エラー付きLS系列に対するエラー訂正アルゴリズム

谷 健太[1,a)]　國廣 昇[2]

概要：RSA 暗号の鍵生成では，最大公約数の計算時に Binary GCD アルゴリズムが使用されることがある．CHES 2019 において，Aldaya らは，Binary GCD アルゴリズムで実行される演算情報の系列をサイドチャネル攻撃により収集し，秘密鍵を復元できることを示した．得られる演算系列には様々な種類のエラーが含まれているため，秘密鍵を復元するためには，演算系列に含まれるエラーを訂正する必要がある．Aldaya らの提案したエラー訂正アルゴリズムは単一種類のエラーのみを訂正可能である．我々は，SCIS 2021 において，複数種類のエラーが含まれる場合でもエラー訂正が可能なアルゴリズムを提案した．提案アルゴリズムは単一種類のエラーに対しても適用可能であるが，復元率が低いという問題があった．本稿では，SCIS 2021 での提案アルゴリズムを改良することにより，単一種類のエラーのみに適用可能であるが復元率が高いエラー訂正アルゴリズムを提案する．単一種類のエラーに限定された状況で提案アルゴリズムの訂正能力を数値実験により検証し，アルゴリズムは，既存方式よりも復元率が高いことを確認した．

キーワード：RSA 暗号，サイドチャネル攻撃，エラー訂正

Error correction algorithm for erroneous LS sequences

Kenta Tani[1,a)]　Noboru Kunihiro[2]

Abstract: In RSA key generation, the Binary GCD algorithm is sometimes used to calculate the greatest common divisor. In CHES 2019, Aldaya et al. proposed that a sequence of operations performed in the Binary GCD algorithm can be collected by a side-channel attack. Moreover they showed that it is possible to recover the secret key from the sequence. Since the obtained sequence contains various types of errors, it is necessary to correct the errors in the sequence in order to recover the secret key. The error correction algorithm proposed by Aldaya et al. can correct only a single type of error. In SCIS 2021, we proposed an algorithm that can correct errors even when multiple types of errors are included in the obtained sequence. Although the proposed algorithm can be also applied to a single type of errors, its recovery rate is low. In this paper, we propose an error correction algorithm with a high recovery rate that can be applied to only a single type of errors by improving our proposed algorithm in SCIS 2021. Numerical experiments are conducted to verify the ability of the proposed algorithm to correct only a single type of error, and it is confirmed that the algorithm has a higher recovery rate than the algorithm in SCIS 2021.

Keywords: RSA, Side-Channel attack, Error correction

1. はじめに

1.1 研究背景

オープンソースの暗号ライブラリの一つに OpenSSL がある．OpenSSL には Binary GCD アルゴリズムが安全に実装されていないバージョンが存在する．Aldaya らは，この脆弱性を発見し，RSA 暗号 [6] の鍵生成時に安全でない Binary GCD アルゴリズムが使用されていることを指摘した [1]．さらに，Binary GCD アルゴリズムで実行される演算系列から秘密鍵を復元する攻撃を提案した [1]．Aldaya らの攻撃では，サイドチャネル攻撃を用いて Binary GCD アルゴリズムで実行される演算系列を収集する．しかし，一般に，サイドチャネル攻撃で得られる情報にはエラーが

[1] 筑波大学情報理工学位プログラム, University of Tsukuba
[2] 筑波大学システム情報系, University of Tsukuba
[a)] s2120617@s.tsukuba.ac.jp

含まれる．攻撃が成功するためには，このエラーを訂正する必要がある．Aldaya らは，単一の種類のエラーを訂正するアルゴリズムを提案した [1]．我々は，SCIS 2021 において，複数の種類のエラーが含まれる場合でもエラーを訂正できるアルゴリズムを提案した [9]．しかし，定性的には複数の種類のエラー訂正が可能であるが，訂正できるエラーの量が非常に限定的であるという問題点がある．本稿では，単一の種類のエラーを訂正するアルゴリズムを提案する．本稿で提案するアルゴリズムは，[9] と比較して，その単一種類のエラーだけに注目すれば多くの量のエラーを訂正することが可能である．

1.2 研究成果

まず，単一種類のエラー訂正に特化したエラー訂正アルゴリズムを提案する．このアルゴリズムは，公開鍵とサイドチャネル攻撃で得られた情報をもとに，解候補を列挙し，その解候補を適切な基準を用いて削減することで効率的なアルゴリズムとなる．適切な基準として，尤度に基づく損失関数とノルムに基づく損失関数を提案する．これらは攻撃者がもつ情報によって使い分けることができる．ついで，数値実験を行い提案アルゴリズムの効果を検証する．我々の SCIS 2021 で提案したアルゴリズム [9] と比較して，提案アルゴリズムが単一種類のエラーのみが発生する状況では，本稿で提案するアルゴリズムはより効果的であることを実証する．本稿で提案するアルゴリズム，SCIS 2021 で提案したアルゴリズム，CHES 2019 で提案されたアルゴリズムを**表 1** に示す．ただし，Type 1, Type2 の意味は 2.3.1 節で述べる．

2. 準備

本節では，本稿全体で用いる記号と用語，既存研究についてまとめる．

2.1 RSA 暗号 [6]

RSA 暗号 [6] の公開鍵を (N, e)，秘密鍵を d とする．このとき，素数 p, q に対して $N = pq$ であり，$ed \equiv 1 \pmod{\phi(N)}$ である．本稿では，N は 2048 ビット，p, q はそれぞれ 1024 ビットであるとする．OpenSSL における RSA 暗号の鍵生成では，$\gcd(p-1, e) = \gcd(q-1, e) = 1$ を満たす素数 p, q をそれぞれ生成する．特定バージョンの OpenSSL では，$\gcd(p-1, e)$, $\gcd(q-1, e)$ の計算に脆弱性のある Binary GCD アルゴリズムが使用されていた．また，OpenSSL では $e = 65537$ とあらかじめ定められており，特に断らない限り，本稿では $e = 65537$ とする．また，log の底は 2 とする．

2.2 Binary GCD アルゴリズム

Binary GCD アルゴリズムを Algorithm 1 に示す．Binary GCD アルゴリズムが入力に依存した挙動が脆弱性となる．いま，$\gcd(p-1, e)$ の計算を考える．まず，p は 1024 ビットであり，e は 17 ビットであるので，$p - 1 \gg e$ が成り立つ．さらに，$p-1$ と e はそれぞれ偶数と奇数である．これらの入力が満たす数学的関係のため，Algorithm 1 は u-loop, sub-step, u-loop, u-loop, sub-step, ... と繰り返すことになる．このとき，v-loop はしばらく実行されない．Aldaya らはこの挙動に注目して，Binary GCD アルゴリズムで実行される演算系列から素数 p, q を復元できることを示した [1]．また，OpenSSL における RSA 暗号の鍵生成における Binary GCD アルゴリズムの脆弱性に注目した研究として Weiser らの研究 [7] がある．

Algorithm 1: Binary GCD algorithm [1]

```
Input  : Integers a, b such that 0 < b < a
Output: Greatest common divisor of a, b
1  begin
2      u ← a, v ← b, i ← 0
3      while even(u) and even(v) do
4          u ← u/2, v ← v/2, i ← i + 1
5      while u ≠ 0 do
6          while even(u) do
7              u ← u/2                          /* u-loop */
8          while even(v) do
9              v ← v/2                          /* v-loop */
10         if u ≥ v then
11             u ← u − v                        /* sub-step */
12         else
13             v ← v − u
14     return v · 2^i
```

Binary GCD アルゴリズムにおける，2 による除算（u-loop）を L，減算（sub-step）を S と表記することにする．そこで，Binary GCD アルゴリズムで実行される演算の系列を LS 系列と呼ぶことにする．また，LS 系列における i 回目の S と，$i+1$ 回目の S の間に実行される L の個数を Z 系列 Z_i とする．ただし，Z_0 については LS 系列の先頭から 1 回目までの S までにある L の個数とする．例えば，LS 系列が LLSLLLLSLLLLLSL の場合は，Z 系列は $Z_0 = 2$, $Z_1 = 4$, $Z_2 = 5$ となる．これらの記号の定義は，[1] を参考にした．

2.2.1 Z 系列 Z_i の特徴

Z_i がどのような値を取るのかを考える．素数のビット列が十分にランダムであると仮定する．この仮定のもとで，Algorithm 1 の変数 u のビット列も十分にランダムである．すなわち，u のビットは 0 か 1 である確率が等しい．u-loop は u の下位ビットに 0 が連続しているときに実行される．これより，$\Pr(Z_i)$ について $\Pr(Z_i) = (1/2)^{Z_i}$ が成り立つ．

表 1　アルゴリズムの比較
Table 1　Comparison between algorithms

	対象とするエラー	Expand で用いる情報	Prune の基準
CHES 2019 [1]	Type 1	公開鍵とエラー付き系列	解候補に含まれるエラーの個数，連続するエラーの個数など
SCIS 2021 [9]	Type 1, 2	公開鍵のみ	解候補の重み付きレーベンシュタイン距離
本稿	Type 1	公開鍵のみ（ [9] と同一）	解候補の尤度ベースとノルムベースの損失値

2.3　既存研究 [1]

Aldaya らは，OpenSSL 1.1.0-1.1.0h, 1.0.2b-1.0.2o での RSA 暗号の鍵生成において安全でない Binary GCD アルゴリズムが使用されていることを指摘した [1]．さらに，LS 系列から，秘密鍵の素数を復元する攻撃を提案した．この攻撃では，まず，サイドチャネル攻撃である Flush+Reload 攻撃 [8] 及び Performance Degradation 攻撃 [2] を用いて LS 系列を収集する．しかし，これらの攻撃を用いて得られる情報にはエラーが含まれている．サイドチャネル攻撃によってエラー付きの LS 系列が得られる．

2.3.1　エラーの種類

Aldaya らの攻撃 [1] で入手したエラー付き LS 系列には，(a) プリエンプションによるエラーと (b) プリエンプションによらないエラーの二つの種類がある [1]．(a) プリエンプションによるエラーは，攻撃対象である Binary GCD アルゴリズムの実行に関連するプロセスや攻撃者に関連するプロセスの実行が一時的に中断されることにより発生するエラーである．(b) プリエンプションによらないエラーは，発生頻度が高い順に以下の通りである [1].

Type 1　S と S の間の L の個数が変わる（Z 系列のエラー）
Type 2　S が削除される
Type 3　S が余分に追加される

Aldaya らのエラー訂正アルゴリズムは Type 1 のエラーのみを訂正できるアルゴリズムである．一般に，含まれるエラーの種類が多いほど，エラー訂正は困難になる．

2.4　既存研究 [9]

我々が SCIS 2021 で提案したアルゴリズム [9] について説明する．Aldaya らのエラー訂正アルゴリズムは単一の種類のエラーしか訂正できないという問題点があった．そのため，他の種類のエラーが LS 系列に含まれていれば，エラーを訂正できず秘密鍵を復元することができない．そこで，[9] では，レーベンシュタイン距離を用いたエラー訂正アルゴリズムを提案した．このアルゴリズムは，Heninger–Shacham の手法 [4] と同じく，解候補を増やす Expand 関数と，候補を枝刈りする Prune 関数から構成される．Heninger–Shacham の手法 [4] と同じ Expand を用いる点は，Aldaya らのアルゴリズムとは異なる．さらに，Type 1 と Type 2 のエラーが含まれる状況でもエラー訂正できることを数値実験により示した．しかし，対応できるエラーの個数が非常に少ないという問題点があった．

3.　エラー訂正アルゴリズム

整数 a の二進数表現が $a_{n-1}a_{n-2}\dots a_0$ であるとき，a の i ビット目を $a[i] = a_i$ と書くことにする．なお，最下位ビットは $a[0]$ とする．また，素数 p についてのサイドチャネル攻撃で得られたエラー付き LS 系列を $\widetilde{\mathbf{LS}}_p$，素数 q についても同様に $\widetilde{\mathbf{LS}}_q$ とする．

3.1　提案するエラー訂正アルゴリズム

本稿で提案するエラー訂正アルゴリズムについて説明する．いま，素数の下位ビットに対する解候補の集合を $\mathcal{M}$ とする． [9] と同じく，Expand と Prune から構成される分枝限定法アルゴリズムを提案する．すなわち，公開鍵 N とエラー付き LS 系列を用いて，素数 p, q の解候補を生成し，解候補の集合 $\mathcal{M}$ に追加する（Expand）．解候補の集合 $\mathcal{M}$ がある基準にしたがって解候補の削減する（Prune）という方針である．Expand と Prune を解候補が目標とするビット長となるまで繰り返し，素数の復元を試みる．Prune の基準として，解候補の中で観測した系列であるエラー付き LS 系列と「最も近い」ものを残し，遠いものは削減するという方針をとる．この近さの定義として，損失関数を導入する．提案するアルゴリズムを Algorithm 2 に示す．公開鍵 (N, e)，エラー付き LS 系列 $\widetilde{\mathbf{LS}}_p, \widetilde{\mathbf{LS}}_q$，目標とする解候補のビット数 t，解候補の集合 $\mathcal{M}$ の要素数の上限 L を入力とする．出力は，解候補の集合である．本稿で提案するアルゴリズムは，Prune が SCIS 2021 のアルゴリズム [9] とは異なっている．

3.2　Expand

素数 p, q の下位ビットに対する解候補である p', q' が $i-1$ ビットであるとする．Heninger–Shacham の手法 [4] では，以下の式を用いて解候補のビット長を拡張していく．

$$p[i] + q[i] = (N - p'q')[i] \mod 2 \qquad (1)$$

$p[0] = q[0] = 1$ であるので，式 (1) を繰り返し用いて，$p[i], q[i]$ を計算できる．また，式 (1) を用いれば，必ず素数 p, q の下位ビットと一致する解候補が得られる．Algorithm 2 における関数 Expand では，集合 $\mathcal{M}$ に含まれる $i-1$ ビット長の各解候補 $\mathbf{b}$ に対して，式 (1) を用いて i ビット長の解候補を 2 個返す．すなわち，1 個の解候補か

ら新たな候補が 2 個生成されるため，解候補の個数は指数関数的に増加する．

3.3 Prune

目標とするビット長 t が大きければ，Algorithm 2 は現実的な時間で終了しない．そのため，解候補を適切な基準を用いて削減する必要がある．関数 Prune によって，解候補を削減する．本稿での Prune では，損失関数を導入して，各解候補に対して損失値を計算する．この損失値が小さい解候補を残すという方針をとる．以下の議論では，サイドチャネル攻撃などで観測された系列を $\mathbf{x}$，関数 Expand で得られた解候補を $\mathbf{b}$ とする．また，本稿では二種類の損失関数を導入する．一方は尤度ベースの損失関数であり，もう一方はノルムベースの損失関数である．

3.4 損失関数

観測系列 $\mathbf{x}$，解候補 $\mathbf{b} \in \mathcal{M}$ の損失関数 $\mathbf{Loss}(\mathbf{b}, \mathbf{x})$ を定義する．解候補の集合 $\mathcal{M}$ の要素数を削減するために，損失値が小さい L 個の解候補 $\mathbf{b}$ のみを残し，他の候補は削除することにする．本稿では尤度ベースの損失関数とノルムベースの損失関数を提案する．攻撃者の立場からすると，尤度ベースの損失関数では攻撃対象におけるエラーの確率分布を推定する必要がある．一方で，ノルムベースの損失関数では，攻撃者は攻撃対象におけるエラーについての事前知識を必要としない．これらの二つの損失関数の違いは数値実験で考察する．

3.4.1 尤度ベースの損失関数

まず，尤度ベースの損失関数のフレームワークを [5] に基づき説明する．$\mathbf{x}$ が得られたときに，解候補 $\mathbf{b} \in \mathcal{M}$ が得られる条件付き確率 $\Pr(\mathbf{b} \mid \mathbf{x})$ を考える．この確率が高い $\mathbf{b}$ が，解として尤もらしいと判断をする．Bayes の定理より

$$\Pr(\mathbf{b} \mid \mathbf{x}) = \frac{\Pr(\mathbf{x} \mid \mathbf{b}) \cdot \Pr(\mathbf{b})}{\Pr(\mathbf{x})}$$

が成り立つ．ここで，$\Pr(\mathbf{x})$ は，全ての解候補 $\mathbf{b}$ に対して共通であるため，順序には影響しない．そのため，これ以降，この項は無視することにする．よって，$\Pr(\mathbf{b} \mid \mathbf{x}) \cdot \Pr(\mathbf{b})$ が大きいほど，候補 $\mathbf{b}$ がもともとの正しい素数に対応した解候補である確率が高いということになる．

まず，$\Pr(\mathbf{x} \mid \mathbf{b})$ について考える．[5] では，$\mathbf{x}$ として，直接観測された素数を考え，$\mathbf{b}$ として，素数の下位ビットを持つ解候補を考えている．しかし，本稿で想定する攻撃の状況では，素数は直接は観測されずに，エラー付き LS 系列 $\widetilde{\mathbf{LS}}_p, \widetilde{\mathbf{LS}}_q$ が観測される．そこで，本稿では，$\mathbf{x} = (\widetilde{Z}_0, \widetilde{Z}_1, \ldots, \widetilde{Z}_{l-1})$ とする．また，$\mathbf{b}$ として Z 系列を考えて，$\mathbf{b} = (Z_0, Z_1, \ldots, Z_{l-1})$ とする．いま，$\widetilde{Z}_i$ の値は Z_i の値のみに依存して決まると仮定する．ここで，$\Pr(Z_i \to \widetilde{Z}_i)$ は，Z_i にエラーが発生し，$\widetilde{Z}_i$ となる確率とする．この仮定のもとで以下の式が成り立つ．

$$\begin{aligned}\Pr(\mathbf{x} \mid \mathbf{b}) &= \Pr(\widetilde{Z}_0, \ldots, \widetilde{Z}_{l-1} \mid Z_0, \ldots, Z_{l-1}) \\ &= \prod_{i=0}^{l-1} \Pr(\widetilde{Z}_i \mid Z_i) = \prod_{i=0}^{l-1} \Pr(Z_i \to \widetilde{Z}_i) \quad (2)\end{aligned}$$

したがって，$\Pr(\mathbf{x} \mid \mathbf{b})$ は，$\Pr(Z_i \to \widetilde{Z}_i)$ によって計算可能である．

次に，$\Pr(\mathbf{b})$ について考える．Z_i は無記憶な定常情報源から発生する系列であると仮定する．この仮定のもとで以下の式が成り立つ．

$$\Pr(\mathbf{b}) = \Pr(Z_0, \ldots, Z_{l-1}) = \prod_{i=0}^{l-1} \Pr(Z_i) \quad (3)$$

したがって，式 (2) と式 (3) より，以下の式が成り立つ．

$$\begin{aligned}\Pr(\mathbf{x} \mid \mathbf{b}) \cdot \Pr(\mathbf{b}) &= \left(\prod_{i=0}^{l-1} \Pr(Z_i \to \widetilde{Z}_i)\right) \cdot \left(\prod_{i=0}^{l-1} \Pr(Z_i)\right) \\ &= \prod_{i=0}^{l-1} \Big(\Pr(Z_i \to \widetilde{Z}_i) \cdot \Pr(Z_i)\Big) \quad (4)\end{aligned}$$

よって，式 (4) の値が大きい候補 $\mathbf{b}$ ほど，正しい候補である確率が高いといえる．ここで，$\Pr(Z_i)$ は，2.2.1 節の議論より $\Pr(Z_i) = (1/2)^{Z_i}$ であるので，攻撃者は既知である．一方で，$\Pr(Z_i \to \widetilde{Z}_i)$ は，攻撃者が推定し決定する必要がある．そのため，尤度ベースの損失関数は，攻撃者が $\Pr(Z_i \to \widetilde{Z}_i)$ を既知である前提で用いられる．

これまでの議論をまとめて，解候補 $\mathbf{b}$ の尤度ベースの損失関数を定める．まず，解候補から LS 系列を計算して Z 系列に変換する方法を述べる．入力となる素数 p の下位ビットが決まれば，$p-1$ と e を入力としたときの Binary GCD アルゴリズムの開始から一部まで挙動が決まる．解候補 $\mathbf{b}$ は素数の下位ビットに対応する p', q' をもつので，解候補を素数の下位ビットと考えれば LS 系列を一部だけ取り出すことができる．そこで，我々は SCIS 2021 において，解候補から LS 系列を取り出すことができるアルゴリズムを提案している [9]．このアルゴリズムを Algorithm 3 に示す．Algorithm 3 を用いて，解候補から LS 系列を計算することができる．その後，それぞれの LS 系列から Z 系列を計算し，その結果を $\{Z'^p_i\}_{i=0}^{l'_p-1}$, $\{Z'^q_i\}_{i=0}^{l'_q-1}$ とする．また，エラー付き LS 系列 $\widetilde{\mathbf{LS}}_p$, $\widetilde{\mathbf{LS}}_q$ からそれぞれ Z 系列を計算した結果を $\{\widetilde{Z}^p_i\}_{i=0}^{\widetilde{l}_p-1}$, $\{\widetilde{Z}^q_i\}_{i=0}^{\widetilde{l}_q-1}$ とする．解候補 $\mathbf{b}$ は，素数の下位ビットを持つ．そのため，素数の下位ビットから計算される LS 系列は，正しい素数から計算される LS 系列よりも短い．以上をもとに，尤度ベースの損失関数を以下のように定める．

$$\begin{aligned}\mathrm{Loss}^{\mathrm{ML}}(\mathbf{b},\mathbf{x}) \coloneqq & -\frac{1}{l'_p}\log\left[\prod_{i=0}^{l'_p-1}\Pr\left(Z'^p_i \to \widetilde{Z}^p_i\right)\cdot\Pr\left(Z'^p_i\right)\right] \\ & -\frac{1}{l'_q}\log\left[\prod_{i=0}^{l'_p-1}\Pr\left(Z'^q_i \to \widetilde{Z}^q_i\right)\cdot\Pr\left(Z'^q_i\right)\right] \\ = & -\frac{1}{l'_p}\sum_{i=0}^{l'_p-1}\left[\log\Pr\left(Z'^p_i \to \widetilde{Z}^p_i\right)+\log\Pr\left(Z'^p_i\right)\right] \\ & -\frac{1}{l'_q}\sum_{i=0}^{l'_p-1}\left[\log\Pr\left(Z'^q_i \to \widetilde{Z}^q_i\right)+\log\Pr\left(Z'^q_i\right)\right] \end{aligned} \tag{5}$$

ただし，$\log 0 = \infty$ とする．また，2.2.1 節の議論より，$\Pr\left(Z'^p_i\right) = (1/2)^{Z'^p_i}$, $\Pr\left(Z'^q_i\right) = (1/2)^{Z'^q_i}$ であるので，式 (5) は以下のようになる．

$$\begin{aligned}\mathrm{Loss}^{\mathrm{ML}}(\mathbf{b},\mathbf{x}) = & -\frac{1}{l'_p}\sum_{i=0}^{l'_p-1}\left[\log\Pr\left(Z'^p_i \to \widetilde{Z}^p_i\right)-Z'^p_i\right] \\ & -\frac{1}{l'_q}\sum_{i=0}^{l'_p-1}\left[\log\Pr\left(Z'^q_i \to \widetilde{Z}^q_i\right)-Z'^q_i\right] \end{aligned} \tag{6}$$

3.5 ノルムベースの損失関数

尤度ベースの損失関数は，攻撃者が $\Pr\left(Z_i \to \widetilde{Z}_i\right)$ を既知である必要がある．一般に，攻撃者は $\Pr\left(Z_i \to \widetilde{Z}_i\right)$ を既知であるとは限らない．$\Pr\left(Z_i \to \widetilde{Z}_i\right)$ を含めたエラーに関する確率分布の情報を全く必要としない，ノルムに基づいた損失関数を定義する．本稿では，L1 ノルムと L2 ノルムに基づく損失関数を定義する．

候補 $\mathbf{b}$ の L1 ノルムベースの損失関数を以下のように定める．

$$\mathbf{Loss}^{\mathrm{L_1}}(\mathbf{b},\mathbf{x}) \coloneqq \frac{1}{l'_p}\sum_{i=0}^{l'_p-1}|\widetilde{Z}^p_i - Z'^p_i| + \frac{1}{l'_q}\sum_{i=0}^{l'_q-1}|\widetilde{Z}^q_i - Z'^q_i| \tag{7}$$

SCIS 2021[9] では，Prune の基準として重み付きレーベンシュタイン距離を用いている．Type 1 のエラーのみに注目してレーベンシュタイン距離を用いることで，L1 ノルムベースの損失関数が導かれることを示す．文字列 $\mathbf{A}$ と文字列 $\mathbf{B}$ のレーベンシュタイン距離は，文字列 $\mathbf{A}$ を文字列 $\mathbf{B}$ に変換する際に必要な最小の操作の回数と定義される．ただし，操作として挿入，削除，置換の三つを考える．Type 1 のエラーのみに注目すれば，エラー付きの LS 系列に含まれる S の個数は変化しない．また，$\widetilde{Z}_i$ の値そのものはエラーの影響を受けるが，$\widetilde{Z}_i$ の長さは変化しない．いま，挿入と削除のコストを 1 とする．このとき，Type 1 のエラーのみが発生する状況で，エラー付きの LS 系列とエラーがない正しい LS 系列のレーベンシュタイン距離は，$\sum_i |\widetilde{Z}_i - Z_i|$ の値に等しい．すなわち，レーベンシュタイン距離を考えることで自然に式 (7) が導かれる．

Algorithm 2: Error correction algorithm

Input : Public key (N, e), erroneous LS sequences pair $(\widetilde{\mathbf{LS}}_p, \widetilde{\mathbf{LS}}_q)$, t and L.
Output: Set of candidates primes M.

```
begin
    p[0] ← 1, q[0] ← 1
    M ← {(p[0], q[0])}
    for Iterate t − 1 times do
        M ← ⋃_{b∈M} Expand(b, N)
        M ← Prune(M, e, L, LS̃_p, LS̃_q)
    return M
```

Algorithm 3: Algorithm for obtaining LS sequence from partial solution

Input : the n' bits candidate solution p' and public exponent $e = 2^m + 1\,(\exists m \in \mathbf{N})$
Output: LS sequence **LS**

```
begin
    LS ← ""
    a ← BitsToInt(p') + 2^{n'} − 1
    while a > 0 do
        while even(a) do
            a ← a/2
            LS ← LS + 'L'
        if a ≥ (e − 1) ∗ 2 then
            a ← a − e
            LS ← LS + 'S'
        else if a > 1 then
            a ← a − 1
            LS ← LS + 'S'
        else
            break
    return LS
```

次に，L1 ノルムベースな損失関数から派生して，L2 ノルムベースの損失関数を定める．候補 $\mathbf{b}$ の L2 ノルムベースの損失関数を以下のように定める．

$$\mathbf{Loss}^{\mathrm{L_2}}(\mathbf{b},\mathbf{x}) \coloneqq \frac{1}{l'_p}\sum_{i=0}^{l'_p-1}|\widetilde{Z}^p_i - Z'^p_i|^2 + \frac{1}{l'_q}\sum_{i=0}^{l'_q-1}|\widetilde{Z}^q_i - Z'^q_i|^2 \tag{8}$$

4. 数値実験

4.1 エラーモデル

数値実験で用いるエラーモデルとして，[9] と同じエラーモデルを用いることにする．ノイズ ϵ_i を確率変数とする．エラー付きの Z_i である $\widetilde{Z}_i$ を $\widetilde{Z}_i = Z_i + \epsilon_i$ とモデル化する．このとき，ϵ_i はそれぞれ独立に定められる．また，ϵ_i が従う確率分布の定義は以下のように定められる [9]．平均 0，標準偏差 σ の正規分布 $N(0, \sigma^2)$ の確率密度関数を

$g(x)$ とする．

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

確率 $\Pr(\epsilon_i = k)$, $k \in \mathbb{Z}$ を以下のように定義する．

$$\Pr(\epsilon_i = k) = \begin{cases} 0 & (|k| > Z_i \text{のとき}) \\ g(k)/C_i & |k| \leq Z_i \text{のとき}) \end{cases}$$

C_i は規格化定数とし，$C_i \coloneqq \sum_{|x| \leq Z_i} g(x)$ とする．

4.2 実験方法

本稿での数値実験の手順について説明する．

(1) OpenSSL で genpkey コマンドを用いて，RSA 暗号の鍵生成を行い素数 p, q を得る
(2) $\gcd(p-1, e)$, $\gcd(q-1, e)$ を Binary GCD アルゴリズムを用いて計算し，それぞれの LS 系列を得る
(3) それぞれの LS 系列に対し，4.1 節で述べたエラーモデルに基づきノイズを発生させることにより，エラー付き LS 系列を得る
(4) エラー訂正アルゴリズムによって目標ビットまでの素数の解候補の集合を列挙する
(5) 解候補の集合に正しい素数 p, q の目標下位ビットまでと一致するものがあれば，成功とする

Coppersmith の手法 [3] を用いれば，素数の連続した半分の下位ビットが既知であるとき，その素数を完全に復元することができる．すなわち，N が 2048 ビットの RSA 暗号では，素数 p, q のいずれかの下位 512 ビットを入手できれば，素数全体を復元することができる．さらに，素数全体を入手できれば，攻撃者は秘密鍵 d を容易に計算でき，この鍵ペアは安全でなくなる．したがって，本稿での実験では素数の下位 512 ビット以上の復元をエラー訂正アルゴリズムで行う．全体としての攻撃を想定すれば，t を決める際は，Coppersmith の手法 [3] による素数の残りのビットの復元も考慮する必要がある．Coppersmith の手法 [3] の実行時間，エラー訂正アルゴリズムの実行時間や復元率などから t を決める必要がある．

4.3 損失関数の計算方法

ノルムベースの損失関数の定義である式 (7), (8) によれば，特に事前知識，事前計算は必要とせずに損失値を計算することができる．その一方で，尤度ベースの損失関数の定義である式 (6) によれば $\Pr(Z'_i \to \widetilde{Z}_i)$ を計算する必要がある．これらの値を数値実験においてどのように計算するかを説明する．ただし，ここでは p, q の添字は省略して説明を行う．

4.3.1 $\Pr(Z'_i \to \widetilde{Z}_i)$

本稿では，攻撃者は Type 1 のエラーの確率分布が 4.1 節のエラーモデルに従うことを既知であるとする．ただし，σ の正確な値は必ずしも既知ではないとする．本稿では，$\Pr(Z'_i \to \widetilde{Z}_i)$ は 4.1 節のエラーモデルに基づいて計算を行う．そのため，事前にエラーモデルのパラメータである σ を推定する必要がある．正しい σ と区別するため，$\Pr(Z'_i \to \widetilde{Z}_i)$ の計算で用いる推定する標準偏差を σ' と表記することにする．この推定した σ' に基づいて $\Pr(Z'_i \to \widetilde{Z}_i)$ を計算する．

4.4 実験結果（成功確率）

まず，損失関数ごとの成功確率を検証する．素数の下位 512 ビットを復元することを目指して $t = 511$ とし $L = 2^{11}$ とする．

まず，尤度ベースの損失関数を用いたときの実験を行う．σ' の設定については以下の二つの状況を考える．

(1) 攻撃者が完全にエラーの分布をっている状況
(2) 攻撃者は正確な σ を知らない状況

(1) は，$\sigma' = \sigma$ の状況である．特に，(2) の状況としては，$\sigma' = 0.3, 0.6$ と攻撃者が推定した状況を考えることにする．尤度ベースの損失関数を用いたときの成功確率を**図 1** に示す．$\sigma' = \sigma, 0.3, 0.6$ として，それぞれの σ の値ごとに 100 回の実験を行った．横軸は σ であり，縦軸は成功確率である．成功確率は，$\sigma' = 0.3$ と設定した状況が高い結果となった．

次に，ノルムベースの損失関数を用いたときの成功確率を**図 2** に示す．L1 ノルムベースの損失関数と L2 ノルムベースの損失関数を用いて実験を行った．L1 ノルムベースの損失関数を用いた場合よりも L2 ノルムベースの損失関数を用いた場合の方が成功確率が高い結果となった．

尤度ベースの損失関数，ノルムベースの損失関数ともに，Type 1 のエラーの対して有効であることが確認できた．さらに，尤度ベースの損失関数を用いた場合の方が，ノルムベースの損失関数を用いた場合よりも，成功確率が高くなった．

次に，SCIS 2021 での結果と比較したグラフを**図 3** に示す．本稿で提案するアルゴリズムは，SCIS 2021 での結果よりも，成功確率が高いことが確認できる．

4.5 実験結果（訂正するエラーの個数）

4.5.1 本稿の実験

Aldaya らは，Coppermsith の手法 [3] での残りのビットの復元する際の効率性を考慮して，エラー訂正アルゴリズムでは素数の下位 522 ビットの復元を目標としている [1]．そこで，本稿のエラー訂正アルゴリズムと Aldaya らのアルゴリズムのエラー訂正能力を比較することを目的として実験を行う．素数の下位 522 ビットを復元することを目指して $t = 521$ とする．いま，出力に含まれる解候補から $\{Z'^p_i\}_{i=0}^{l'_p-1}, \{Z'^q_i\}_{i=0}^{l'_q-1}$ を計算したとする．エラー訂正アルゴリズムが訂正できたエラーの個数

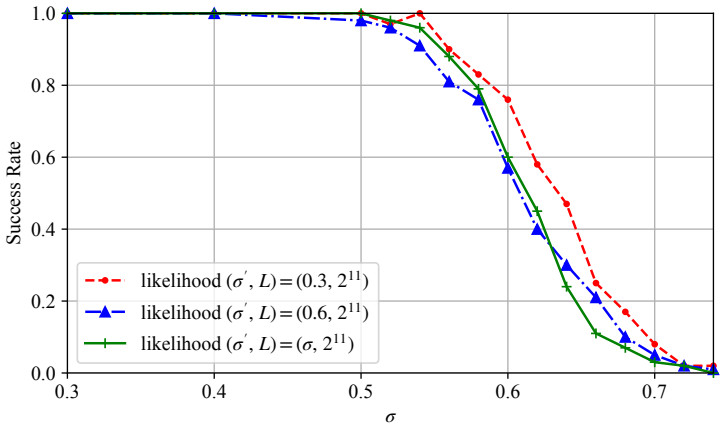

図 1 尤度ベースな損失関数を用いたときの成功確率
Fig. 1 Success rate when using a likelihood based loss function

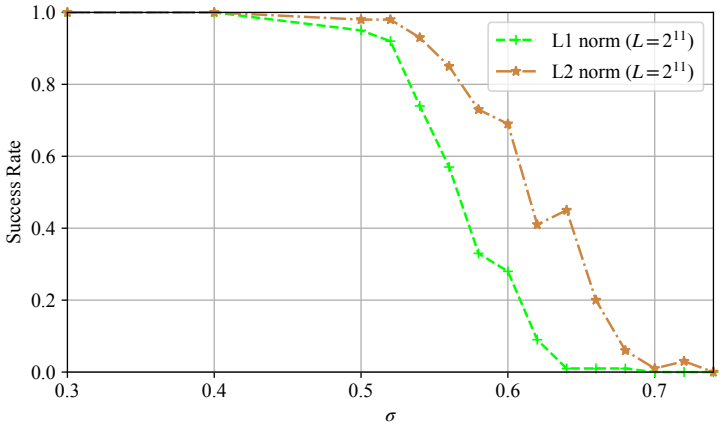

図 2 ノルムベースな損失関数を用いたときの成功確率
Fig. 2 Success rate when using a norm based loss function

を $\sum_{i=0}^{l'_p-1}|\widetilde{Z}_i^p - Z'^p_i| + \sum_{i=0}^{l'_q-1}|\widetilde{Z}_i^q - Z'^q_i|$ と定義する．尤度ベースとノルムベースのそれぞれで最も成功確率の高かった $(\sigma', L) = (0.3, 2^{11})$ と設定した損失関数と L2 ノルムベースの損失関数を用いて実験を行った．また，両者には $\sigma = 0.6$ としてエラーを発生させた同じエラー付き LS 系列のデータセットを入力として与えた．成功確率は尤度ベースの損失関数を用いた方が 72%であり，L2 ノルムベースの方が 58%であった．さらに，エラー訂正できたエラーの個数を箱ひげ図にして**図 4** にまとめる．

4.5.2 Aldaya らの実験 [1]

Aldaya らは，実際の計算機からサイドチャネル攻撃を用いてエラー付き LS 系列を入手している．そのエラー付き LS 系列を Aldaya らのエラー訂正アルゴリズムを用いて 522 ビットの素数を復元した．このとき，エラー訂正できたエラーの個数は最大で 108 であった [1]．

4.5.3 本稿と Aldaya らのアルゴリズムの比較

Aldaya らは実際にサイドチャネル攻撃で得られたエラー付き LS 系列を用いたのに対して，我々はエラーモデルによって人工的に生成したエラー付き LS 系列を用いた．そのため，単純にエラー訂正能力を比較することができない．その一方で，図 4 から，我々のエラー訂正アルゴリズムは 108 より多くのエラーが含まれる場合でもエラー訂正可能であることが確認できる．これより，本稿のアルゴリズムは Aldaya らのアルゴリズムと比較して，より有効であると言える．

5. まとめ

本稿では，RSA 暗号の鍵生成時における Binary GCD アルゴリズムの演算系列がエラー付きで得られた時に，秘密鍵を復元するアルゴリズムを提案している．異なる攻撃環境で有効となる 2 種類の損失関数を導入し，数値実験によりその有効性を確認した．一つは，尤度ベースの損失関数であり，高いエラー訂正能力を持つものの，攻撃者が生じるエラーの分布に関する知識を有している場合のみに有

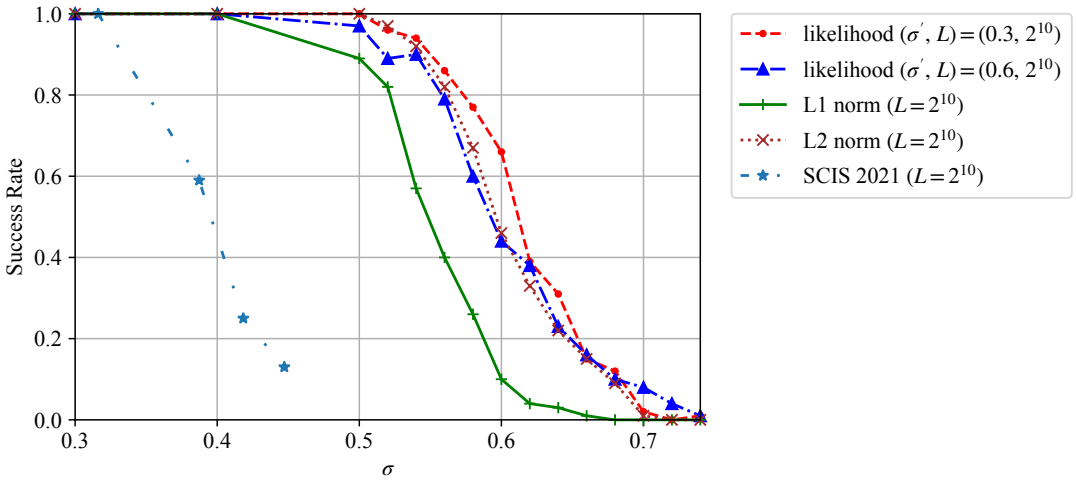

図 3　尤度ベースとノルムベース，SCIS 2021 [9] の比較
Fig. 3　Comparison between likelihood and norm, SCIS 2021 [9]

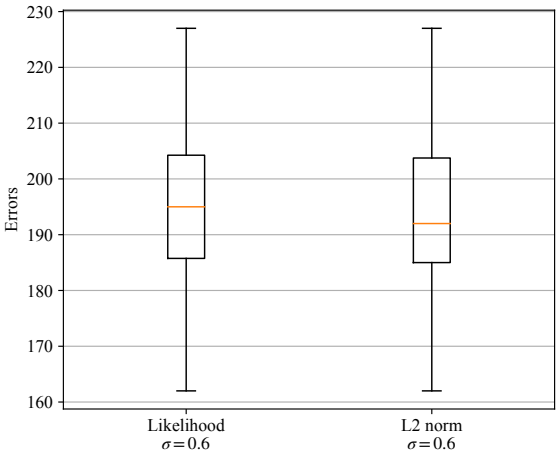

図 4　エラー訂正できたエラーの個数
Fig. 4　Counts of errors that error correction algorithms could handle with

効である．一つは，ノルムベースの損失関数であり，十分高いエラー訂正能力を持ちつつ，知識がない場合にも有効である．Aldaya らのアルゴリズム，SCIS 2021 で我々が提案したアルゴリズムなどの既存のアルゴリズムと比較して，Type1 のエラーのみが発生する状況では，高い性能を有することを確認した．

5.1　今後の課題

本稿では，提案アルゴリズムの有効性を数値実験により検証した．尤度ベースの損失関数を用いた場合では，$\sigma' = \sigma$ の場合よりも $\sigma' = 0.3$ の場合の方が成功確率が高い結果となった．$\sigma = \sigma'$ の場合の方が，攻撃者がもつ情報量は多いため，直感的には最も成功確率が高いはずである．今後の課題として，この点も踏まえて成功確率の理論的解析を進める予定である．さらに，Type 1 以外のエラーが含まれる状況下でも高い成功確率を実現できるアルゴリズムの提案も今後の課題とする．

謝辞　本研究の一部は，JST CREST JPMJCR14D6 および JPSP 科研費 JP21H03440 の支援の助成を受けて行われた．

参考文献

[1] Aldaya, A. C., García, C. P., Tapia, L. M. A. and Brumley, B. B.: Cache-timing attacks on RSA key generation, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 213–242 (2019).
[2] Allan, T., Brumley, B. B., Falkner, K., Van de Pol, J. and Yarom, Y.: Amplifying side channels through performance degradation, *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 422–435 (2016).
[3] Coppersmith, D.: Finding a Small Root of a Univariate Modular Equation, *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'96, Springer-Verlag, p. 155–165 (1996).
[4] Heninger, N. and Shacham, H.: Reconstructing RSA private keys from random key bits, *Advances in Cryptology - CRYPTO 2009*, Springer, pp. 1–17 (2009).
[5] Kunihiro, N.: *Mathematical Approach for Recovering Secret Key from Its Noisy Version*, Mathematics for Industry, pp. 199–217, Springer Singapore (2018).
[6] Rivest, R. L., Shamir, A. and Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, Vol. 21, No. 2, pp. 120–126 (1978).
[7] Weiser, S., Spreitzer, R. and Bodner, L.: Single trace attack against RSA key generation in Intel SGX SSL, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 575–586 (2018).
[8] Yarom, Y. and Falkner, K.: FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack, *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 719–732 (2014).
[9] 谷　健太，國廣　昇：RSA 暗号の鍵生成における Binary GCD アルゴリズムの安全性評価，2021 年暗号と情報セキュリティシンポジウム（SCIS2021）．