暗号技術における
乱数の活用例と落とし穴

縫田　光司（NUIDA, Koji）
東京大学 大学院情報理工学系研究科 数理情報学専攻

RCMSサロン@筑波大学　2019年12月18日

RSA暗号（“textbook RSA”）

- 鍵生成：
  - 秘密鍵　$sk = (p, q, d)$
  - 公開鍵　$pk = (N, e)$　（$N = pq$、$ed \equiv_{\phi(N)} 1$）
- 暗号化：$Enc(m) = m^e \ mod \ N$
- 復号：$Dec(c) = c^d \ mod \ N$
  - $Dec\big(Enc(m)\big) = m$　（正当性）

安全性上の問題

- 暗号化が確定的（同じ平文は同じ暗号文になる）
- 例：集計者に、３人のうち誰に投票するか暗号化して送信
  → 暗号文を入手した攻撃者は、候補者名すべてについて手元で暗号化して比較すれば平文（投票先）がわかる

- （現代の基準で）**安全な暗号化は確率的**でなければならない
  - 平文を**ランダム**に変換してから暗号化（RSA-OAEP）
  - はじめから**乱数**を用いて暗号化を設計（ElGamal暗号など）

秘密計算（[Yao 1982]～）

- 複数人で入力を持ち寄って何らかの計算をする
- ただし、入力そのものは隠す
- 例：二人の入力が一致しているか判定
  - 「＝」なら、相手の入力が自分と同じとわかる（仕様）
  - 「≠」のとき、「入力が異なる」ことしかわからない

- 応用研究が盛ん（プライバシー保護データマイニング）

実現法の一つ：秘密分散

- 例：２名の場合、入力はZ/NZの元（N：素数、2冪、など）
- Share(x) = [[x]] := ([[x]]_1,[[x]]_2)
  [[x]]_1 ← Z/NZ（**ランダムに選ぶ**）
  [[x]]_2 := x – [[x]]_1 in Z/NZ
- Reconst([[x]]) = [[x]]_1 + [[x]]_2 in Z/NZ

- 参加者P_iに[[x]]_iを渡す（i = 1,2）
  - 二人揃えばもとのxを復元可能
  - どちらか一人だけではxはランダムにしか見えない

秘密分散（加法、スカラー倍）

- Add([[x]],[[y]])：
  - 各自が[[x]]_i + [[y]]_iを手元で計算（通信なし）
  - Σ([[x]]_i + [[y]]_i) = (Σ[[x]]_i) + (Σ[[y]]_i) = x + y

- Mult(c,[[x]])　（cは公開値）：
  - 各自がc・[[x]]_iを手元で計算（通信なし）
  - Σ(c・[[x]]_i) = c・(Σ[[x]]_i) = cx

秘密分散（乗法）

- 三つ組([[a]],[[b]],[[c]])で、c = ab、かつa,b,cは**二人とも知らないランダムな値**であるものを使う（生成法は割愛）
- Mult([[x]],[[y]])：
  1. 先ほどの手順で各自がシェア[[x-a]],[[y-b]]を作る
  2. シェアを送りあってx’ := x − a, y’ := y − bを復元
     （**通信が1回発生**）
  3. [[z]]_1 := x’・[[b]]_1 + y’・[[a]]_1 + [[c]]_1 + x’y’
     [[z]]_2 := x’・[[b]]_2 + y’・[[a]]_2 + [[c]]_2
  - Reconst([[z]]) = (x-a)b + (y-b)a + ab + (x-a)(y-b) = xy

秘密計算の効率化の主な方針

- 「通信」は「計算」よりかなり遅い
- 「通信」の回数をできるだけ少なくする
- 1回の「通信」で多くのデータをまとめて送る
  - （およそ）通信時間＝1回ごとの遅延＋データ送信時間

多入力の乗算 [Ohata-N., to appear]

- 2入力の乗算には3個のシェアの組を用いた
  - (x-a)(y-b) – xy が3個の項からなるため
- 同様に、$2^M - 1$個のシェアの組をうまく作ると、
  M個の値の乗算を通信1回で計算可能
  - 素朴な二分法では通信Θ(log M)回
  - ただし、現実的にはMをあまり大きくできない（8程度？）

多項式計算 [樋渡-大畑-縫田 2019]

- シェアの組$([[a]],[[a^2]],\dots,[[a^d]])$を用いると、
  1変数のd次多項式の計算を通信1回で可能
  - $(x-a)^d - x^d$がd個の項からなるため

入力の一致判定 [樋渡-大畑-縫田 2019]

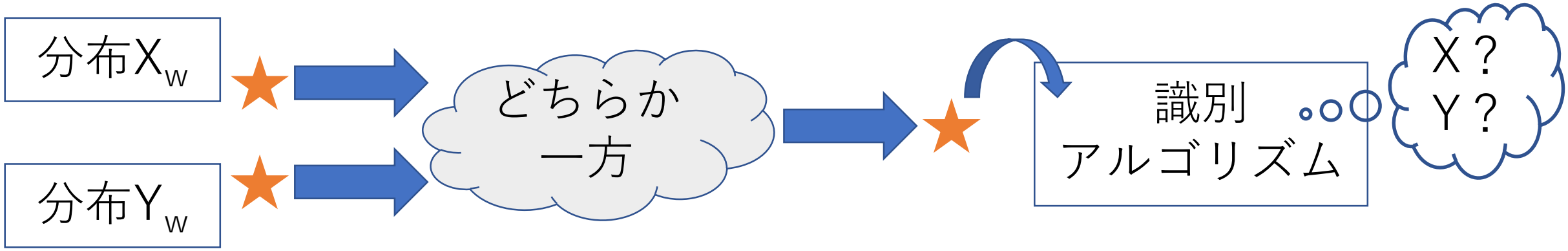
- P_1の入力xとP_2の入力yの一致判定
  - $x, y \in \mathbb{Z}/N\mathbb{Z},\ N = 2^n$　（n = 32, 64など）
  - zの二進k桁目のビットをz[k]で表す
  - $x = y \Leftrightarrow \forall k\ (\ x[k] + y[k] \in \{0,2\}\ )$

1. nより（少し）大きな素数pを用意
2. 法pのシェアとして$[[z_k]] \coloneqq \left([[\ x[k]\ ]], [[\ y[k]\ ]]\right)$とする
3. $[[w]] \coloneqq \sum_{k=0}^{n-1}[[z_k(2 - z_k)]]$　$(x = y \Leftrightarrow w = 0)$
4. $[[1 - w^{p-1}]]$を出力

暗号技術と擬似乱数

- 質の良い（安全な）乱数を大量に用意するのは難しい
- （暗号学的）擬似乱数生成器（PRG）を用いる
  - G(seed) = (long random bits)
- PRGの安全性：Gの出力分布とランダムな分布が
  効率的（多項式時間）識別アルゴリズムでは「識別不能」

確率分布（族）の識別不能性



分布（族）が**識別不能** … ほぼ1/2の確率でしか正答できない

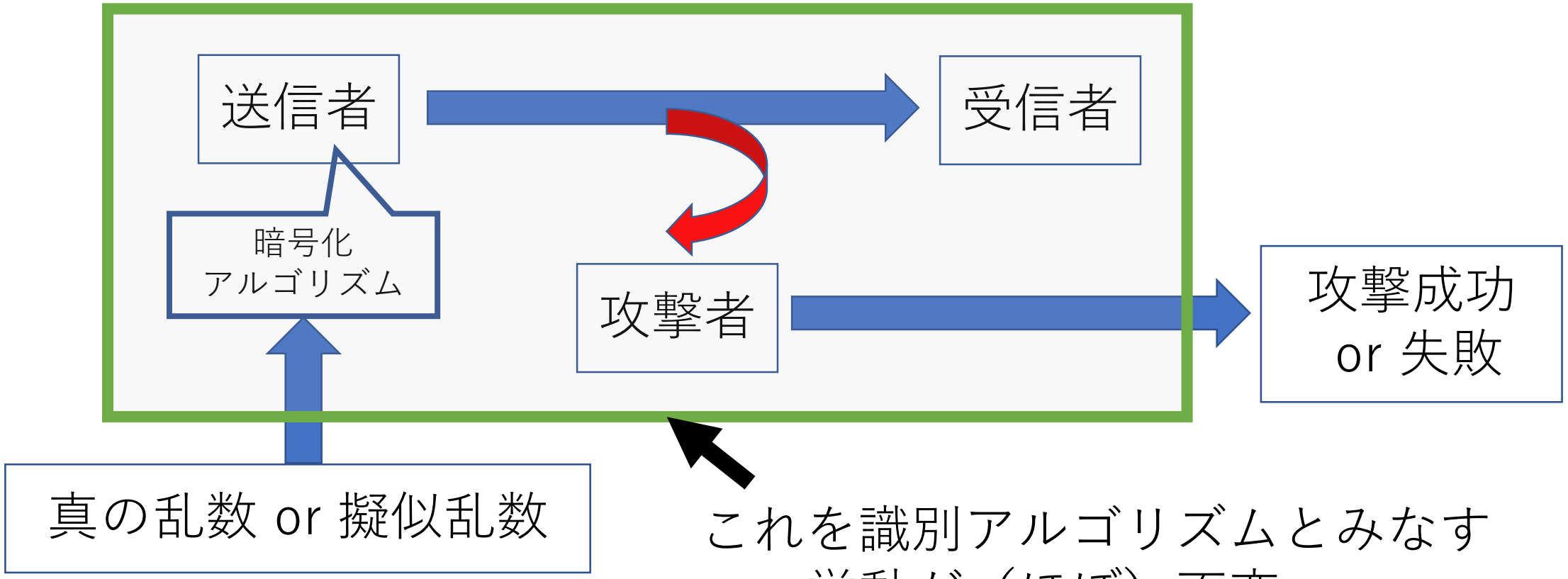
- **識別アルゴリズムはPRGのseedを見ない前提**

暗号学者の「直感」：

乱数をそれと識別不能な物と交換しても安全性は不変（のはず）

（むしろ、そうなるように識別不能性を定義している）

例：公開鍵暗号化の安全性



これを識別アルゴリズムとみなす
→　挙動が（ほぼ）不変
→　安全性が保たれる

秘密計算の場合 [N., preprint]

- 秘密計算の安全性：
  どの参加者も別の参加者の入力についての情報を得ない
- ある参加者が使う乱数を擬似乱数に置き換えたら？
- **定理**　以下を満たす秘密計算プロトコルとPRGが構成できる：
  - プロトコルは（真の乱数を使えば）安全
  - PRGは暗号学的に安全
  - ある参加者がそのPRGを使った場合、その参加者は
    別の参加者の入力を入手可能となる（安全性が失われる）
- 直感的な説明：攻撃者もプロトコルの参加者で、自分のPRGの seedを見られる（本来の識別アルゴリズムにはない機能）

まとめ + α

- 乱数は暗号において不可欠、かつ便利な道具
- 実用上は（暗号学的）擬似乱数が用いられる
- その際、安全性を直感的に論じるだけでは危険
  - 秘密計算の安全性以外にも、公開鍵暗号化の正当性が擬似乱数の利用により失われる例が存在 [N., preprint]
  - 他の暗号技術では大丈夫か？

References

- Satsuya Ohata, Koji Nuida: Communication-Efficient (Client-Aided) Secure Two-Party Protocols and Its Application. In: Financial Cryptography and Data Security 2020, to appear
- 樋渡 啓太郎、大畑 幸矢、縫田 光司：「近似多ビット乗算と新しい定数ラウンド基本ツールを用いた省ラウンド秘匿除算プロトコル」、コンピュータセキュリティシンポジウム2019
- Koji Nuida: Semi-Honest Secure Multiparty Computation Can Be Insecure with Use of Even Almost Uniformly Random Number Generators. https://eprint.iacr.org/2016/1040 (2016)
- Koji Nuida: Keeping or Losing Tiny-Error Correctness of Cryptosystems Implemented by Secure Pseudorandom Generators. https://eprint.iacr.org/2018/718 (2018)