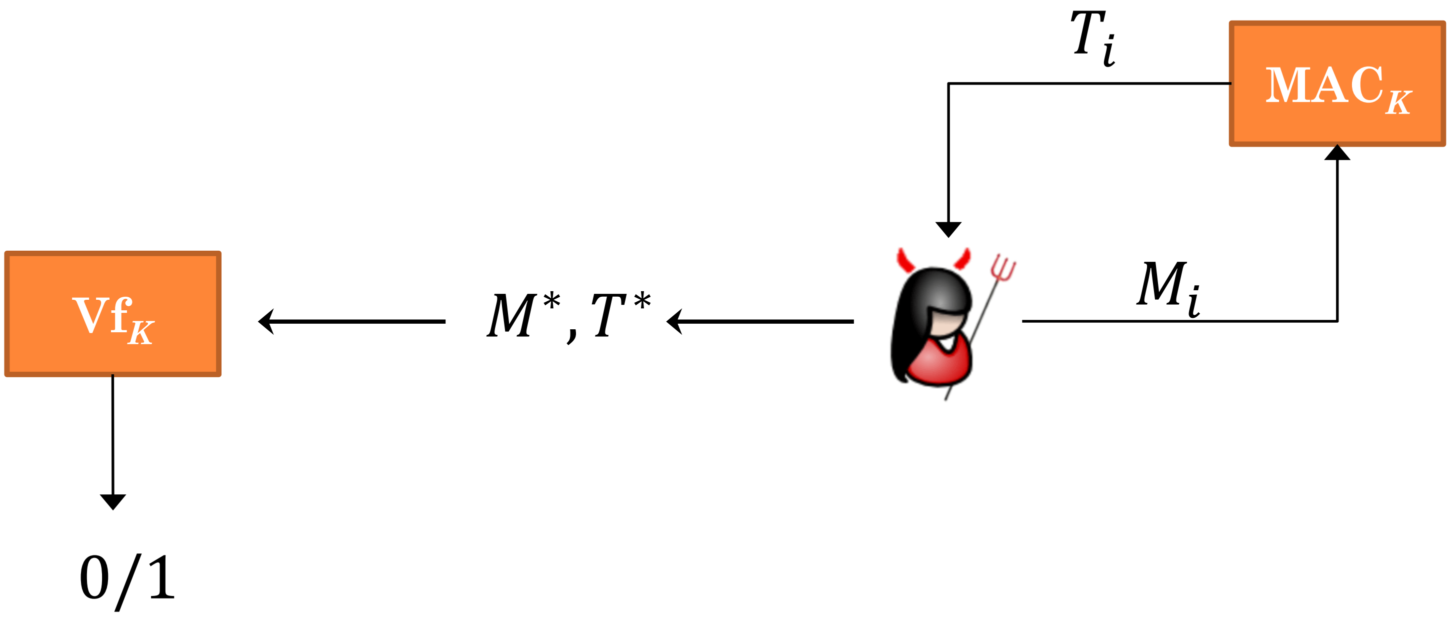
MAC security intuition

- How do we use a MAC?
  - Assume Alice sends message and MAC to Bob
    - Say message is unencrypted, an update or a file
  - An adversary may intercept, change, or replace it
  - Bob receives the message and the MAC
  - Bob verifies the MAC. Ideally:
    - If the MAC verifies: it's Alice's untampered message
    - If the MAC verification fails: the message was tampered with

- A MAC cannot be forged for a new message
  - But using an old (M, T)-tuple will lead to verification

The Unforgeability Game

- Not real/random indistinguishability this time
- Unforgeability of fresh messages:



- Adv. wins iff. $M^* \notin \{M_1, \dots, M_n\}$ and $\text{Vf}(K; M^*, T^*) = 1$

Game Description

- A plays the game against challenger
  - First, challenger generates key, but keeps it private
  - A can query a MAC oracle on messages m
    - The challenger uses MAC(k, m) to return output
  - Finally, A returns tuple (m*, T*)
- A wins iff. Vf(k, m*,T*) = 1 and m* not queried to Sign

Exercise: try to write this def. in game form!

Unforgeability in game notation

- Existential Unforgeability against Chosen Message Attacks – EUF-CMA:

$$K \xleftarrow{\$} \mathrm{KGen}(1^{\lambda})$$

$$(M^*, T^*) \leftarrow \mathcal{A}^{MAC_K(*)}$$

$$\overline{}$$

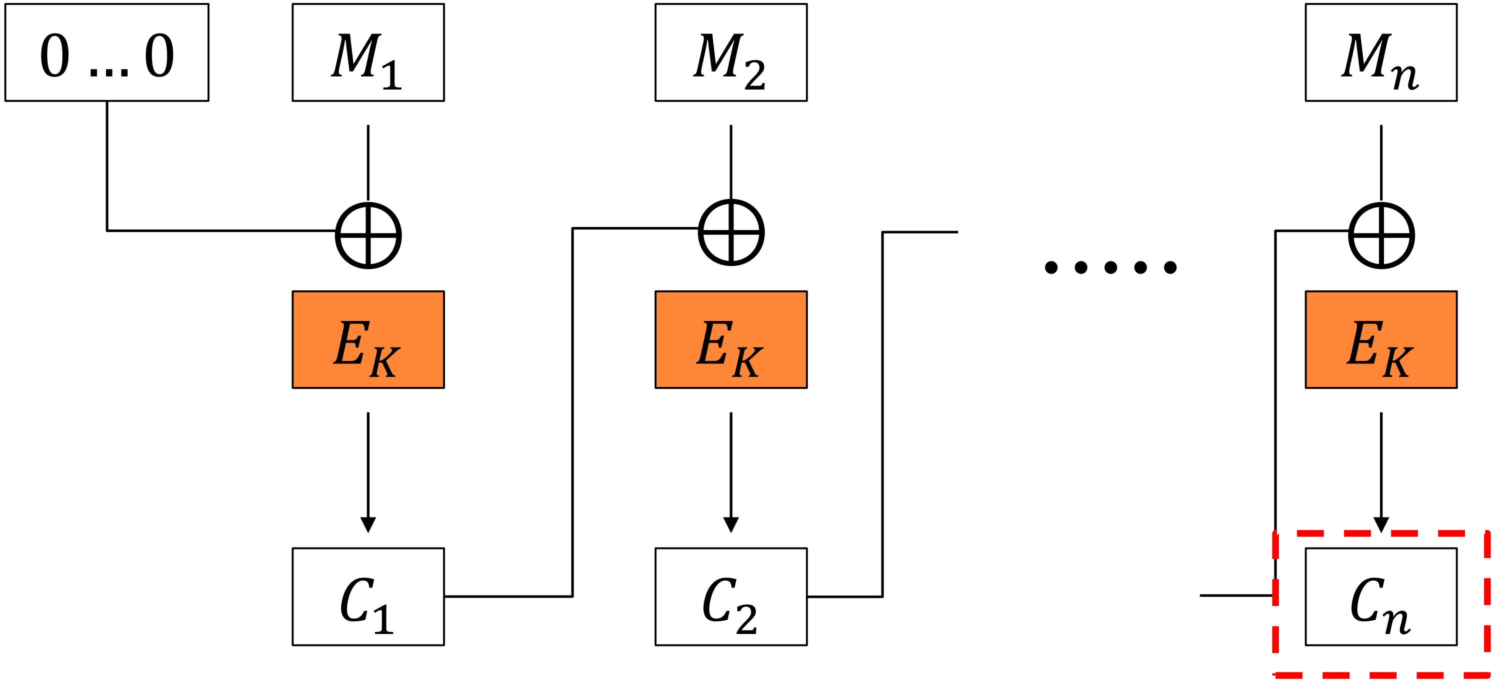
$\mathcal{A}$ wins iff. M^* not queried to $\mathrm{MAC}_K(M^*)$

$$\mathrm{Vf_K(M^*, T^*) = 1}$$

- Trivial attacks:
  - A could just guess a correct tag, or a correct key
  - The probability is $2^{|\mathrm{MAC}_K(*)|} + 2^{|\mathrm{KSpace}|}$
  - Goal: make that probability negligible in λ
- (k, ε)-security: $\mathcal{A}$ with k MAC queries wins w.p. ε

Constructing MACs

- Two ways of doing it:
  - Using block ciphers
  - Based on hash functions (which we will see later)
- CBC-MAC:



CBC-MAC AND ITS SECURITY

- If the block cipher E_K is a PRP, then:
  - If we consider only messages of a fixed length, we can prove CBC-MAC is a PRF (no proof here)
  - Any MAC scheme that is a PRF is unforgeable (but not the reverse). Proof in TDs

- However, if we can allow messages of ANY length, we can play on prefixes to get a forgery

A Prefix-Based Attack

- Ask for the MAC of some 1-block message M_1:

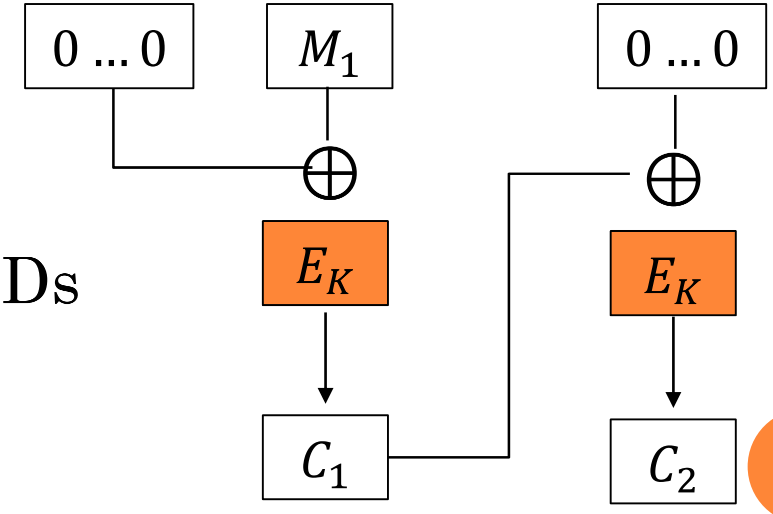
$$C_1 = E_K(0 \oplus M_1)$$

- Then ask for the MAC of this ciphertext:

$$C_2 = E_K(0 \oplus C_1)$$

- Look at MAC of $M_1 | \mathbf{0}$
  - Collision: C_1 and $M_1 | \mathbf{0}$

- Generalization of attack: TDs



MACs for variable lengths

- Problem is that MAC of messages of any lengths is of length 1 block exactly (last c-text block)
  - We get collisions of messages of variable length

- Obvious solution: authenticate the length, too.

- Option 1: if length n is known: $\mathrm{MAC}(K; n, M_1, \dots, M_n)$
  - In theory, perfect; in practice, Vaudenay attacks
- Option 2: length unknown, 1 key: $\mathrm{MAC}(K; M_1, \dots, M_n, n)$
  - Broken in 1984
- Option 3: use 2 keys: $E_{K'}(\mathrm{MAC}_K(M_1, \dots, M_n))$

Hash functions and MACs

Hash Functions

- Another way to build MACs (will see later)

- What is a hash function?
  - Function $f\colon \{0,1\}^* \to \{0,1\}^n$ with variable-length input and fixed-length output
  - Inevitably, this means collisions. Why?
  - Ideally not many, and hard to find

Security of Hash Functions

- Weak collision resistance: for any $x \in \{0,1\}^*$ it is hard to find $x' \neq x$ such that $h(x') = h(x)$
  - For any x (**universal**) there exists no adversary $\mathcal{A}$ which, given x and access to h, can output such an x' with non-negligible probability
  - **Average**: for $x \overset{\$}{\leftarrow} \{0,1\}^*$, there exists no adversary $\mathcal{A}$ which, given x and access to h, can output such an x' with non-negligible probability

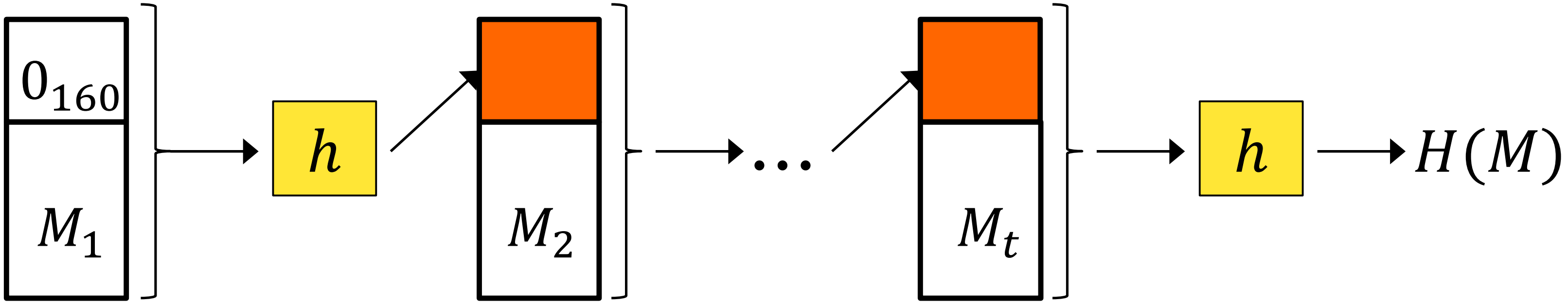
- Strong collision resistance: it is hard to find any pair $x, x' \neq x$ such that $h(x) \neq h(x')$
  - In general, easier to find than for fixed x

Finding Collisions

- The birthday paradox:
  - Probability 1 in 23 people have the same bday as Henri Poincaré (April 29th) : 23/365
  - Probability that 2 people in 23 have the same birthday : $\sum_{i=1}^{365} \binom{365}{2} \left(\frac{1}{365}\right)^2$, which gives about $\frac{1}{2}$
- What does this mean for us?
  - First case: similar to weak collision resistance
  - Second case: similar to strong collision resistance

Merkle Damgaard

- Arbitrary-length input from fixed-length input hash function
- Say $h: \{0,1\}^{512} \rightarrow \{0,1\}^{160}$ (standard input and output sizes)
  - Want to extend it to $H: \{0,1\}^* \rightarrow \{0,1\}^{160}$
  - How do we do this?
- MD: kind of CBC-mode extension
  - $M = M_1 \ldots M_t$ with length of M_i equal to 512-160 = 352



Security of this Construction

- Theorem:
  - For any adversary $\mathcal{A}$ that can find, with non negligible probability $p_{\mathcal{A}}$, a collision $M, M' \neq M$ such that $H(M) = H(M')$ …
  - … There exists an adversary $\mathcal{B}$ that can find messages $m, m' \neq m$ with $h(m) = h(m')$ with non-negligible probability $p_{\mathcal{B}}$

- Conclusion: as long as h is collision-resistant, H is also collision-resistant

Collisions and Collisions…

- First signs of weakness:
  - Partial collisions, or collisions only in latter stages of the bigger H function
- Further weaknesses:
  - First true collisions appear, but they are heavily contrived: it’s a strong collision-resistance attack
    - While valid they fail to convince users that this means in a short time the hash function will be broken
- Hash function is “broken”:
  - We get collisions on chosen messages: given certificate M, we find certificate $M' = M$ s.t. $H(M) = H(M')$

MACs from Hash Functions

- To key or not to key: MACs use keys, hashes do not

- From no-key to keys:
  - First idea: hash key, then message (key for authentication, m for integrity): problem is something similar to CBC prefix problem for Merkle Damgaard
  - Second idea: hash message, then key (now message is variable prefix, rather than the constant *k*): can do birthday attack on MAC to find collision in hash function *h*
  - Better solution: use something like HMAC

HMAC

- Given key K, message m, hash function h
  - Also take 2 fixed, known 64-bit strings: $\mathrm{pad}_{\mathrm{in}}, \mathrm{pad}_{\mathrm{out}}$
  - Key K of 64 bits – or padded to that length if necessary

- HMAC is defined then as:
  - $\mathrm{MAC}_K(m) \ \coloneqq h(K \oplus \mathrm{pad}_{\mathrm{out}} \ , \ h(K \oplus \mathrm{pad}_{\mathrm{in}}, m))$

- There exists a proof (which we will not cover here), that says that if HMAC is insecure, then:
  - h is not collision resistant; or
  - The output of h is “predictable”

Unforgeability, PRF, PRP

- HMACs must only offer unforgeability
- However, the use of the hash function gives more security than just unforgeability

- Pseudorandomness vs. Unforgeability
  - (Keyed) Pseudorandomness (PRP, PRF), always implies unforgeability
  - However, one can have an unforgeable scheme whose output is not indistinguishable from random

What we Learned Today

CIPHERS

- Stream ciphers
  - Most of them rely on OTP + PRG paradigm
  - RC4 is very efficient, but biased and in fact insecure
- Block ciphers
  - Ideally a PRP of a message of a specific length
  - Can be extended to longer messages by using modes
    - ECB is bad, CBC is average, CTR seems best
  - Ideally they are PRFs

Message Authentication Codes

- MACs provide a proof of integrity and authentication of sender, by means of a shared key

- Security: MACs should be existentially unforgeable under chosen ciphertext attacks (EUF-CMA)

- Constructions:
  - Based on block ciphers
  - Using hash functions

Hash Functions

- Take input of varying length and outputs fixed-length strings

- Hash functions must be collision-resistant
  - Weak CR: given x, find x' with $H(x) = H(x')$
  - Strong CR: finx x, x' with $H(x) = H(x')$
  - Can be extended from smaller compression functions to larger hash functions using Merkle Damgaard

- HMAC:
  - Uses hash function twice, with outer and inner pad functions