数理的技法による情報セキュリティの 2024 年度前半の研究動向

荒井 研一 [1]，鈴木 幸太郎 [2]，中林 美郷 [3]，花谷 嘉一 [4]，
三重野 武彦 [5]，山本 光晴 [6]，吉田 真紀 [7]，米山 一樹 [8]
[1] 長崎大学，[2] 豊橋技術科学大学，[3]NTT 社会情報研究所，[4] 株式会社 東芝，
[5]EPSON AVASYS 株式会社，[6] 千葉大学，[7] 情報通信研究機構，[8] 茨城大学
e-mail : [3]misato.nakabayashi@ntt.com

1　はじめに

数理的技法は暗号プロトコルやシステムの安全性検証など，情報セキュリティの様々な分野で応用されている．本発表では，数理的技法による情報セキュリティに関する 2024 年前半の研究動向として，2024 年 4 月から 8 月までに開催されたトップ会議における関連論文の発表件数や特色，概要について紹介する．さらに，全体を通したトレンドや特に注目されている分野と論文を紹介する．

2　調査した会議と関連論文の件数

本発表のために調査した会議と関連論文の件数は以下の通りである．なお，関連論文は論文のタイトルおよびアブストラクトから抽出している．

2.1　Symposium on Security and Privacy (S&P) [1]

セキュリティ 4 大会議の一つ．理論的・実用的両方の発表が含まれるが，より実用を意識した発表が多い．数理的技法を用いた事例研究が盛ん．2024 年度は 4 件の関連論文が発表された．

2.2　USENIX Security [2]

セキュリティ 4 大会議の一つ．評価実験による実証を伴う実利用システムの安全性解析に関する発表が多い．数理的技法分野ではツールに関する発表が多く見られる．2024 年度は 4 件の関連論文が発表された．

2.3　The Network and Distributed System Security Symposium (NDSS) [3]

セキュリティ 4 大会議の一つ．特に分散環境下でのシステムなどの安全性解析や安全な開発に関する実用的な発表が多い．数理的技法分野の応用は少ない．2024 年度の関連論文は 0 件であった．

2.4　Computer Security Foundations Symposium (CSF) [4]

数理的技法によるセキュリティを主要なフォーカスの一つとした歴史ある会議．その後の研究に大きな影響を与えるような理論的な成果が集まる．フレームワークの提案や拡張，性質の証明，理論的限界の解明などに関する発表が多い．2024 年度は 20 件の関連論文が発表された．

2.5 International Conference on Computer Aided Verification (CAV) [5]

1980 年代から続く形式検証分野のトップ会議．検証の基礎となる理論から応用までを幅広くカバーする．例年セキュリティのセッションもあるが，2024 年度の関連論文は 0 件であった．

3 関連論文の件数の推移

図 1 に 2020 年から 2024 年までの調査対象会議における関連論文件数の推移を示す．S&P や USENIX Security，CSF では継続して一定数の関連論文が発表されていることがわかる．特に，CSF では年々増加傾向にある．

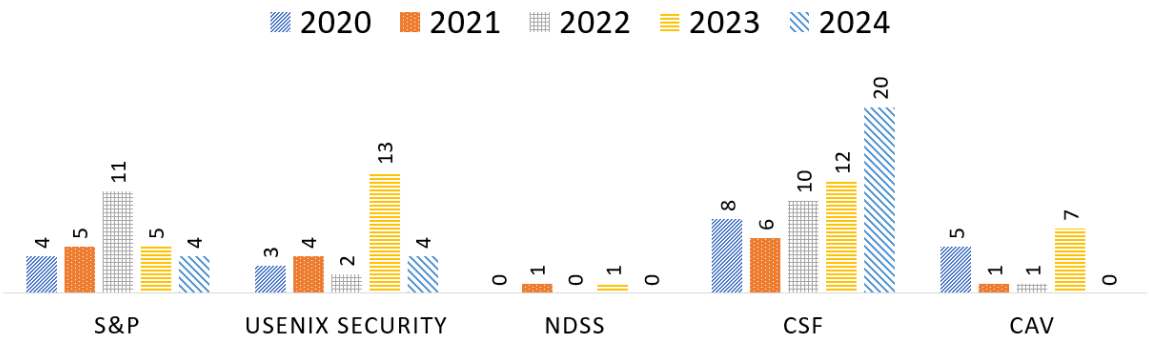


図 1. 2020 年から 2024 年までの調査対象会議における関連論文件数の推移.

参考文献

[1] 45th IEEE Symposium on Security and Privacy Web Page, https://sp2024.ieee-security.org/.

[2] 33rd USENIX Security Symposium Web Page, https://www.usenix.org/conference/usenixsecurity24.

[3] NDSS Symposium 2024 Web Page, https://www.ndss-symposium.org/ndss2024/.

[4] 37th IEEE Computer Security Foundations Symposium Web Page, https://csf2024.ieee-security.org/.

[5] 36th International Conference on Computer Aided Verification Web Page, https://i-cav.org/2024/.

LWE 暗号の弱鍵について

白勢 政明 [1]
[1] 公立はこだて未来大学
e-mail : shirase@fun.ac.jp

1　LWE 問題と LWE 暗号

行列 $A \in \mathbb{F}_p^{n\times m}$ $(n \le m)$ と行ベクトル $\boldsymbol{s} \in \mathbb{F}_p^n$, $\boldsymbol{e} \in \mathbb{F}_p^m$, $\boldsymbol{b} \in \mathbb{F}_p^m$ は $\boldsymbol{b} = \boldsymbol{s}A + \boldsymbol{e}$ を満たしているとする．但し，ノイズベクトル $\boldsymbol{e}$ の各成分は，十分に小さな σ に対して $D_{\mathbb{F}_p,\sigma}$ (平均値 0, 標準偏差 σ の $\mathbb{F}_p$ 上離散 Gauss 分布) に従って選ばれる．$(A, \boldsymbol{b})$ が与えられた時に $\boldsymbol{s}$(または $\boldsymbol{e}$) を求める問題を探索 LWE 問題 (以下，LWE 問題と呼ぶ) と言う．LWE 問題を解くには格子問題に帰着させることが一般的であるが，本稿は LWE 問題を整数計画 (IP) 問題に帰着させる方法に焦点を当てる．2015 年にバイナリ LWE 問題の IP 問題への帰着が提案され [1]，2023 年に一般の LWE 問題の IP 問題への帰着が提案された [2, 3]．LWE 問題の困難性を利用した耐量子計算機暗号である LWE 暗号は，公開鍵を $(A, \boldsymbol{b})$，秘密鍵を $\boldsymbol{s}$ とする．$n = 232,\ m = 1042,\ p = 32749,\ \sigma = 0.00217p\ \cdots(1)$ は [4] により推奨されているパラメータの一例である．

2　LWE 問題の IP 問題への帰着

行列 $A \in \mathbb{F}_p^{n\times m}$ $(n \le m)$ と行ベクトル $\boldsymbol{s} \in \mathbb{F}_p^n$, $\boldsymbol{e} \in \mathbb{F}_p^m$, $\boldsymbol{b} \in \mathbb{F}_p^m$ に対して，$(A, \boldsymbol{b})$ を LWE 問題のインスタンスとする．以下のように $A_0, A_1, \boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{e}_0, \boldsymbol{e}_1$ を定義する．

$$\begin{cases} A_0 = (A \text{ の左側の } n\times n \text{ 行列}), & A_1 = (A \text{ の右側の } n\times(m-n) \text{ 行列}) \\ \boldsymbol{b}_0 = (\boldsymbol{b} \text{ の左側の } n \text{ 次元ベクトル}), & \boldsymbol{b}_1 = (\boldsymbol{b} \text{ の右側の } (m-n) \text{ 次元ベクトル}) \\ \boldsymbol{e}_0 = (\boldsymbol{e} \text{ の左側の } n \text{ 次元ベクトル}), & \boldsymbol{e}_1 = (\boldsymbol{e} \text{ の右側の } (m-n) \text{ 次元ベクトル}) \end{cases}$$

A_0 は $\mathbb{F}_p$ 上正則であると仮定する．$W = A_0^{-1}A_1$ とおくと，$\boldsymbol{s} = (\boldsymbol{b}_0 - \boldsymbol{e}_0)A_0^{-1}\ \cdots(2)$ や $\boldsymbol{e}_0W - \boldsymbol{e}_1 = \boldsymbol{b}_0W - \boldsymbol{b}_1$ (移項して $\boldsymbol{b}_1 = \boldsymbol{b}_0W - (\boldsymbol{e}_0W - \boldsymbol{e}_1))\ \cdots(3)$ が成り立つ [3]．$w_{i,j}$ と u_i を $(w_{i,j}) = W$, $(u_1, u_2, \cdots, u_{m-n})^T = \boldsymbol{b}_0W - \boldsymbol{b}_1$ と定義する．$[-(p-1)/2, (p-1)/2]$ で表されている $\boldsymbol{e}$ の成分を変数 x_i を使って $\boldsymbol{e} = (x_1, x_2, \cdots, x_m)$ とおく．すると，式 (3) より次のような IP 問題を構成でき，最適解が $\boldsymbol{e}$ を表す (可能性が高い)．

$$\left.\begin{array}{l} \textbf{目的関数:}\ x_1^2 + x_2^2 + \cdots + x_m^2 \to \text{最小} \\ \textbf{制約式:} \\ \qquad\begin{array}{cccccc} w_{1,1}x_1 + & w_{2,1}x_2 + \cdots + & w_{n,1}x_n - & x_{n+1} + & py_1 = & u_1 \\ w_{1,2}x_1 + & w_{2,2}x_2 + \cdots + & w_{n,2}x_n - & x_{n+2} + & py_2 = & u_2 \\ & & \vdots & & & \\ \underbrace{w_{1,m-n}x_1 + w_{2,m-n}x_2 + \cdots + w_{n,m-n}x_n}_{\boldsymbol{e}_0W} & & \underbrace{-\ x_m}_{-\boldsymbol{e}_1} & \underbrace{+\ py_{m-n}}_{\mathbb{F}_p\text{を整数に}} & = & \underbrace{u_{m-n}}_{\boldsymbol{b}_0W-\boldsymbol{b}_1} \end{array} \end{array}\right\} \quad (4)$$

3　LWE 暗号の弱鍵生成 (本稿の寄与)

正方行列に対して，対角成分とその巡回右隣以外の成分が 0 の時，その行列は “ほぼ対角”であると言うことにする．

本稿は，$n = 60,\ 80,\ 100,\ \cdots,\ 240$ に対して，(ほぼ) 対角な W をランダムに選び，p は $p > n^2$ を満たす最小素数とし，(1)と同様に$\sigma = 0.00217p$ として $D_{\mathbb{F}_p,\sigma}$ に従ってノイズベクトル $\boldsymbol{e}$ を選び，(4) のように構成される IP 問題をそれぞれ 10 個作成しソルバー SCIP を使って解き，正解率と実行

時間を評価した．なお，$n \geq 200$ の場合は，時間の都合上ほぼ対角な W による IP 問題は解くことができなかった．各 n に対して，10 個の問題中正しく解が得らた問題の個数は

W のタイプ	$n=60$	80	100	120	140	160	180	200	220	240
対角な W	10	10	8	6	8	3	3	6	1	0
ほぼ対角な W	10	10	10	9	10	10	10	−	−	−

となり，対角な W による IP 問題は正解が得られないことが多かったが，ほぼ対角な W ではほとんどで正解が得られた．SCIP の実行時間の「平均 (s)/その分散」は次のようになった (環境は OS: Windows11，CPU: Intel(R) Core(TM) i3-8145U)．なお，下表中の A_B は $A \times 10^B$ を意味する．

W	$n=60$	80	100	120	140	160	180	200	220	240
対角	0.0/0.0	0.1/0.1	0.1/0.1	0.0/0.0	0.1/0.1	0.3/0.2	0.3/0.2	0.7/0.4	1.0/0.6	5.8/10.2
ほぼ対角	0.0/0.0	0.0/0.0	0.0/0.0	0.3/0.4	2.4/21	$2.4_2/3.5_4$	$3.4_3/3.3_7$	−	−	−

この表から，ほぼ対角な W による IP 問題に対して，n が 20 増えるごとに SCIP の実行時間は 12 倍程度増加しそうである．よって，$n=200$ で半日程度，$n=220$ で 5，6 日，$n=240$ で 2 か月程度と予想される．この時間は暗号攻撃としては現実的な時間である．

$n=240$ 程度のほぼ対角な W による IP 問題は，高い確率で現実的な時間で解けると仮定する．すると，次の**アルゴリズム 1** によって，$A \in \mathbb{F}_p^{n\times 2n}$ による LWE 暗号の弱鍵を生成することができる．なお，ステップ 7 は (3) により，ステップ 9 は (2) による．

アルゴリズム 1 (LWE 暗号の弱鍵生成)

入力: n, p, σ

出力: LWE 暗号の公開鍵 $(A, \boldsymbol{b}) \in \mathbb{F}_p^{n\times 2n} \times \mathbb{F}_p^{2n}$，秘密鍵 $\boldsymbol{s} \in \mathbb{F}_p^n$

1. (ほぼ) 対角な $W \in \mathbb{F}_p^{n\times n}$ をランダムに選ぶ	6. $\boldsymbol{b}_0 \in \mathbb{F}_p^n$ をランダムに選ぶ
2. $D_{\mathbb{F}_p,\sigma}$ に従って $\boldsymbol{e}_0, \boldsymbol{e}_1 \in \mathbb{F}_p^n$ を選ぶ	7. $\boldsymbol{b}_1 = \boldsymbol{b}_0 W - (\boldsymbol{e}_0 W - \boldsymbol{e}_1)$ を計算する
3. $A_0 \in \mathbb{F}_p^{n\times n}$ をランダムに選ぶ	8. $\boldsymbol{b} = (\boldsymbol{b}_0\ \boldsymbol{b}_1)$ とおく
4. $A_1 = A_0 W$ を計算する	9. $\boldsymbol{s} = (\boldsymbol{b}_0 - \boldsymbol{e}_0)A_0^{-1}$ を計算する
5. $A = (A_0\ A_1)$ とおく	10. $(A, \boldsymbol{b})$ と $\boldsymbol{s}$ を返す

この方法による弱鍵の対策であるが，公開鍵 $(A = (A_0\ A_1), \boldsymbol{b})$ に対して，$A_0^{-1}A_1$ が (ほぼ) 対角でないことを確かめれば十分である．例え A の列の入れ換えが行われていても，列の入れ換え前の $A_0^{-1}A_1$ が (ほぼ) 対角ならば，入れ換え後の $A_0^{-1}A_1$ も (ほぼ) 対角になりそうである．

本稿は，$A \in \mathbb{F}_p^{n\times 2n}$ に限定したが，より一般の A に対しても弱鍵の生成の可能性を調査したい．

参考文献

[1] 町野義貴, 青野良範, 高安敦, 國廣昇. 整数計画問題による binary-LWE 問題の求解アルゴリズム. SCIS2015, 2015.

[2] Masaaki Shirase. Reduction of search-LWE problem to integer programming problem. Cryptology ePrint Archive, Paper 2023/1162, 2023.

[3] 白勢政明. LWE 問題に関する整数計画問題の一考察. SCIS2024, 2024.

[4] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pp. 147–191. Springer, 2009.

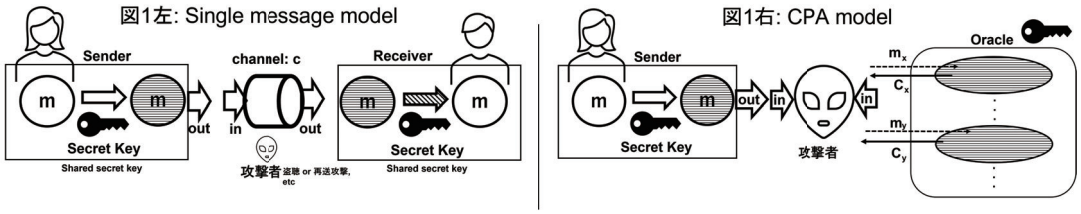
Tamarin prover の Heuristic Oracle を利用した CPA Model の安全性検証

三重野 武彦 [1], 岡崎 裕之 [2], 荒井 研一 [3], 布田 裕一 [4]
[1]EPSON AVASYS 株式会社, 信州大学,[2] 信州大学, [3] 長崎大学, [4] 東京工科大学
e-mail : mieno.takehiko2@exc.epson.co.jp

1　概要

Chosen Plaintext Attack(CPA) は, 攻撃者が選択した平文に対して, 暗号文を取得できることを前提とした暗号解読の攻撃モデルである. これは, 攻撃者がブラックボックスとみなした暗号化 Oracle と対話できるようにすることで形式化される. 本稿では, Dolev–Yao モデルのプロトコル検証ツールである Tamarin prover[1] を用い, 幅優先探索 (BFS) と組み合わせることができる Heuristic Oracle を使用して, CPA モデルの安全性検証を実施した. Tamarin prover の正式な扱いは, Schmit ら [2] によって与えられているが, Heuristic Oracle を使用する検証は, Tamarin prover がデフォルトで実行する標準的な攻撃発見手法 (完全自動モード) に比べて, 証明探索の選択肢をカスタムで記述できる. 目的の証明に繋がる Heuristic Oracle を導入することで, 効率的に証明したい lemma を検証することができることを示す.

2　CPA モデル



使用するモデルは, 共有する秘密鍵 (一回限り) を作成後, それを使って平文を暗号化するものとする (図 1 左). 攻撃者は, 暗号化 Oracle を利用して暗号文を得ることができる. 暗号化 Oracle にアクセス出来るタイミングは, 任意のタイミングである. 検証 (平文の秘匿性を含む) は, 状態の網羅探索を実施することになるため, 攻撃発見まで永久に止まらないモデルとする (図 1 右).

3　形式化

検証する CPA モデルは, メッセージを暗号プリミティブを表す関数記号から構成される抽象項として記述する. 暗号プリミティブの振る舞いは, 項に関する等式理論によって規定される. 共通鍵を漏らさない限り, 送信者の暗号文と攻撃者が作った暗号文, それぞれの値の不一致が保たれていれば良いことを証明するため, 図 2 左の “Tamarin CPA Code” 23 行目で, 証明したい Action Fact “Notequalvalue” を入れている. Oracle は python プログラムを利用し, 証明の順番を Output する処理を意図的に入れている. 但し, Oracle が正しく動作しているかどうかの証明ツリーは, 手作業で検査する必要があった. そのため作成した Oracle は, Tamarin prover の Web I/F を使用し, 証明を Step 実行することで検証した. 攻撃者は, In Fact と Out Fact を介して Oracle にアクセスすることが出来る. したがって, 下記 2 点を主なアイデアとした. (1) 特定の事象 (In, Out で起きている, senc

図2左 : Tamarin CPA Code

```
                    :↓
                (omitted)↓
                    :↓
↓
rule Encrypt_oracle:↓
 [ In(<$A, m_>), Fr(~key) ,St_Attacker(m_)] --[Send($A, senc(<$A, m_>, ~key)),
Secret(m_), Oracle($A), Sender($B), Role('Oracle'), Oracle_send_plain(m_) ]->
[ Out(senc(<$A, m_>,~key)), St_ReceiveKeyOracleplain(m_,~key) ]↓
↓
rule Sender_encrypt:↓
[ Fr(~m), Fr(~key)  ]  --[Send($B, senc(<$B, ~m>, ~key)), Secret(~m),↓
Sender($B), Oracle($A), Role('Sender'), Sender_send_plain(~m) ]->↓
[ Out(senc(<$B, ~m>,~key)), St_ReceiveKey(~m,~key) ]↓
↓
rule Attacker:↓
let↓
c_ = senc(m_ , key_)↓
c  = senc( m, key)↓
message = sdec(c_, key)↓
in↓
 [St_ReceiveKeyOracleplain(m_,key_), In(senc(<$A, m_>,key_)),↓
St_ReceiveKey(~m,key), In(senc(<$B, ~m>,key)) ]↓
 --[ Notequalvalue(<c_, c>), Secretmessage(message)]-> [  ]↓
                    :↓
                (omitted)↓
                    :↓
restriction choiceplain:↓
 "All x #i. Oracle_send_plain(x)@i↓
   ==> (Ex y #j. Oracle_send_plain(y)@j & #j < #i) |↓
  (Ex y #j. Sender_send_plain(y)@j & #j < #i)"↓
↓
lemma value_notequal:↓
 "All #i #j x. Notequalvalue(x)@i & Notequalvalue(x)@j ==> #i=#j"↓
                    :↓
                (omitted)↓
                    :[EOF]
```

図2中: My oracle

```
                :↓
            (omitted)↓
                :↓
↓
lines = sys.stdin.readlines()↓
↓
la = []↓
lb = []↓
lc = []↓
ld = []↓
lemma = sys.argv[1]↓
↓
for line in lines:↓
  num = line.split(':')[0]↓
↓
  if lemma == "value_notequal":↓
      if ": St_ReceiveKey" in line:↓
        la.append(num)↓
      elif "senc(<$B," in line↓
       or "senc(<$B, ~m>,key" in line:↓
        lb.append(num)↓
      elif "KU(key" in line ↓
       or "KU(~key" in line:↓
        lc.append(num)↓
      else:↓
        ld.append(num)↓
  else:↓
↓
    exit(0)↓
↓
ranked = la + lb + lc + ld↓
↓
for i in ranked:↓
  print(i)↓
[EOF]
```

図2右: Code execution

```
                :
            (omitted)
                :
>>>>>>>>>>>>>>>>>>>>>>> START INPUT
0:  (#i < #j)  ∧ (#j < #i)
1: St_ReceiveKeyOracleplain( m_, key_ ) ▸₀ #i
2: !KU( senc(<$A, m_>, key_) ) @ #vk
3: St_ReceiveKey( ~m, key ) ▸₂ #i
4: !KU( senc(<$B, ~m>, key) ) @ #vk.1
5: St_ReceiveKeyOracleplain( m_, key_ ) ▸₀ #j
6: !KU( senc(<$A.1, m_>, key_) ) @ #vk.2
7: St_ReceiveKey( ~m, key ) ▸₂ #j
8: !KU( senc(<$B.1, ~m>, key) ) @ #vk.3

>>>>>>>>>>>>>>>>>>>>>>> START OUTPUT
1
3
5
7
4
0
2
6
8
>>>>>>>>>>>>>>>>>>>>>>> END Oracle call
                :
            (omitted)
value_notequal (all-traces): verified (5 steps)
secret_sender_message (all-traces): verified (6 steps)
m_secret (all-traces): verified (34 steps)
```

による任意の平文の暗号化) の制約優先順位を優先する．(2) 攻撃者が秘密鍵を推論する制約の優先順位を下げる．証明したい lemma の Action Fact を Oracle に指定するため, 図 2 中 “Myoracle” の 16 行目 “value_notequal” を条件として記述している．Oracle は, 全ての証明目標を EOL で区切って, Stdin 経由でプロセスとして呼び出される．図 2 右 “Code execution” は, 証明の順番 (一部抜粋) を “START OUTPUT” から “END Oracle call” で確認することができる．

4　検証結果

Oracle の有無と, 証明したい lemma の検証結果をまとめた (表 1)．本研究で使用した検証コードはすべて GitHub へ掲載している [3].

表1: Oracle 有無による検証結果

Oracle 無し	Oracle 有	証明したい補題: lemma	検証結果
検証が止まらない	検証が止まる	value_notequal (送信者の暗号文と攻撃者の暗号文が一致しない)	verified(値が一致しない証明が完了)
		secret (秘匿性が保たれている)	verified(攻撃発見無し)

5　結論と今後

Tamarin prover の自動検証は, 一般的には BFS の方が早く攻撃を見つけることが出来ると言われているが, 証明についてはそうではない．本稿は, BFS と組み合わせることが出来る Heuristic Oracle を使用して, CPA モデルの安全性検証を実施した．検証が終わらないモデルを Heuristic Oracle を使用し, 証明したい lemma を効率的に検証できることを示した．今後は, 実務で利用している暗号プロトコルのセキュリティ特性を, Heuristic Oracle を利用し検証することに取り組んでいく．

謝辞　本研究の一部は JSPS 科研費’22K11982 の助成を受けたものです．

参考文献

[1] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, B. Schmidt. Tamarin prover, Version 1.8.0, 2024. Available at: `https://tamarin-prover.github.io/`

[2] Schmidt, B. “Formal Analysis of Key Exchange Protocols and Physical Protocols,” Ph.D. thesis (2012).

[3] T. Mieno, H. Okazaki, K. Arai and Y. Futa., Tamarin prover models and logs to reproduce the results in this paper, 2024. Available at: `https://github.com/mienotakehiko/fais2024_summer`

カードベース暗号の形式検証の再考

藤田 和弘 [1], 米山 一樹 [1], 品川 和雅 [1,2]
[1] 茨城大学, [2] 産業技術総合研究所
e-mail : 23nm738n@vc.ibaraki.ac.jp

1　形式手法を用いたカードベース暗号の不可能性証明

近年，トランプのようなカードを用いて秘密計算やゼロ知識証明を物理的に実行するカードベース暗号の研究が進み，カード枚数やステップ数の下界についての関心が高まっている。Koch ら [1] はカードベース暗号プロトコルのカード枚数またはステップ数に関する下界証明のため，指定したカード枚数およびステップ数のプロトコルが存在するかどうか形式検証を行った。この手法を応用することで様々な関数についての不可能性証明が進展することが期待されるが，我々は検証に使用されたプログラムの誤りを発見した。よって，誤りに起因して正しい結果が得られない可能性がある。本研究では，Koch らの検証プログラムの誤りをモジュールごとに具体的な反例とともに指摘し，検証結果への影響を明らかにする。また，指摘した誤りの適切な修正方法を提案し，反例を正しく処理できることを示す。

2　検証プログラムの誤りと修正

Koch らが検証に用いたプログラムの誤りとその影響範囲は表 1 の通りである。なお，それぞれの誤りについての反例や修正を実際に確認可能なプログラムは GitHub [2] を閲覧されたい。

2.1　shuffle 後の correctness 検証

correctness の検証では，shuffle 後の state に bottom sequence が含まれていないことを確認する。ここで，あるカード列が bottom sequence であることはカード列が出力 0 と 1 の両方に属することであるが，検証プログラムでは記述の誤りにより出力 1 だけに属するかどうかという条件を検証してしまっている。これは，例えば確率が X_{11} であるカード列（出力 1 だけに対応するカード列）を bottom sequence であると誤判定してしまうことになり，偽陽性による誤りである。

誤り	影響範囲				
	カードの種類		検証する安全性		
	数字カード	2 色カード	probabilistic	input-possibilistic	output-possibilistic
shuffle 後の correctness 検証	✓		✓	✓	
shuffle 後の確率計算	✓	✓	✓		
turn 後の security 検証	✓	✓	✓		

表 1. 検証用プログラムの誤りとその影響範囲

“✓” はプログラムの誤りがその条件下での検証結果に影響を及ぼすことを示す。ただし，検証条件のうちカードの種類と検証対象の安全性の両方が “✓” の付いている条件である場合のみ，プログラムの誤りが検証結果に影響を及ぼす。

2.2 shuffle 後の確率計算

shuffle は置換集合 Π から確率分布（ここでは一様分布とする）に従って置換 π を選択しカード列に適用する操作として定式化されているため，カード列 s_n の確率を $\mu(s_n)$ とすると，shuffle 後のカード列の確率は $\mu(s_n) := \frac{1}{|\Pi|}\sum \mu(\pi^{-1}(s_n))$ と表される。これを変形し入力確率の係数に着目すると $\frac{1}{|\Pi|}\cdot\frac{n_i}{d_i}$ のように積の形になるが，検証プログラムでは $\frac{1}{|\Pi|}+\frac{n_i}{d_i}$ のように和の形で記述されているため，計算結果が誤りとなる。例として，図 1 のような state と shuffle では shuffle 後のカード列の確率の係数はすべて $\frac{1}{2}$ だが，検証プログラムで計算すると係数がすべて $\frac{3}{2}$ となってしまい，明らかに誤りである。

2.3 turn 後の security 検証

turn 後の security の検証では安全性定義に従い，各 observation の確率 ρ が定数であることを確認する。turn によって state は図 2 のように分岐するが，ρ は分岐後のそれぞれの state について state 中のすべてのカード列の確率の和をとったものである。しかし，検証プログラムでは ρ を計算する際に分岐後のすべての state のカード列の確率の和をとっているため，誤った計算結果となる。例えば図 2 の observation が 1 の場合では，正しい計算では $\rho = \frac{1}{3}X_{00}+\frac{1}{3}X_{01}+\frac{1}{3}X_{10}$ となりこれが定数でないため security を満たさないと判定すべきだが，検証プログラムで計算すると $\rho = 3(\frac{1}{3}X_{00}+\frac{1}{3}X_{01}+\frac{1}{3}X_{10}+\frac{1}{3}X_{11}) = 1$ となり ρ が定数であることから security を満たすと誤判定してしまうことになり，偽陰性による誤りである。

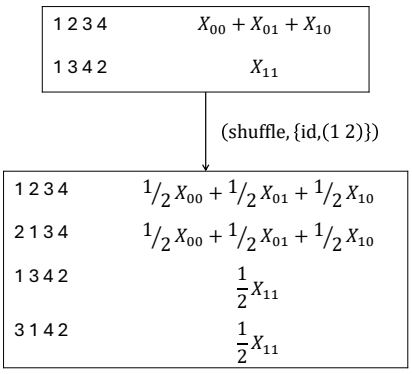


図 1. shuffle 後の確率計算についての反例

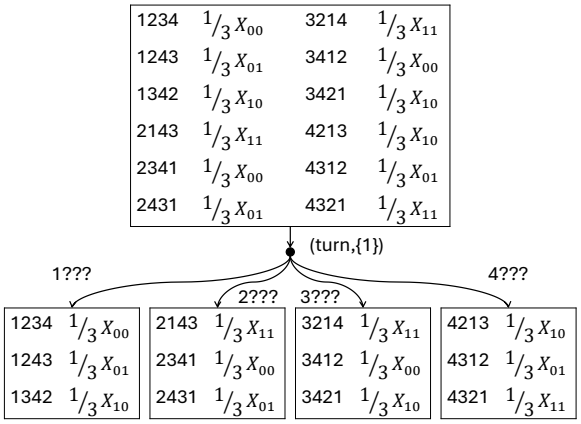


図 2. turn 後の security 検証についての反例

謝辞　本研究は，JST，CREST，JPMJCR22M1 の支援を受けたものである。

参考文献

[1] Koch, A., Schrempp, M. and Kirsten, M., Card-based cryptography meets formal verification, New Generation Computing, vol.39 (2021), 115-158.

[2] GitHub, `https://github.com/fujitargz/fix-cardCryptoVerification`

