

# 共通鍵暗号1

光成滋生

last update: 2025/10/23

## 目的

- 共通鍵暗号の基本的な考え方
- ワンタイムパッドと情報理論的安全性の考え方
- 代表的な共通鍵暗号のアルゴリズムの概略
- 計算量の評価について
- 暗号化の安全性の種類
- 数学的な定式化も紹介する

# 目次

## 用語一覧

- 共通鍵暗号, 乱数, 排他的論理和 XOR (exclusive or)
- ワンタイムパッド OTP (One-Time Pad), 情報理論的安全性
- PRG, PRF, ストリーム暗号, ChaCha20
- ブロック暗号, AES, 暗号利用モード, ECB, CBC, CTR
- 確率的アルゴリズム, 計算量, O記法, クラス P, クラス NP
- 暗号の安全性, 強秘匿性, 頑強性

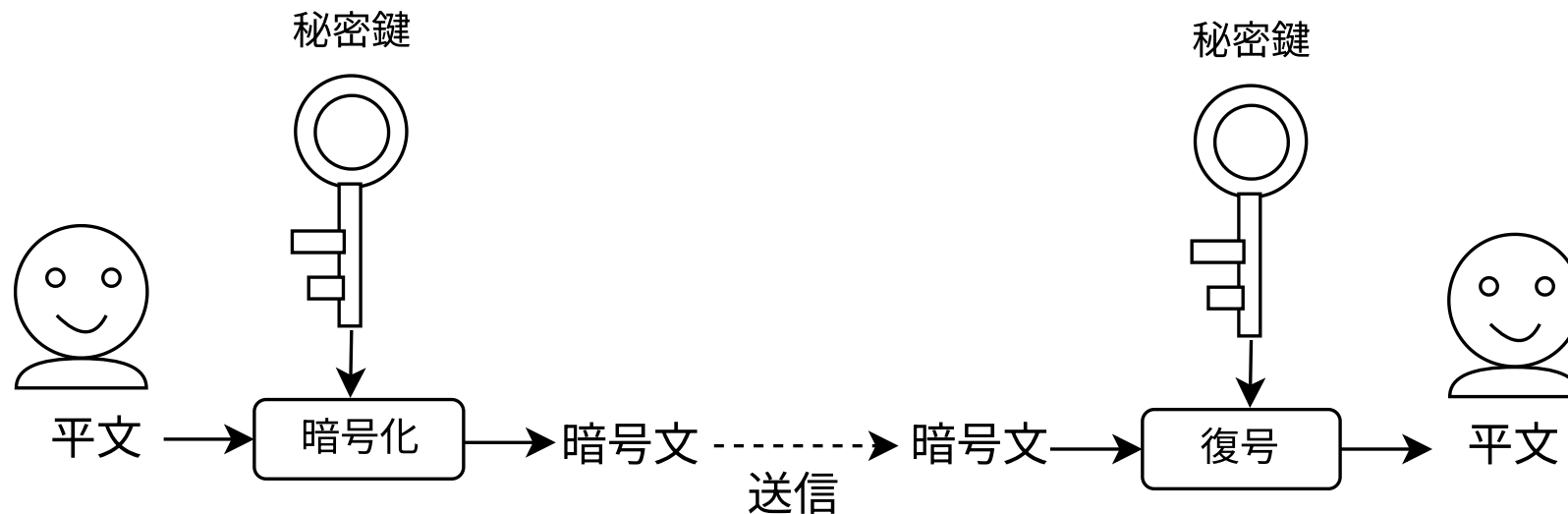
## 後半

- MAC, 存在的偽造不可能
- IND-CPA安全, IND-CCA2安全, Enc-then-Mac
- ハッシュ関数, SHA-2, 伸長攻撃, SHA-3, XOF
- AEAD, 拡大体, 軽量暗号, Ascon

# 共通鍵暗号とは

## 「暗号鍵＝復号鍵＝秘密鍵」な暗号方式

- 秘密鍵暗号, 対称鍵暗号ともいう
- 秘密鍵は事前に二者間で安全に共有しておく必要がある
- 秘密鍵  $s$  で平文  $m$  を暗号化して暗号文  $c$  を得る:  $c = Enc(s, m)$
- 秘密鍵  $s$  で暗号文  $c$  を復号して平文  $m$  を得る:  $m = Dec(s, c)$ 
  - 暗号化して復号したら元に戻る必要がある:  $Dec(s, Enc(s, m)) = m$



- 秘密鍵: secret key, 暗号化: encrypt, 復号: decrypt, 平文: message, 暗号文: ciphertext

# 共通鍵暗号の種類

## ストリーム暗号

- ノイズ（乱数）を生成し、平文とビット単位で混ぜ合わせて暗号化する
  - 主な方式: ChaCha20
- ストリーム暗号 = 乱数 + 排他的論理和

## ブロック暗号

- 平文を一定の固まり（ブロック）ごとに分割し、ブロック単位で暗号化する
- 主な方式: AES

## でたらめな数の列

- 「でたらめ」とは?
- それまでに出力されている数の列を見ても次に出る数が予測できないこと
  - 次に出る値が過去の履歴と独立
  - 理想的なコインの裏表の出方
  - 裏が出たら0, 表が出たら1とすると次に出る数の確率は0か1が1/2ずつ

## でたらめではない例

- 010101010...
  - 0と1が交互（ただしコインを振って偶然交互に出る可能性はある）

# 排他的論理和 XOR (exclusive or)

## 定義

- 1bit変数  $a$  と  $b$  の排他的論理和（以降xorと略す）  $a \oplus b$  は次の表で定義される

$a$	0	0	1	1
$b$	0	1	0	1
$a \oplus b$	0	1	1	0

## 性質

- $a \oplus b = b \oplus a$ （交換法則）
- $a \oplus 0 = a$ （単位元）
- $a \oplus 1 = \neg a$ （否定）
- $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ （結合法則）
- $(a \oplus b) \oplus b = a$ （同じ値を2回作用させると元に戻る）

# ワンタイムパッド OTP (One-Time Pad)

## 乱数とxorを組み合わせた暗号

- $n$  bitの平文  $m$  に対して  $n$  bitの乱数  $s$  を用意する
  - $s$  が秘密鍵
- 暗号文はビットごとに  $m$  と  $s$  のxorをとり暗号文  $c$  を作る  
それをまとめて  $c = \text{Enc}(s, m) = m \oplus s$  と書く
- 復号はビットごとに暗号文  $c$  と乱数  $s$  のxorをとる  
 $\text{Dec}(s, c) = m$ 
  - xorの性質により  $\text{Dec}(s, \text{Enc}(s, m)) = (m \oplus s) \oplus s = m$

## 特徴

- 平文と秘密鍵の大きさが同じ
- 情報理論的安全性（次のスライド）を持つ



# 情報理論的安全性

## 絶対に破れない暗号?

- $n = 1$  のとき暗号文  $c$  は 0 or 1

秘密鍵  $s$  とそのときの平文  $m$  の組み合わせは

秘密鍵 \ 暗号文	0	1
0	0	1
1	1	0

- 平文  $m$  は確率  $1/2$  で 0 or 1: これは暗号文  $c$  を知らなくても同じ
- 一般の  $n$  bit暗号文のとき  $c$  は 000...0 から 111...1 の  $2^n$  通り
- 秘密鍵  $s$  も 000...0 から 111...1 の  $2^n$  通り
- 平文も 000...0 から 111...1 の  $2^n$  通りでどれも同じ確率  $1/2^n$ 
  - やはり  $c$  を知らなくても同じ.  $c$  の情報が  $m$  の推測に役に立たないので安全という
  - $1/2^n$  の確率でたまたま当たることはある

# 欠点

## 一度しか使えない

- One-Time Pad = 一度きりの便箋
- 同じ秘密鍵で複数回使うと安全性が保てない
  - $c_1 = m_1 \oplus s, c_2 = m_2 \oplus s$  とする（単純化のために1bitで考える）と
  - $c_1 = c_2$  なら  $m_1 = m_2, c_1 \neq c_2$  なら  $m_1 \neq m_2$ : 平文の情報が漏れる

## 平文のサイズ＝秘密鍵のサイズ

- 平文  $m$  が1GiBのデータなら秘密鍵  $s$  も1GiB必要
  - $s$  を安全に共有できるのならその方法で  $m$  を共有すればよいのでは

## ビット反転に弱い

- 暗号文の特定のビットを反転させると対応する平文のビットが反転する
  - 中身は分からなくても平文を制御できる可能性（完全性は別の方法で担保する）

## 秘密鍵の転送

- 64TiBのHDDデータを東京から大阪に転送したい
  - 秘密鍵を安全に送るには?
  - 10Gbpsの専用線で送るとどれぐらい時間がかかる?

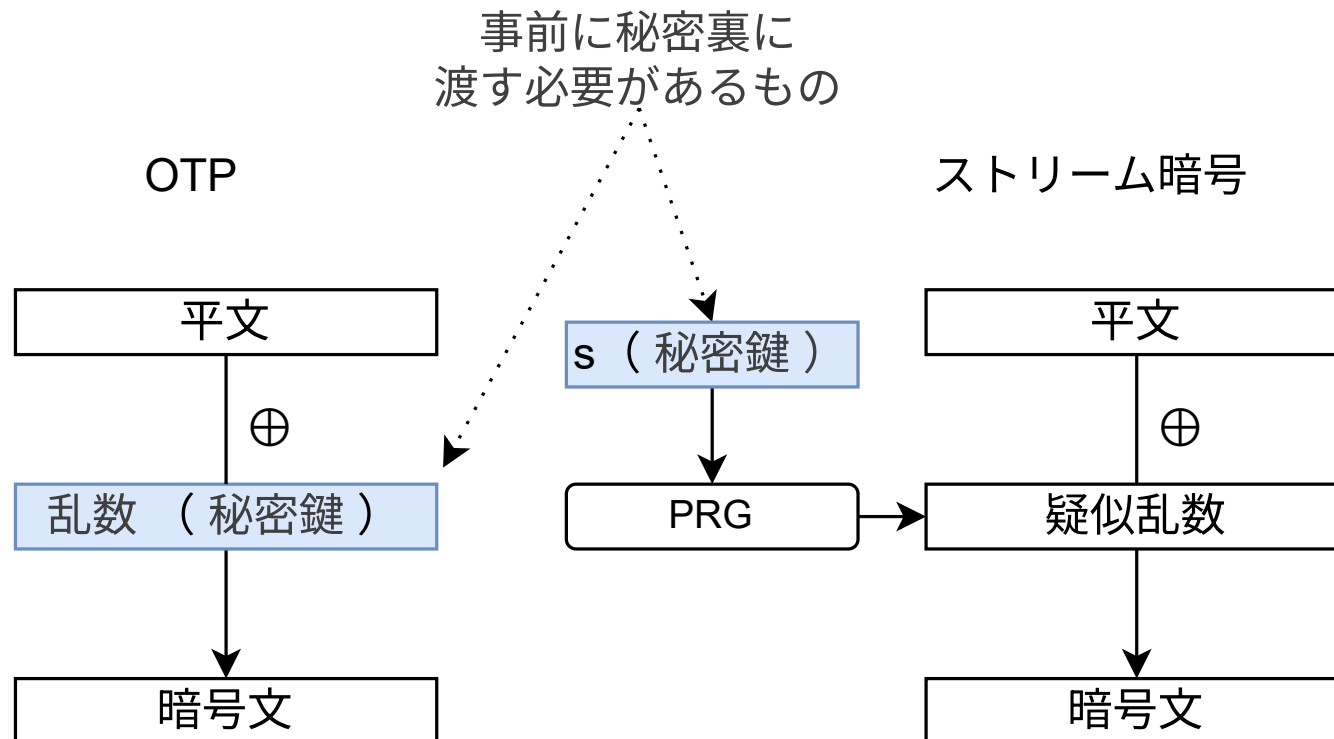
## 秘密鍵の2回利用

- 年齢を1バイトの整数値としてOTPによる暗号化を考える
  - 子供の年齢の暗号文が0x1eだったとき, 同じ秘密鍵を利用した大人の暗号文が0x5eだった
  - 大人の年齢は何歳以上?

# 疑似乱数生成器 PRG (Pseudo-Random Generator)

## OTPの秘密鍵を小さくしたい

- PRGは「小さい情報」から「大きな疑似乱数」を作る決定的アルゴリズム
- 情報理論的安全性は無くなる: 代わりに計算量的安全性を考える (後の講義)
  - 決定的 = 入力と同じならいつも同じ出力をするアルゴリズム
  - 「小さい情報」 =  $s$  = 秘密鍵

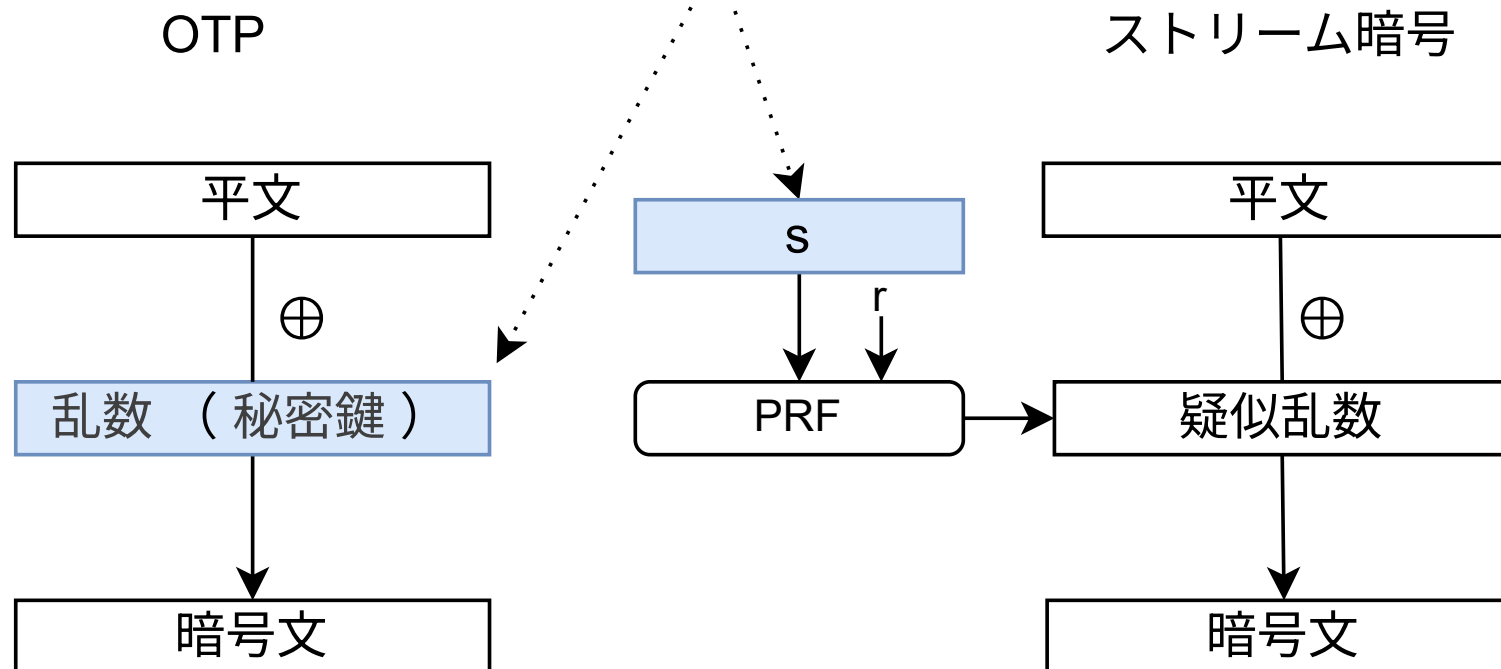


# 疑似ランダム関数 PRF (Pseudo-Random Function)

## 秘密鍵を繰り返し使いたい

- 「小さい情報」  $s$  と「入力」  $r$  を与えると疑似乱数を出力する決定的アルゴリズム  $F(s, r)$ 
  - $r$  は秘密でなくてよいが2回同じものを使ってはいけない
- ストリーム暗号  $Enc(s, m) := (r, F(s, r) \oplus m)$ , ここで  $r$  はランダムに選んだ値

事前に秘密裏に  
渡す必要があるもの



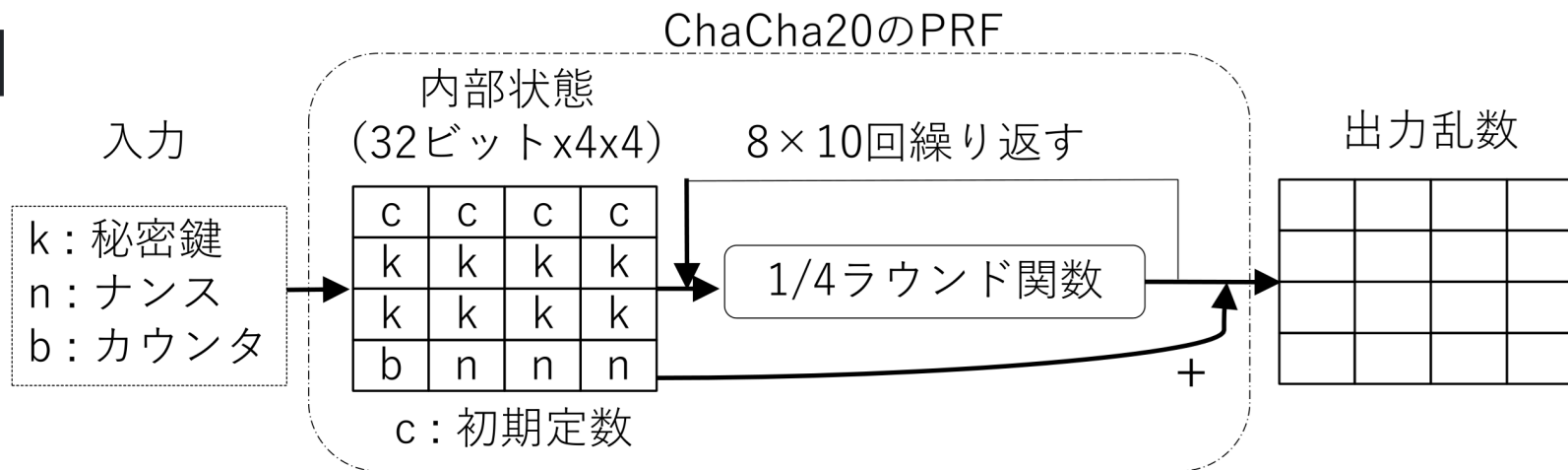
# ストリーム暗号の例

## ChaCha20

- 入力: 256bitの秘密鍵  $k$  と96bitのナンス  $n$ 
  - ナンス (nonce): 一度だけ使われる値
  - ナンスが64bitは小さい, 96bitだとAES-GCMと同じ長さで扱いやすい
- 出力: 32bitのカウンタ  $b$  1個につき512bitの乱数
- $b$  を1から始めて1ずつ増やして疑似乱数を生成
- 平文と排他的論理和をとり暗号文とする

# ChaCha20のPRF

## 全体図



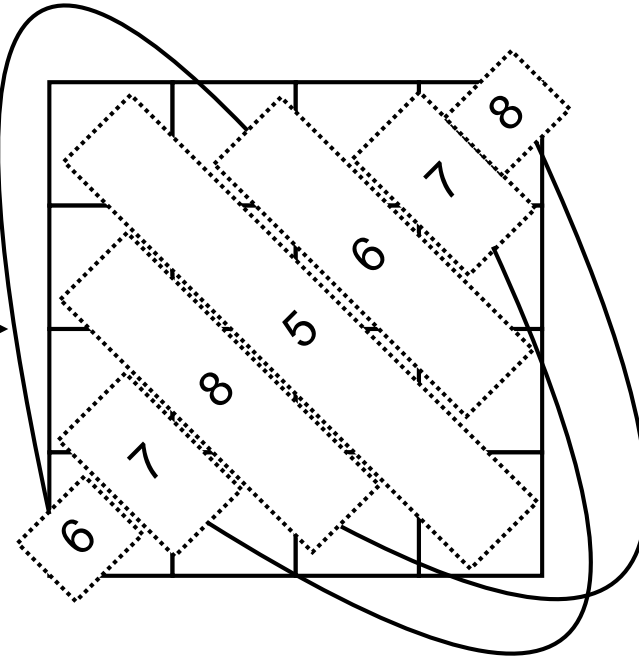
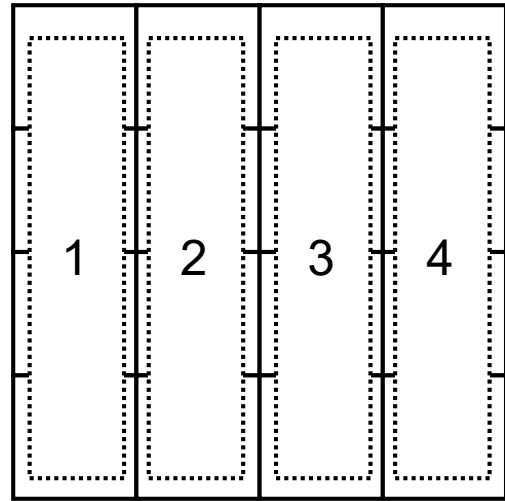
- 1/4ラウンド関数  $QR(a, b, c, d)$  :  $a, b, c, d$  は32bit整数
- $rotLeft(x, n)$  は  $x$  を左に  $n$  bit回転させる  
 $x=[H:L]$  ,  $H:n$  bit,  $L:(32-n)$  bitのとき  $rotLeft(x, n)=[L:H]$

```
def QR(a, b, c, d):
    a ← a + b; d ← d ⊕ a; d ← rotLeft(d, 16);
    c ← c + d; b ← b ⊕ c; b ← rotLeft(b, 12);
    a ← a + b; d ← d ⊕ a; d ← rotLeft(d, 8);
    c ← c + d; b ← b ⊕ c; b ← rotLeft(b, 7);
    return (a, b, c, d)
```

# 1/4ラウンド関数QRの適用方法

## 512bitの内部状態を32bit×4×4の正方形に並べる

- |          |          |          |          |
|----------|----------|----------|----------|
| $x_0$    | $x_1$    | $x_2$    | $x_3$    |
| $x_4$    | $x_5$    | $x_6$    | $x_7$    |
| $x_8$    | $x_9$    | $x_{10}$ | $x_{11}$ |
| $x_{12}$ | $x_{13}$ | $x_{14}$ | $x_{15}$ |



- 縦ライン1~4, 斜めライン5~8の4個の  $x_i$  に対して QR() を適用 (合計8回)
- これを10回繰り返す



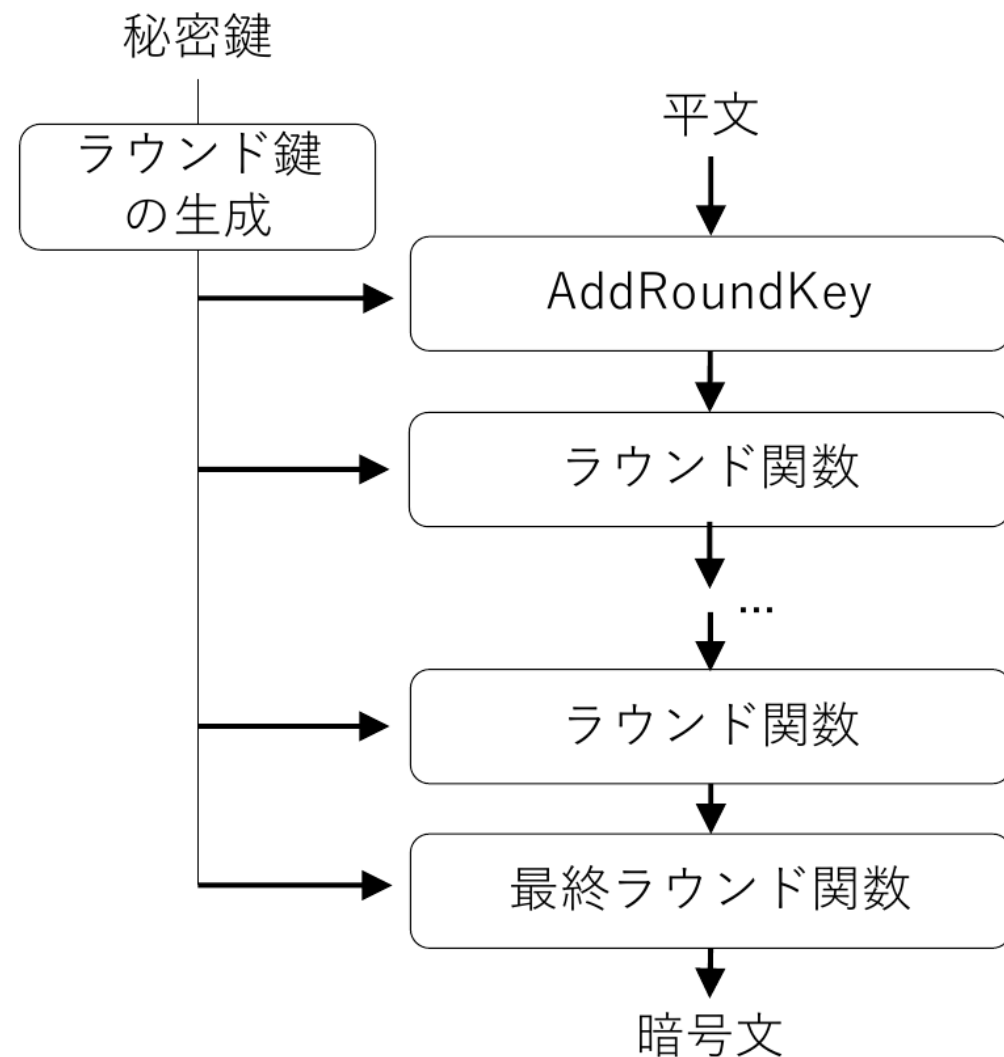
# AES (Advanced Encryption Standard)

## 最も広く使われているブロック暗号

- ブロックの単位は128bit
- 秘密鍵のサイズは128, 192, 256bitのいずれか

## 暗号化方法

- 秘密鍵からラウンド鍵を生成
- AddRoundKey (1回)
- ラウンド関数 (128bitなら9回)
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey
- 最終ラウンド関数 (1回)



# AddRoundKeyとSubBytes

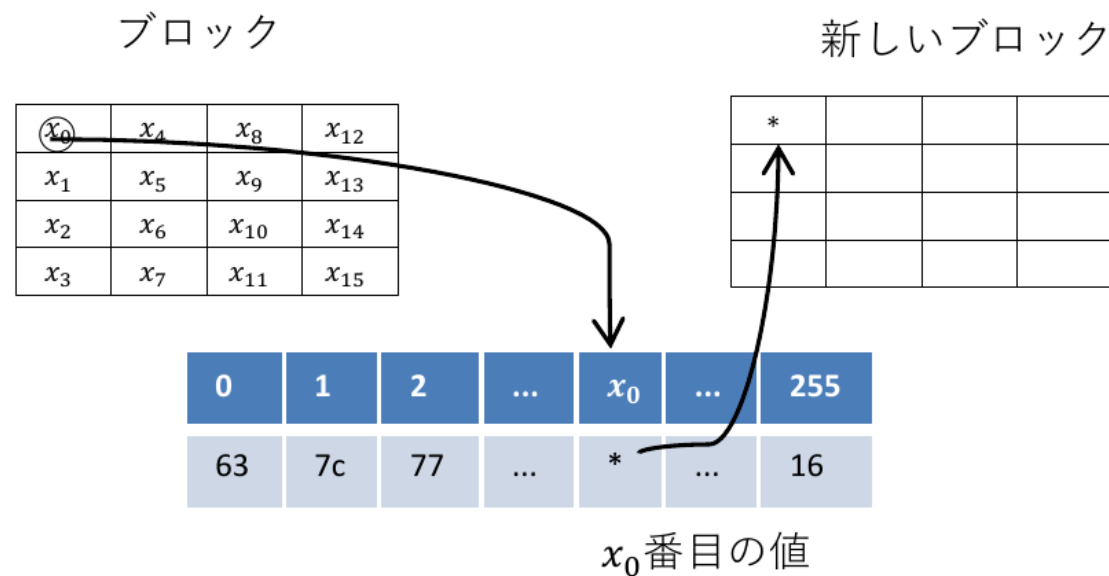
## AddRoundKey

- ラウンド鍵との排他的論理和
- 128bitのブロックを8bit×4×4の正方形に並べる



## SubBytes

- 各8bitの値をindexとするテーブル引き
- テーブル (S-Box) の値は固定
  - 0 番目の  $x_0$  が1ならS-Boxのテーブル1番目の値 0x7c になる
  - $i$  番目は  $\text{Table}[x_i]$  になる



# ShiftRowとMixColumns

## ShiftRow

- データをずらす

## MixColumns

- 行列の掛け算
- $x'_0 = A_{00}x_0 + A_{01}x_1 + A_{02}x_2 + A_{03}x_3$
- 掛け算や足し算は有限体（後の講義）を利用

ブロック

$x_0$	$x_4$	$x_8$	$x_{12}$
$x_1$	$x_5$	$x_9$	$x_{13}$
$x_2$	$x_6$	$x_{10}$	$x_{14}$
$x_3$	$x_7$	$x_{11}$	$x_{15}$



新しいブロック

$x_0$	$x_4$	$x_8$	$x_{12}$
$x_5$	$x_9$	$x_{13}$	$x_1$
$x_{10}$	$x_{14}$	$x_2$	$x_6$
$x_{15}$	$x_3$	$x_7$	$x_{11}$

ブロック

$x_0$	$x_4$	$x_8$	$x_{12}$
$x_1$	$x_5$	$x_9$	$x_{13}$
$x_2$	$x_6$	$x_{10}$	$x_{14}$
$x_3$	$x_7$	$x_{11}$	$x_{15}$



新しいブロック

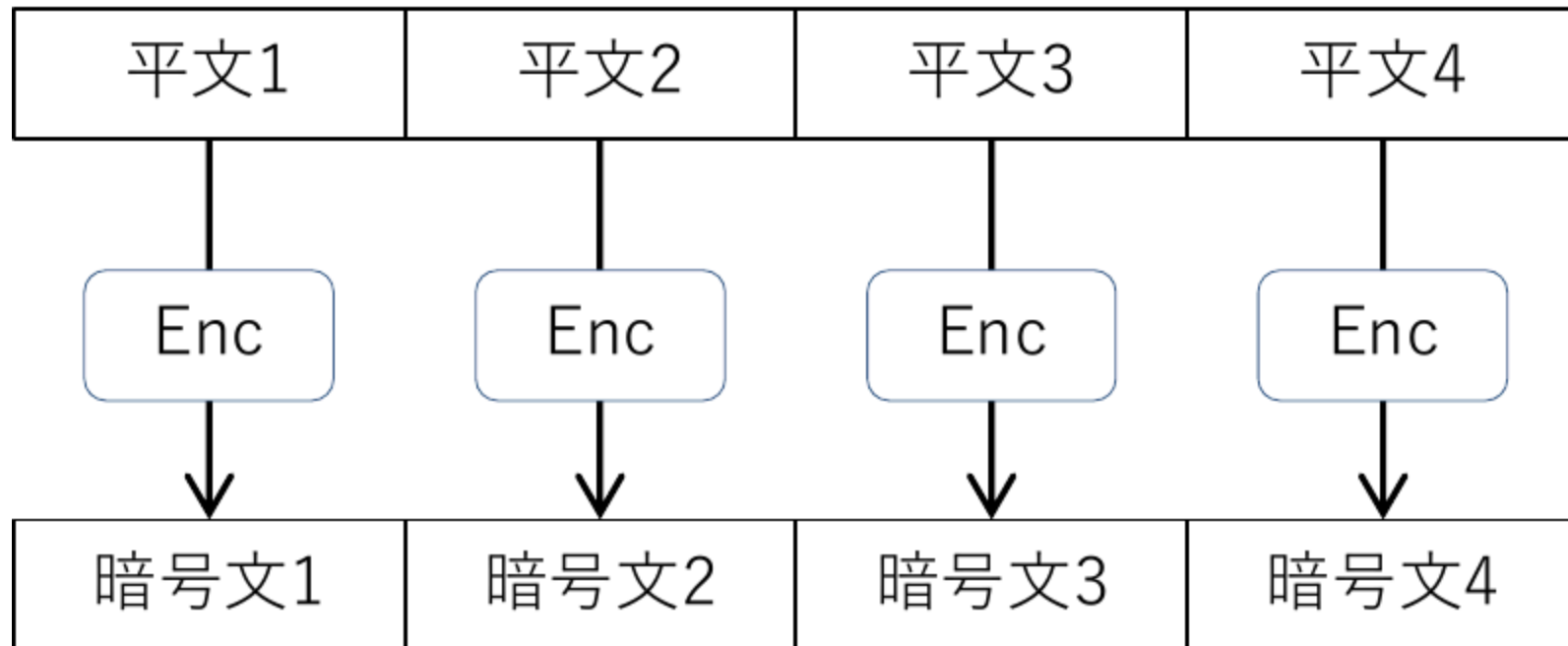
$x'_0$			
$x'_1$			
$x'_2$			
$x'_3$			

縦の列ごとに  
ある行列Aを掛ける

# 暗号利用モード

## ブロック暗号の使い方

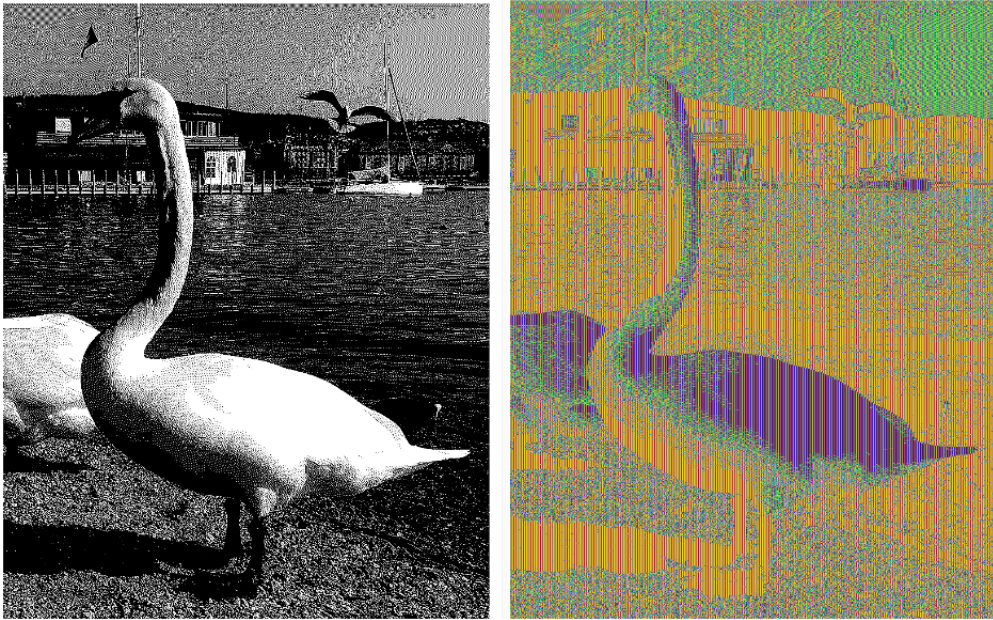
- ECB (Electronic Codebook) モード



- 平文を128bitごとに分割して暗号化
- 一番単純な使い方
- 安全ではない

# ECBモードで画像を暗号化

## 32bitカラー画像を白黒2値化した無圧縮データをECBモードで暗号化



- 元の情報がある程度見えてしまう（極端な例）
  - 128bit AESだと32bitが1ピクセル白黒なので横4ピクセルの白黒パターン16通りしかない
  - 16通りのデータがランダムな暗号文に対応する色に置き換わる
- 2020年Zoomの暗号化方式がECBモードを使っていたとしてニュースになる
  - （注意）動画などの圧縮データは分割されたブロックが同じ値になる確率は低い

# 確率的アルゴリズム

## そもそもアルゴリズムとは

- 入力に対して出力を返す手続き（ざっくりとした説明）

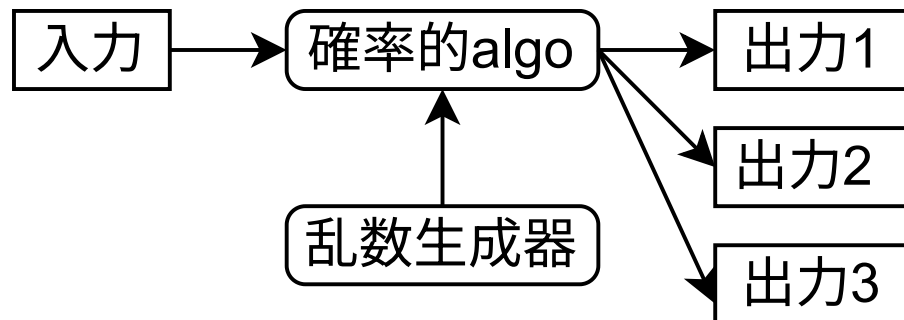
## 決定的 (deterministic) アルゴリズム

- 同じ入力に対して常に同じ出力を返すアルゴリズム



## 確率的 (probabilistic) アルゴリズム

- 同じ入力に対して異なる出力を返す可能性があるアルゴリズム
- アルゴリズム実行中に乱数を参照して結果を変化させる

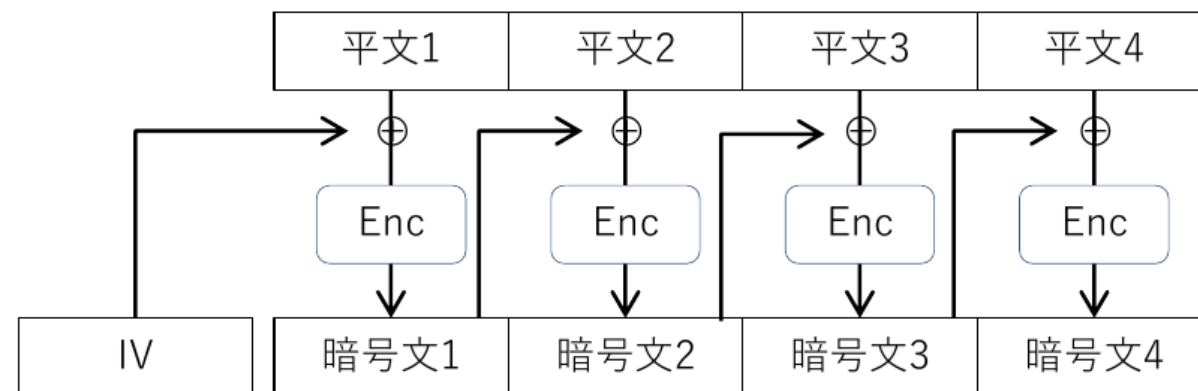




# CBC モード

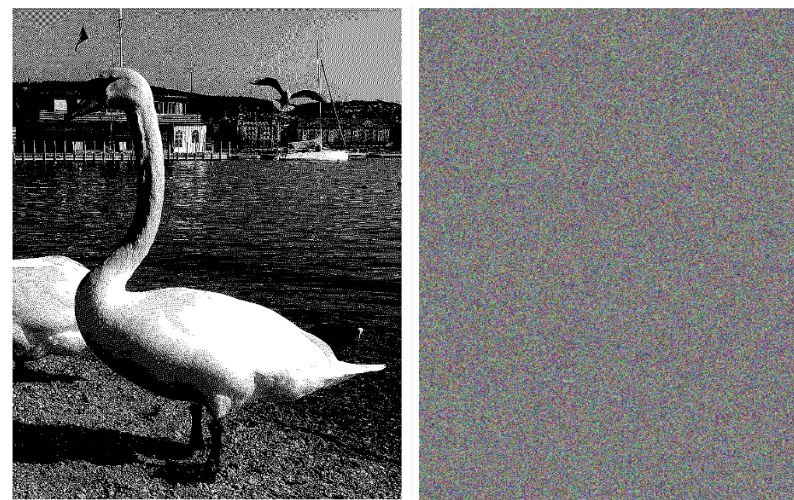
## IV (Initialization Vector) を利用

- CBC=Cipher Block Chaining = ブロックの鎖
- IVは予測できない一度しか使わないナンス
- 暗号文と一緒にIVも送る
- $c_1 = \text{Enc}(s, m_1 \oplus IV)$
- $c_k = \text{Enc}(s, m_k \oplus c_{k-1})$  for  $k \geq 2$ .



## 特徴

- IVが変わると同じ平文でも暗号文が変わる
- ECBでは見えていた鳥の情報が見えない



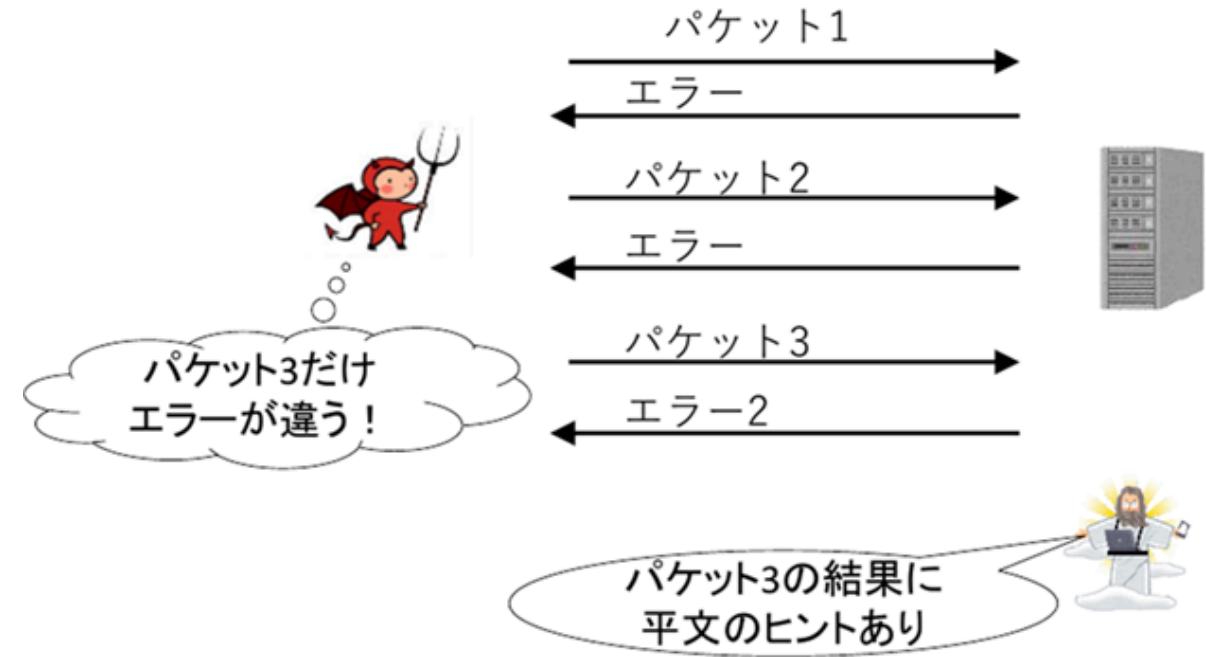
# CBCモードの利用減少

## 弱点

- 先頭から順番にしか暗号化できない
  - 複数のCPUで並列に暗号化はできない
- 問題: CBCモードの復号は並列化可能か否か?

## TLS1.3で廃止

- パディングオラクル攻撃POODLE (2014)
- ブロック暗号のパディング処理の実装問題
  - ブロックサイズに満たない端数の処理
  - サーバに不正な暗号文を送り返答を利用
  - サーバは復号時にエラー詳細を返してはいけない

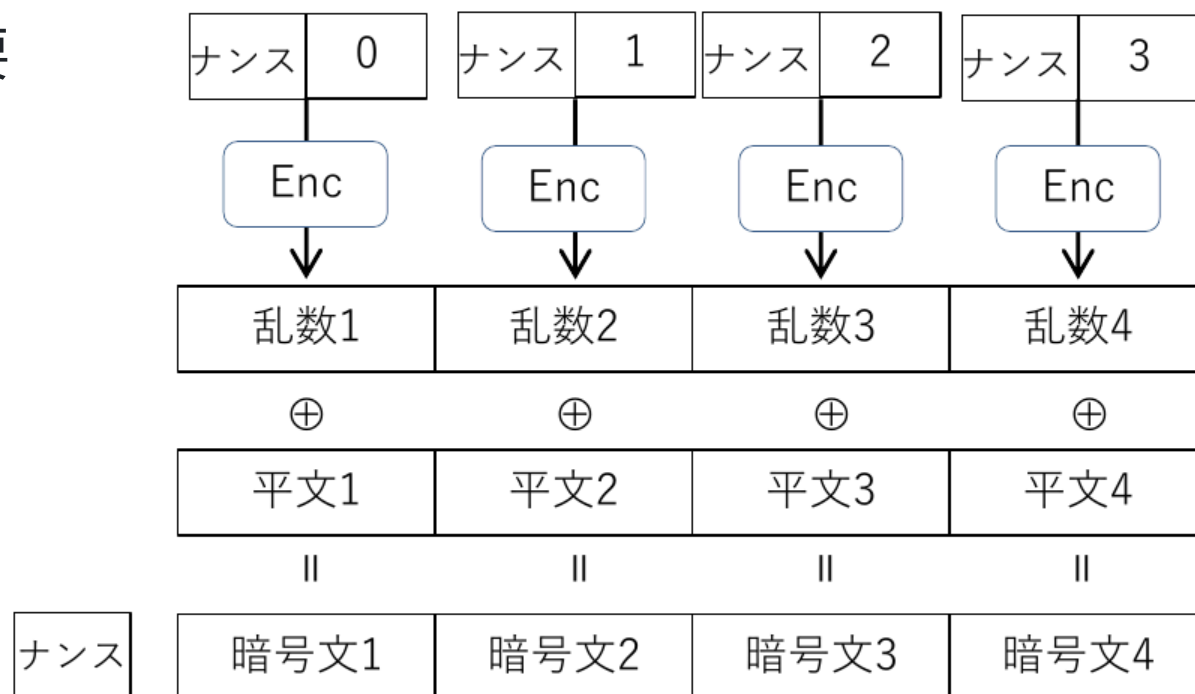




# CTR モード

## ブロック暗号をPRFとして利用するストリーム暗号

- CTR = CounTeR
- 「ナンス+カウンタ」を暗号化して乱数とする
- その乱数と平文のxorをとって暗号化
- 暗号化・復号共に並列処理可能
- パディング処理不要



## アルゴリズムの善し悪し

- 同じ結果を得るならより速いアルゴリズムがよい
- 少ないメモリ・少ない時間
- 入力パラメータの大きさ  $n$  について  $n$  が大きくなったときの振る舞いをみる

## O記法（オー）

- 関数  $f(n)$  が  $n$  の増加に伴いどのように増加するかを表す記法

## 例

- $f(n) = O(1)$ : 定数時間:  $f(n)$  は  $n$  がどんなに増えてもある定数以下である
- $f(n) = O(n^d)$ : 多項式時間:  $f(n)$  は  $n$  が大きいとき高々  $n^d$  の定数倍の大きさ
- $f(n) = O(\log(n))$ : 対数時間:  $f(n)$  は  $n$  が大きいとき高々  $\log(n)$  の定数倍の大きさ
- $f(n) = O(e^n)$ : 指数時間:  $f(n)$  は  $n$  が大きいとき高々  $e^n$  の定数倍の大きさ

# グラフの比較

## O記法での分類

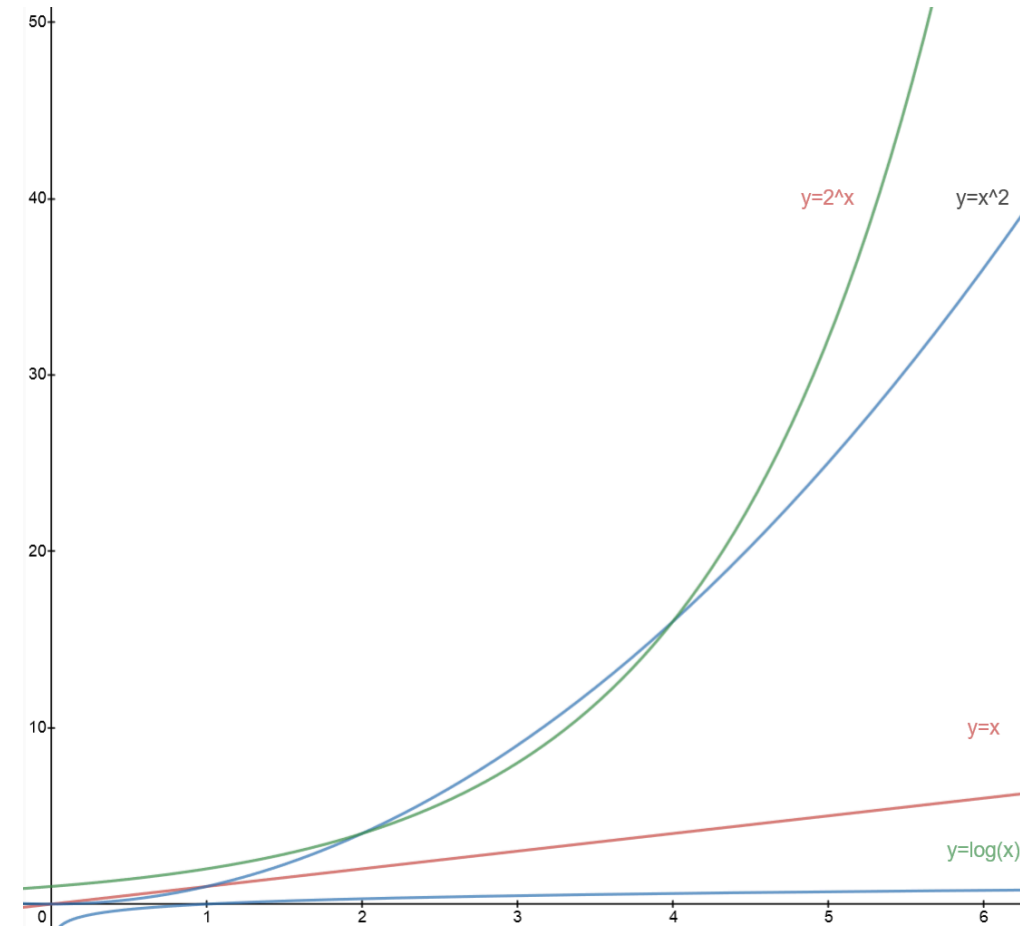
- 定数時間 < 対数時間 < 多項式時間 < 指数時間
- $2^x$  は指数時間,  $x^{100}$  は多項式時間
  - $x = 10$  のときは  $2^x \ll x^{100}$  だけれども

## 例

- $n$  bitの鍵の種類（鍵空間）は  $2^n$  通り
- $n$  個のデータの (merge) ソートは  $O(n \log n)$

## 注意

- $O(g(n))$  は「高々  $g(n)$  の定数倍の挙動」なので  $g(n)$  より小さくてもOK
- 丁度  $g(n)$  と言うときは  $\Theta(g(n))$  を使う (  $\iff f(n) = O(g(n))$  かつ  $g(n) = O(f(n))$  )
- $f(x) = o(g(n))$  であるとは  $\lim_{x \rightarrow \infty} f(x)/g(x) = 0$  であること



# クラス $P$

## 決定問題

- 与えられた入力に対して「はい」か「いいえ」で答えられる問題

## クラス $P$

- 多項式時間で解ける決定問題の集合

## 素数判定

- 「与えられた自然数  $n$  が素数である」という決定問題
- Agrawal–Kayal–Saxena (AKS2002) で多項式時間で判定できるアルゴリズムが提案された
  - クラス  $P$  に属する
  - 暗号ではそれよりも高速な別のPPTアルゴリズムが使われることが多い

# クラス $NP$

## NP = 非決定性多項式時間 (Non-deterministic Polynomial time)

- 「はい」の証拠をもらったとき、それを多項式時間で検証できる決定問題の集合
- クラス  $P$  は証拠を多項式時間で検証できるのでクラス  $NP$  に含まれる.  $P \subseteq NP$
- 「多項式時間で解けない問題の集合」 **ではない**

## SAT (Satisfiability Problem)

- 与えられた論理式  $\phi(x)$  について  $\phi(a) = 1$  となる  $a$  は存在するか?
  - 証拠  $a$  が  $\phi(a) = 1$  となるのを確認するのは多項式時間なのでクラス  $NP$  に属する

$P = NP$  or  $P \subsetneq NP$ ?

- $P = NP$  なら暗号理論はやり直し
  - 多項式時間で解けるかそうでないかの境界に安全性の根拠を置くことが多い

# 暗号の安全性

## 計算量の目安

- $n$  bitの秘密鍵の共通鍵暗号の最良の解読が鍵を一つ一つ試すしらみ潰し法しかないとき  $n$  bitセキュリティという
- 2025年現在のコンピュータでは  $2^{128}$  個の計算は無理:  $n = 128$  なら安全
  - 国家規模なら  $n \sim 80$  ぐらいは破られる

## 暗号理論で扱う計算量

- $n$  bitの秘密鍵に対して多項式時間で解読できる場合、安全ではないという
- 指数時間かかるのが理想
- 多項式時間と指数時間の間の準指数時間で解読できる暗号も実用的に扱う

## 暗号文の一部の情報も漏れてほしくない（強秘匿性）

- 例えば  $c = Enc(m)$  の  $m$  の偶奇やパリティビットが漏洩してほしくない

# 強秘匿性だけで十分か

## 受動的な攻撃者

- 盗聴するだけの攻撃者に対しては安全
- 暗号文を改竄する能動的な攻撃者については?
  - 特定のビットを反転させて平文を制御できた

## 能動的な攻撃者

- 能動的な攻撃者に対しても安全であって欲しい
  - 暗号文の完全性 (integrity)
    - MACやAEADなどで担保する (後の講義)
  - 頑強性 NM (non-malleability)
    - 平文を制御できるような暗号文を作れない (非展性とも)
    - $c = Enc(m)$  に対して  $m$  と関係のある別の  $m'$  に対応する  $c' = Enc(m')$  を作れない

どのような攻撃者を想定すべきか?

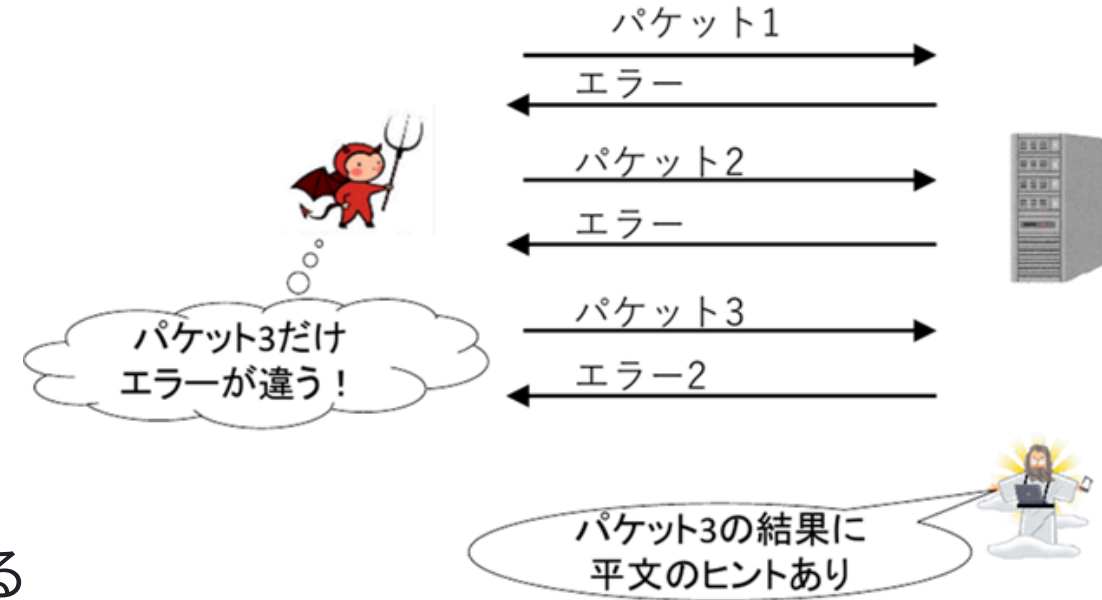
# オラクル (oracle)

## 問い合わせ可能なブラックボックス

- 神様の信託 (おつけ)
  - 何故か知らないけど答えを教えてくれる対象
  - 例: 暗号文を渡したら復号してくれる  
平文を渡したら暗号文を返してくれる

## POODLE (再掲)

- CBCモードにおけるパディングオラクル攻撃
- 改竄した暗号文を多数サーバに送り  
エラーの応答の違いから平文の情報を入手して攻撃
  - 復号はしてないが復号結果の一部を得られている





# 想定する状況

## 一番攻撃者に有利な状況を考える

- 攻撃者は暗号文  $c$  から平文の一部でもよいので情報を得たい
- 攻撃者は暗号文を知っている
  - 例: あるyes/noで答えるアンケート結果を盗聴しているので暗号文は0か1と推測できる
  - POODLEの極端な例として暗号文を渡したら復号してくれるオラクルを想定する
  - ただし流石に暗号文  $c$  を渡して復号してもらったら終わりなので  $c$  は渡せない

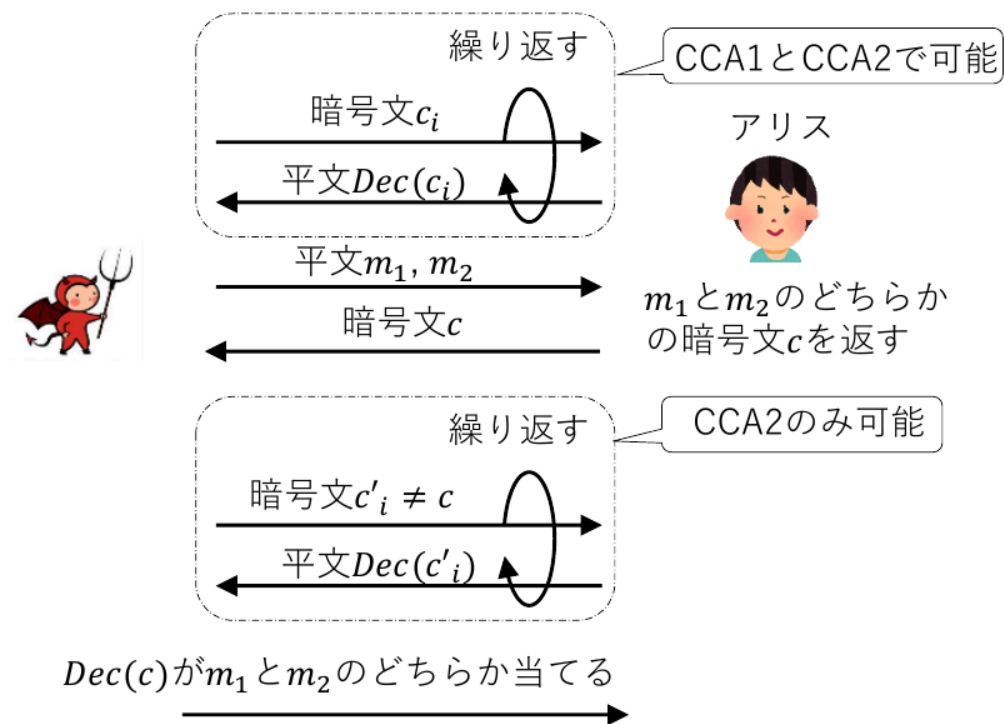
## 識別 (distinguish)

- そういう状況で元の暗号文  $c$  が 0 か 1 のどちらを暗号文したものかか識別できるか?
  - できたら攻撃者の勝ち (=その暗号方式は安全ではない)
- 識別できない IND (indistinguishability) なら安全
- この状況の定式化をする

# 選択暗号文攻撃

## CCA (Chosen Ciphertext Attack)

- 攻撃者  $\mathcal{A}$  は平文  $m_1, m_2$  を選ぶ
- アリスは  $i = 1, 2$  のどちらかを選び  $c = \text{Enc}(m_b)$  を返す
- $\mathcal{A}$  は対象暗号文  $c$  に対し暗号文  $c_i (\neq c)$  を選び  
対応する平文  $m' = \text{Dec}(c_i)$  を得られる状況
  - その時  $m = \text{Dec}(c)$  の情報を入手できるか?
- CCA1:  $c$  を受け取る前のみクエリ可能
- CCA2:  $c$  を受け取った後もクエリ可能
  - 適応的 (adaptive) CCAともいう
  - 試験中に先生に問題の類題の答えを聞ける状況



## IND-CCA(1/2)安全

- CCA(1/2)に対して安全な暗号

## それぞれ理由を記せ

1. ChaCha20はIND-CCA2安全ではない理由
2. CBCモードはIND-CCA2安全ではない理由