Sliding Window法からの漏洩情報を用いる秘密鍵復元アルゴリズムの改良

平成 30 年度

修士論文

Sliding Window 法からの漏洩情報を用いる秘密鍵復元アルゴリズムの改良

47176106　黄　暁萱

指導教員　國廣　昇

2019 年 1 月 29 日

東京大学大学院新領域創成科学研究科

複雑理工学専攻

概要

高速にべき乗を計算できる Sliding Window 法を用いて暗号を復号する時，サイドチャネル攻撃により秘密鍵に関する部分的な情報を取得することができる．これらの情報を用いて秘密鍵を復元する研究が行われている．2017 年に，Bernstein らは部分的に秘密鍵のビットを復元できるビット復元ルールを提案した．その上，Heninger-Shacham アルゴリズムを適用することにより，25% の確率で 1024-bit RSA の秘密鍵の復元に成功する．しかし，彼らのルールは挙動が明確ではないところがあった．そこで，本稿では，まず新たなビット復元ルールを提案し，Bernstein らより多くのビットを復元できた．その上，Heninger-Shacham アルゴリズムを適用し，秘密鍵の復元率を 29% まで向上した．Heninger-Shacham アルゴリズムでは，全て CRT-RSA の制約式を満たす秘密鍵の候補を探索した．残り未知のビットが 0 か 1 となる確率が調べられなかった．そこで，ビット復元ルールを適用した後，残り未知ビットの分布を調べるため，本稿では元の SM 時系列に矛盾しないビット列をランダムサンプリングできる Random Sampling アルゴリズムを提案した．その結果，分布に偏りがあることを判明した．この事実を踏まえて，新たな秘密鍵復元アルゴリズムを提案した．秘密鍵の復元率は 77% まで向上した．

キーワード　サイドチャネル攻撃，秘密鍵復元，Sliding Window 法，CRT-RSA

目次

第 1 章

はじめに

1.1 研究背景

安全通信で使われている公開鍵暗号は，誰でも見ることができる公開鍵と真の復号者のみが持つ秘密鍵により構成されている．しかし，暗号のアルゴリズムを計算機上に実装した際に，物理的な手段により秘密鍵に関する情報の一部を知ることが可能である．この漏洩した情報を用いて，秘密鍵全体を復元する研究が多く行われている．この公開されている情報のみではなく，他の秘密情報に関する情報を用いて攻撃する手法はサイドチャネル攻撃と呼ばれる．

CRT-RSA 暗号 [2] は公開鍵暗号である RSA[11] 暗号の一種である．中国人剰余定理を使い，高速に復号できる特徴も持ち，暗号プロトコルに広く使われている．また，復号する時，べき乗算を高速に計算するために，Sliding Window 法を利用することがある．Sliding Window 法はパラメータ w を用いて，秘密鍵を変形する上で 2 乗算（S）と掛け算（M）を繰り返すことにより行われる．秘密鍵は，2 乗算と掛け算が行う順番の列で表すことができる．これを SM 時系列と呼ぶ．しかし，実装する時に，サイドチャネル攻撃を受ける可能性がある．復号における実行時間 [5] や，消費電力 [6] など物理的な情報により，Sliding Window 法での 2 乗算と掛け算を行った順番の履歴を取得することができる．つまり，秘密鍵の SM 時系列をサイドチャネル攻撃により，得ることができる．

1.2 既存研究

2017 年，Bernstein ら [1] は Flush+Reload[13, 14] により，SM 時系列を取得し，秘密鍵復元の研究を行った．Flush+Reload とは，サイドチャネル攻撃中のキャッシュ攻撃の一種類である．攻撃者は秘密鍵を持つ正当な復号者が復号する際に，ターゲットとしたメモリ領域にアクセスしたか否かを利用し，どのタイミングでどのメモリ領域にアクセスしたかを調べることにより，秘密鍵の SM 時系列を得ることができる．Bernstein らは SM 時系列から，部分的に秘密鍵のビットを復元できるビット復元ルールを提案した．その上，Heninger-Shacham アルゴリズム [4] を用い，鍵全体の復元に成功した．

Heninger-Shacham アルゴリズムは，秘密鍵を探索するアルゴリズムである．ビット復元

ルールを適用した後，残りの未知ビットに対して，木を生成し，枝刈りをして行く．木を生成する時に，注目するビットは 1 になる確率と 0 になる確率は，1/2 であると暗黙のうちに仮定していた．実際に，残りの未知ビットは，全て 1/2 の確率で 1 か 0 になるかどうかは，調べられていなかった．

1.3　研究成果

Bernstein らのルールの挙動が不明確なところがあった．その点に対して，本稿では，まず新たなビット復元ルールを提案した．鍵全体の復元率は Bernstein らの 25% から 29% まで向上した．

次に，ビット復元ルールを適用した後，残りの未知ビットが 0 になる確率を評価するため，SM 列に矛盾しないビット列を生成するランダムサンプリングアルゴリズムを提案した．実験結果により，全ての未知ビットが確率 1/2 で 1 か 0 になるには限らず，大きな偏りがあることを判明した．

さらに，高い確率で 1 になるビットを 1 として設定した上で，Heninger-Shacham アルゴリズムを用いることにより，1 ビットの反転が許される新たな鍵復元アルゴリズムを提案した．従来より高い確率で鍵全体の復元に成功した．具体的に，$w = 4$，90% の確率で 1 になるビットを 1 として設定する時に，復元率は従来の 29% から 54% まで向上した．

1.4　本論文の構成

本稿では，まず，2 章では公開鍵暗号である RSA と CRT-RSA，高速にべき乗を計算できる Sliding Window 法とサイドチャネル攻撃を紹介する．3 章では，秘密鍵復元の既存研究となる Bernstein ら [1] が提案したビット復元ルールと，Heninger-Shacham アルゴリズム [4] を紹介する．4 章では，本研究で提案した新たなビット復元ルールを紹介する．最後に，本研究で提案した新たな鍵復元アルゴリズムを紹介する．

第 2 章

研究背景

本章では，研究背景となる公開鍵暗号，Sliding Window 法とサイドチャネル攻撃を順番に説明する．

2.1 公開鍵暗号

通信者 A は通信者 B にメッセージを送る際，第三者に情報が漏れないように通信するために，公開鍵暗号を使用する．公開鍵暗号は鍵生成アルゴリズム，暗号化アルゴリズム，復号アルゴリズムより構成される．公開鍵暗号の鍵は，公開鍵と秘密鍵より構成される．公開鍵とは，公開される情報であり，誰でも入手することができる．暗号化に用いると，誰でも暗号文を作ることができる．秘密鍵は，正規の復号者しか入手できない情報である．復号に用いると，真のメッセージが得られる．

公開鍵暗号の安全性を確かめるため，公開される情報のみから，秘密情報を求める研究が行われる．暗号の安全性は，解くことが困難な数学問題により保証される．安全な暗号は，期待される時間内で秘密情報を求められない．

本章では，典型的な公開鍵暗号の一つである RSA 暗号と CRT-RSA 暗号を紹介する．本稿では CRT-RSA 暗号方式の安全性について議論する．

2.1.1 RSA 暗号

RSA 暗号は 1978 年に提案された公開鍵暗号方式である．大きい合成数の素因数分解問題の困難性を安全性の根拠とする．二つ大きい素数 p と q が与えられた際，その積 $N = pq$ は簡単に計算できる．しかし逆計算，つまり二つ大きい素数の積である N が与えられた際，p と q を求めることが難しい．素数が十分に大きいと設定する場合，その積となる合成数の素因数分解が難しいと信じられる．これが RSA 暗号の安全性の根拠である．現状，RSA 暗号は SSL/TLS などでよく使われている．サーバ証明書の公開鍵サイズは 2048 ビット以上であることが推奨されている．以下，RSA 暗号アルゴリズムを説明する．

鍵生成：まず異なる奇素数 p と q を選ぶ．次に，N を $N = pq$ とする．さらに，$e, d \in$

$\mathbb{Z}^*_{(p-1)(q-1)}$ を $ed \equiv 1 \bmod (p-1)(q-1)$ を満たすように選ぶ．公開鍵を (N, e)，秘密鍵を (p, q, d) とする．

暗号化：平文 m に対して，暗号文 $c = m^e \bmod N$ を出力する．

復号：平文 $m = c^d \bmod N$ を出力する．

2.1.2　CRT-RSA 暗号

CRT-RSA 暗号は RSA 暗号の一種である．秘密鍵として，さらに d_p, d_q を使用する．鍵生成の段階で，$d_p := d \bmod p - 1$，$d_q := d \bmod q - 1$，$q_p = q^{-1} \bmod p$ を計算し，公開鍵を (N, e)，秘密鍵を (p, q, d, d_p, d_q, q_p) とする．CRT-RSA の暗号化は標準的な RSA と同じである．復号する際，

$m_1 = c^{d_p} \bmod p$，$m_2 = c^{d_q} \bmod q$ を計算する．$m = m_1 \bmod p$，$m = m_2 \bmod q$ となる m を中国人剰余定理を用いることにより計算する．CRT-RSA 暗号の復号アルゴリズムは Algorithm 1 で与えられる．

Algorithm 1 CRT-RSA 暗号の復号

Input: 暗号文 $c \in \mathbb{Z}_N$，公開鍵 $(N, e) \in \mathbb{N} \times \mathbb{Z}_N$，秘密鍵 (p, q, d, d_p, d_q, q_p)

Output: 平文 $m \in \mathbb{Z}_N$

$m_1 = c^{d_p} \bmod p$，$m_2 = c^{d_q} \bmod q$

$h = (m_1 - m_2) q_p \bmod p$

$m = m_2 + qh$

return m

本稿では，e の値を，実際に使われている $2^{16} + 1$ とする．また，公開鍵と秘密鍵は $k, k_p, k_q \in \mathbb{Z}$ におき，以下の式を満たす．

$$N = pq \tag{2.1}$$
$$ed = 1 + k(p-1)(q-1) \tag{2.2}$$
$$ed_p = 1 + k_p(p-1) \tag{2.3}$$
$$ed_q = 1 + k_q(q-1) \tag{2.4}$$

2.2　Sliding Window 法

Sliding Window べき乗算は高速に $b^d \bmod N$ を計算する計算方法である．実社会では RSA 暗号や楕円曲線暗号 [9] でよく使われている．本稿では，CRT-RSA 暗号の復号に使われることを前提として議論する．これから，計算方法について述べる．まず，べき指数を windowed form に変形してから計算を行う．Windowed form に変形する際，w ビットごとで 2^w 進数に変換する．この w ビットを本稿ではブロックと呼ぶ．次に，底に対する掛け算と 2 乗算を行

うことにより，べき乗を計算する．本節では windowed form に変形する方法を説明した後，Sliding Window 法の手順を説明する．

2.2.1 Windowed Form

Windowed form に変形するには，Left-to-Right と Right-to-Left 二つ変形方法がある．以下，この二つ変換方法を説明する．

まず，Left-to-right 変形ではビット列を左から右に読み込む．上位ビットから下位ビットまでブロックを読み込んでいくが，ブロック内に下位ビットが連続な 0 である場合，0 を変換しないように，ブロック内の最下位非 0 ビットまでの部分のみ 2^w 進数へ変換する．また，注目するブロックの変換が完了し，次のブロックを読み込む時，0 を飛ばし，初めて 1 になるところから次のブロックを読み込む．

$w = 3$ の時，$d = 185$ に対する変形例を示す．d をビット列で表すと，10111001 となる．まず上位から 3 ビットの 101 を読み取り，005 に変換する．次に，110 を読み取る時，一番後ろのビットが 0 なので，0 を含まないように 11 のみを 03 へ変換する．最後，00 を飛ばして 1 から読み取る．そして，5 の前の 0 を省略し，$\boxed{101}\boxed{11}00\boxed{1} \rightarrow 503001$ となる．

それに対して，Right-to-Left 変形ではビット列を右から左に読み込むが，各ブロックを変形する際，ブロックの上位ビットから下位ビットまで変形する．ブロック内に上位ビットが連続な 0 である場合は，実質にこれらの 0 が変形されていない．また，Left-to-Right 変換と同じ，次のブロックを取ろうとする際，0 を飛ばし，初めて 1 になるところから次のブロックを読み込む．例えば，$d = 185, w = 3$ の時，下位から 001 を読み取り，001 になる．次に 111 を読み取り，007 に変換する．最後，0 を飛ばして 1 を読み取る．そして，$\boxed{1}0\boxed{111}\boxed{001} \rightarrow 10007001$ となる．

2.2.2 Left-to-Right 変形における Sliding Window 法

本節では Left-to-Right 変形における Sliding Window 法の手順を説明する．

b を底，d をべき指数，N を法とする．まず，w を選定する．現実で $w = 4$，$w = 5$ がよく使われている．次に，べき指数 d を windowed form$(d_i \cdots d_1)$ に変形する．さらに，b，b^3，b^5，$b^7 \cdots$，$b^{2^w-1} \pmod N$ の値をメモリに格納する．次に，d の windowed form の上位桁から下位桁まで，演算を行う．d_i が 0 の場合，2 乗算のみ行う．d_i が 0 以外の場合，2 乗算を行った後，掛け算を行う．掛け算を行う際，格納した与えを直接に呼び出して使う．まとめると，Algorithm 2 となる．

Algorithm 2 の 14 と 15 行目は **S**quaring と **M**ultiplication を行う．本稿では，これらの演算が行う順番 **S** と **M** の列で表すことにする．この列を SM 時系列と呼ぶ．前述の例を説明すると，d の windowed form が 503001 となる場合，d の SM 時系列は SMSSMSSSM となる．

Algorithm 2 Sliding Window 法 [8]

Input: 整数 p, b と d を表現するビット列 $(d_n d_{n-1} \cdots d_1)$
Output: $a \equiv b^d (\mathrm{mod} p)$.
1 : **procedure** MOD_EXP(b, d, p)
2 : $b_1 = b$, $b_2 = b^2$, $a = 1$, $z = 0$
3 : **for** $i = 1$ **to** $2^{w-1} - 1$ **do**
4 : $b_{2i+1} = b_{2i-1} \cdot b_2 \bmod p$
5 : $i = n$
6 : **while** $i \neq 1$ **do**
7 : $z = z$+COUNT_LEADING_ZEROS$(d_i \cdots d_1)$
8 : $i = i - z$
9 : $l =$min(i, w)
10 : $u = d_i \cdots d_{i-l+1}$
11 : $t =$COUNT_TRAILING_ZEROS(u)
12 : $u =$SHIFT_RIGHT(u, t)
13 : **for** $j = 1$ **to** $z + l - t$ **do**
14 : $a = a^2 \bmod p$
15 : $a = a \cdot b_u \bmod p$
16 : $i = i - l$
17 : $z = t$
18 : **return** a
19 : **end procedure**

2.3　サイドチャネル攻撃

2.3.1　概要

公開鍵暗号に対する標準的な解読手法は，公開されている情報のみを用いることにより，秘密情報を求める．暗号の安全性は数学問題の困難さにより守られ，標準的な解読は困難な数学問題を解くことに帰着できる．しかし，現実で暗号を実装する際，物理的に安全性に対する脅威が存在している．平文や暗号文など公開されている情報のみではなく，他の秘密情報に関する情報も取り盗まれる可能性がある．例えば，暗号処理を行う装置が発する電磁波 [3]，電力量 [6] や処理時間 [5] の違いなどを観察することで秘密鍵に関する漏洩情報が得られる．このように物理的手段で秘密情報を得ようとする暗号解読手法は，サイドチャネル攻撃と呼ぶ．サイドチャネルとは、正規の入出力経路ではないことを意味しており，暗号本体のアルゴリズムとは異なる副次的情報であることからこのように呼ばれている．

2.3.2 Flush+Reload

本稿で既存研究となる Bernstein ら [1] の研究は Flush+Reload[13, 14] により CRT-RSA 秘密鍵の SM 時系列を得た上，研究を行った．Flush+Reload とは，サイドチャネル攻撃中のキャッシュ攻撃の一種類である．キャッシュ攻撃では，攻撃者はまずあるキャッシュ領域をクリアする．正当の復号者が復号の動作を完了した後，攻撃者は再びそのキャッシュ領域へアクセスする．アクセス時間を測ることにより，復号者はターゲットとしたメモリ領域にアクセスしたか否かを観測できる．Flush+Reload は全ての CPU コアで共有された領域 L3 キャッシュを対象とし，サイドチャネル攻撃を行い，秘密鍵の SM 時系列を得ることができる．

第 3 章

既存研究

3.1 Bernstein らの秘密鍵復元の研究

2017 年，Bernstein らは d_p, d_q の SM 時系列から，1024-bit RSA の秘密鍵の復元に成功した．彼らはまず，ビット復元ルールを提案し，秘密鍵の SM 時系列から一部のビットを復元できた．さらに，部分的なビットが知られた鍵を Heninger-Shacham アルゴリズム [4] に適用することにより，1024-bit RSA の秘密鍵全体を復元できた．本章では，まず，Bernstein らが提案したルールを説明する．さらに，彼らが行った実験の結果を考察する．

3.1.1 ビット復元ルールの説明

Bernstein らは秘密鍵の SM 時系列から 2 進数展開のビット列を得るため，4 つのルールを提案した．x は 0 か 1 が確定できないビットを表す．$\underline{*}$は掛け算を行った位置を表す．x^i は連続する x の個数が i 個であることを表す．以下，2 乗算のみ行ったビットを 2 乗算ビットと呼び，2 乗算を行ってから掛け算を行ったビットを掛け算ビットと呼ぶこととする．各ルールは全て上位ビットから下位ビットへ変換していく．

- **前処理： SM$\rightarrow\underline{\mathbf{x}}$，S$\rightarrow$ x**
  SM は掛け算ビットを意味し，S は 2 乗算ビットを意味する．
- **Rule 0 :** $\underline{\mathbf{x}}\rightarrow\underline{1}$
  掛け算ビットを$\underline{1}$とする．
- **Rule 1 :** $\underline{1}\mathbf{x}^i\underline{1}\mathbf{x}^{w-i-1}\rightarrow\underline{1}\mathbf{x}^i\underline{1}0^{w-i-1}$ $(0 \leq i < w-1)$
  掛け算ビットの次のビットから，新しい w サイズのブロックを取り始め，w ビット以内に掛け算ビットが現れた場合，掛け算ビットの後ろの $w-i-1$ ビットは全部 0 で埋め込む．
- **Rule 2 :** $\mathbf{x}\mathbf{x}^{w-2}\underline{11}\rightarrow 1\mathbf{x}^{w-2}\underline{11}$
  掛け算ビットが二つ連続する時，前の掛け算ビットが所属するブロックの先頭ビットを 1 とする．
- **Rule 3 :** $\underline{1}\mathbf{x}^i\mathbf{x}^{w-1}\underline{1}\rightarrow\underline{1}0^i\mathbf{x}^{w-1}\underline{1}$

後の掛け算ビットが所属するブロックに含まれないビットを 0 で埋め込む．

$w = 4$ の例を挙げる．SMSSSSSSMSSSMSSSSSSMSMSSSSSSMSSSSSSM の場合は，
前処理：$\underline{\mathrm{x}}$xxxxx$\underline{\mathrm{x}}$xx$\underline{\mathrm{x}}$xxxx$\underline{\mathrm{xx}}$xxxxx$\underline{\mathrm{x}}$xxxxx$\underline{\mathrm{x}}$
Rule 0：$\underline{1}$xxxxx$\underline{1}$xx$\underline{1}$xxxx$\underline{11}$xxxxx$\underline{1}$xxxxx$\underline{1}$
Rule 1：$\underline{1}$xxxxx$\underline{1}$xx$\underline{1}$0xxx$\underline{11}$000xx$\underline{1}$xxxxx$\underline{1}$
Rule 2：$\underline{1}$xxxxx$\underline{1}$xx$\underline{1}$01xx$\underline{11}$000xx$\underline{1}$xxxxx$\underline{1}$
Rule 3：$\underline{1}$00xxx$\underline{1}$xx$\underline{1}$01xx$\underline{11}$000xx$\underline{1}$00xxx$\underline{1}$
となる．

Sliding Window 法を根拠とし，上記の Rule 0〜3 は確実にビットを正しく復元できる．これから，各ルールがビットが復元できる理由を説明する．

- **Rule 0 :** $\underline{\mathbf{x}} \rightarrow \underline{1}$
  掛け算ビットは一つブロックの最下位非 0 ビットであるため，1 となる．
- **Rule 1 :** $\underline{1}\mathbf{x}^i\underline{1}\mathbf{x}^{w-i-1} \rightarrow \underline{1}\mathbf{x}^i\underline{1}0^{w-i-1}$ $(0 \leq i < w-1)$
  先頭の $\underline{1}$ は前位ブロックの最下位ビットである．$\mathrm{x}^i\underline{1}\mathrm{x}^{w-i-1}$ は w ビットを持ち，下位ブロックである．2 番目の $\underline{1}$ は下位ブロックで最下位非 0 ビットであるため，後ろのビットは全て 0 となる．
- **Rule 2 :** $\mathbf{x}\mathbf{x}^{w-2}\underline{11} \rightarrow 1\mathbf{x}^{w-2}\underline{11}$
  前の掛け算ビットが所属するブロックの最下位ビットとなるため，所属ブロックの先頭は必ず 1 となる．もし先頭ビットが 0 であれば，後ろの掛け算ビットもブロックに入り，一つブロックの中で掛け算ビットが一つしかないことに矛盾する．
- **Rule 3 :** $\underline{1}\mathbf{x}^i\mathbf{x}^{w-1}\underline{1} \rightarrow \underline{1}0^i\mathbf{x}^{w-1}\underline{1}$
  Sliding Window 法では，次のブロックは初めて 0 ではないビットから読み取るため，ブロックとブロックの間に全て 0 となる．

3.1.2 ビット復元率の考察

前節のビット復元ルールにより，$w = 4$ の時に 1,000,000 回以上実行した結果は以下のようになる．1024-bit RSA の d_p, d_q は Left-to-Right 変換を行った時，平均 251 ビット (49%) が復元できた．$w = 5$ の時に，2048-bit RSA の秘密鍵は，平均 425 ビット (41.5%) が復元できた．Right-to-Left 変換を行った時に，平均 204 ビット (40%) 復元できた．Left-to-right 変換は right-to-left 変換より，多くの情報が漏洩することを示した．Right-to-left 変換では，秘密鍵の windowed form での二つ非零桁の間に，少なくとも $w-1$ 個 0 がある．対照的に，Left-to-Right 変換では，windowed form での二つ非零桁の間の 0 は何個でも取りうる．間の 0 が $w-2$ 個以下の非零桁は，Right-to-Left 変換より多くのビットを漏洩する．しかし，d_p, d_q の 50% 以上のビットを知らないと，Heninger-Shacham アルゴリズム [4] による秘密鍵の復元が困難であることが知られている [10, 7].

さらに，Bernstein らは $w = 4$ と $w = 5$ の時，Heninger-Shacham アルゴリズム [4] を用い，鍵全体を復元する実験を行った．実験結果は以下のようになった．

$w = 4$ の時の実験結果：

彼らの実験では，1024-bit RSA における秘密鍵復元実験を 500,000 回実行した．秘密鍵候補が 1,000,000 個以上になる場合，アルゴリズムは停止し，秘密鍵を復元できなかったと見なす．Rule 0-3 により，平均的に 251 ビット (49%) を復元できた．50% 以上のビットが復元できた回数は総回数の 32% であった．さらに，Heninger-Shacham アルゴリズムにより秘密鍵を完全に復元できた回数は総回数の 28% であった．

$w = 5$ の時の実験結果：

2048-bit RSA における秘密鍵復元実験を 500,000 回実行した．Rule 0-3 により，平均的に 41% のビットが復元できた．しかし，秘密鍵を完全に復元するには一度も成功しなかった．これは平均値 41% は閾値 50% との距離が大きいからである．

3.2　Heninger-Shacham アルゴリズム

Heninger-Shacham アルゴリズムは秘密鍵の既知のビットから，未知のビットを求めるアルゴリズムである．本節では，このアルゴリズムを説明する．以下，本アルゴリズムで用いる記号を定義する．非負整数 x の二進数展開は $x_{n-1}x_{n-2}\cdots x_0$ である時，x の i 番目のビットを $x[i] = x_i (0 \leq i \leq n-1)$ と表す．また，$\tau(x)$ を $2^M | x$ となる最大の $M \in \mathbb{Z}$ と定義する．さらに，

$$\mathrm{Slice}[i] = (p[i], q[i], d[i + \tau(k)], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)])$$

とする．このアルゴリズムは，まず公開されている情報から，Slice[0] を求める．探索の段階では，Slice$[i-1]$ から Slice$[i]$ を求め，制約条件に満たさない候補を枝刈りする．最後に残される候補の中で，秘密鍵を決定する．

3.2.1　初期状態

本節では，木の根となる Slice[0] を求める．$(\tau(k), \tau(k_p), \tau(k_q))$ を求めるために，(k, k_p, k_q) を知る必要がある．まず，(k, k_p, k_q) について述べる．

まず，式 (2.2) より，k は下記の関係式を満たす．

$$0 < k < \frac{ed}{(p-1)(q-1)} < e$$

k は整数なので，候補が $e - 1 = 2^{16}$ 個とある．

次に，式 (2.3) と式 (2.4) より，k_p, k_q は下記の関係式を満たす．

$$0 < k_p < \frac{ed_p}{p-1} < e$$

$$0 < k_q < \frac{ed_q}{q-1} < e$$

故に，k_p, k_q は $0 < k_p < e, 0 < k_q < e$ を満たす整数である．

また，式 (2.1)，(2.2)，(2.3)，(2.4) の両辺を $\mod e$ をとると，

$$N \equiv pq \mod e \tag{3.1}$$
$$0 \equiv 1 + k(p-1)(q-1) \mod e \tag{3.2}$$
$$0 \equiv 1 + k_p(p-1) \mod e \tag{3.3}$$
$$0 \equiv 1 + k_q(q-1) \mod e \tag{3.4}$$

となる．式 (3.2) の両辺に $k_p k_q$ をかけると，

$$\begin{aligned} & 0 \equiv k_p k_q + k k_p (p-1) k_q (q-1) \mod e \\ \Leftrightarrow\ & 0 \equiv k_p k_q + k \mod e \\ \Leftrightarrow\ & k \equiv -k_p k_q \mod e \end{aligned} \tag{3.5}$$

となる．式 (3.5) を式 (3.2) に代入すると，

$$\begin{aligned} & 0 \equiv 1 + k(N-p-q+1) \mod e \\ \Leftrightarrow\ & 0 \equiv 1 + k(N-1) - k(p-1) - k(q-1) \mod e \\ \Leftrightarrow\ & 0 \equiv 1 + k(N-1) + k_p k_q (p-1) + k_p k_q (q-1) \mod e \\ \Leftrightarrow\ & 0 \equiv 1 + k(N-1) - k_q - k_p \mod e \\ \Leftrightarrow\ & 0 \equiv k_p + k(N-1)k_p - k_p^2 - k_p k_q \mod e \\ \Leftrightarrow\ & k_p^2 - (k(N-1)+1)k_p - k \equiv 0 \mod e \end{aligned} \tag{3.6}$$

となる．上記の式で，k_p を k_q に置き換えても成り立つ．故に，k の与えに応じて，k_p, k_q は下記二次方程式の解となる．

$$x^2 - (k(N-1)+1)x - k \equiv 0 \mod e \tag{3.7}$$

上記方程式の解は $(x_1, x_2) \in \mathbb{Z}_e \times \mathbb{Z}_e$ となる時，(k_p, k_q) は (x_1, x_2) あるいは (x_2, x_1) となる．

一つの k に対して，2 通りの (k_p, k_q) が存在する．故に，(k, k_p, k_q) は合計 $2(e-1)$ 通りである．$2(e-1)$ 個の木を作り，それぞれの木を探索する必要がある．以後のステップでは，一つの木を注目し，(k, k_p, k_q) を既知情報として議論を進める．

次に，Slice[0] を求める．まず，p, q は奇素数であるので，$p[0] = 1$，$q[0] = 1$ となる．次に，(k, k_p, k_q) が知られる以上，下記の式が自明になる．

$$\text{Slice}[0] = (1, 1, d[\tau(k)], d_p[\tau(k_p)],\ d_q[\tau(k_q)])$$

本節では，$d[\tau(k)], d_p[\tau(k_p)],\ d_q[\tau(k_q)])$ が実際に計算できることを説明する．

p, q が奇素数である性質より，$2|p-1$，$2|q-1$ が成り立つ．ゆえに，$2^{\tau(k)+2}|k(p-1)(q-1)$，$2^{\tau(k_p)+1}|k_p(p-1)$，$2^{\tau(k_q)+1}|k_q(q-1)$ となる．式 (2.2)，(2.3)，(2.4) は下記の式に置き換えることができる．

$$ed \equiv 1 \mod 2^{\tau(k)+2} \tag{3.8}$$
$$ed_p \equiv 1 \mod 2^{\tau(k_p)+1} \tag{3.9}$$
$$ed_q \equiv 1 \mod 2^{\tau(k_q)+1} \tag{3.10}$$

ここでは，$d = d[n-1]2^{n-1} \cdots d[\tau(k)+2]2^{\tau(k)+2} + d[\tau(k)+1]2^{\tau(k)+1} \cdots d[0]2^0$ なので，式 (3.8) により，$d[i](0 \leq i \leq \tau(k)+1)$ が求められる．同じく，式 (3.9)，(3.10) より，$d_p[i](0 \leq i \leq \tau(k_p))$，$d_q[i](0 \leq i \leq \tau(k_q))$ も求められる．

以上より，ある既知な (k, k_p, k_q) において，Slice[0] は既知となる．

3.2.2　枝生成と枝刈り

枝生成の段階では，既知なビット情報から新たな未知のビットを探索する．つまり Slice[$i-1$] から Slice[i] を求める．Slice[$i-1$] の一つの枝について，その枝により生成する Slice[i] は $2^5 = 32$ 通りの候補になるわけではない．CRT-RSA 暗号の制約式においてその Slice[i] は 2 通りになる．枝刈りは Slice[i] を生成するたび，既知の観測データと矛盾する枝をカットし，鍵の候補を減らす．ここの観測データとは，ビット復元ルールによって部分なビットが復元できた d_p, d_q のことを指す．

これから，Slice[$i-1$] から Slice[i] を 2 通り求めることを説明する．Slice[$i-1$] が求められた時，p, q は i 番目以下のビット，d は $\tau(k)+i$ 番目以下のビット，d_p は $\tau(k_p)+i$ 番目以下のビット，d_q は $\tau(k_q)+i$ 番目以下のビットを全て既知情報として使える．まず，p', q', d', d'_p, d'_q を下記のように定義する．

$$p' = \sum_{j=0}^{i-1} p[j]2^j, q' = \sum_{j=0}^{i-1} q[j]2^j, d' = \sum_{j=0}^{i+\tau(k)-1} d[j]2^j, d'_p = \sum_{j=0}^{i+\tau(k_p)-1} d_p[j]2^j, d'_q = \sum_{j=0}^{i+\tau(k_q)-1} d_q[j]2^j$$

これらの情報と Hensel の補題 [12] を用いて，下記の式が得られる．

$$p[i] + q[i] \equiv (N - p'q')[i] \mod 2 \tag{3.11}$$
$$d[i+\tau(k)] + p[i] + q[i] \equiv (k(N+1) + 1 - k(p'+q') - ed')[i+\tau(k)] \mod 2 \tag{3.12}$$
$$d_p[i+\tau(k_p)] + p[i] \equiv (k_p(p'-1) + 1 - ed'_p)[i+\tau(k_p)] \mod 2 \tag{3.13}$$
$$d_q[i+\tau(k_q)] + q[i] \equiv (k_q(q'-1) + 1 - ed'_q)[i+\tau(k_q)] \mod 2 \tag{3.14}$$

式 (3.12)，(3.13)，(3.14)，(3.15) の右部分は既知情報より計算できる．故に，下記の式に書き換えることができる．

$$p[i] + q[i] \equiv c_1 \mod 2 \tag{3.15}$$
$$d[i+\tau(k)] + p[i] + q[i] \equiv c_2 \mod 2 \tag{3.16}$$
$$d_p[i+\tau(k_p)] + p[i] \equiv c_3 \mod 2 \tag{3.17}$$
$$d_q[i+\tau(k_q)] + q[i] \equiv c_4 \mod 2 \tag{3.18}$$

ここで，c_1, c_2, c_3, c_4 は　mod 2 のもとで，0 か 1 となる既知な値である．Slice$[i]$ の五つ未知数が上記の四つ制約式に制限され，2 通りの解を持つ．

Slice$[i-1]$ のノードの候補が a である時，Slice$[i+1]$ の枝の数は $2a$ となる．この $2a$ の候補から観測情報と矛盾する枝を枝刈りする．公開鍵 N が n ビットの時，p, q, d_p, d_q は $n/2$ ビット以下なので，最大 $\lceil n/2-1 \rceil$ ビット目まで枝生成と枝刈りを繰り返して探索すれば良い．

3.2.3 アルゴリズム

探索完了後，残りの候補の中で，必ず真の鍵が入っている．探索から得られた p, q の積は N になるかどうかを確かめ，真の鍵を探し出す．Heninger-Shacham アルゴリズムは Algorithm 3 でまとめた．まず，記号を定義する．ビット復元ルールにより部分的に復元できた d_p, d_q を $[d_p], [d_q]$ とする．

Algorithm 3 Heninger-Shacham アルゴリズム

Input: 公開鍵 (N, e)，$[d_p], [d_q]$
for $k = 1$ to $k = e - 1$
　Slice[0] を計算する．
　for $i = 1$ to $i = \lceil n/2 - 1 \rceil$
　　Slice$[i-1]$ から二つ Slice$[i]$ を葉として出す．
　　各 Slice$[i]$ と $[d_p], [d_q]$ を比較し，異なる葉を削る．
最後に残っている鍵候補が $pq = N$ かどうかを検証する．正しい鍵を出力する．
Output: 秘密鍵 (p, q, d, d_p, d_q)

第 4 章

新たなビット復元ルール

Bernstein らの手法では，Rule 0 から Rule 3 を一回しか適用していなかった．実際に，Rule 2 を繰り返すと，もっと多くのビットが復元できると Bernstein らは主張している．しかし，その挙動は明確にされていなかった．そこで，秘密鍵のビットを復元する新たなルールを提案する．このルールによって，一回のみの適用で最大限ビットの復元に実現する．本章では，新たに提案するビット復元ルールを説明した後，それによって行った実験の結果を考察する．

4.1 提案ルールの説明

Bernstein らのルールを改良し，多くのビットが復元できるルールを提案する．ルールは前処理と Rule 0〜Rule 3 より構成される．前処理と Rule 0 は Bernstein らと同じである．Rule 1〜Rule 3 は Algorithm 4〜Algorithm 6 に対応する．

まず，記法を定義する．x は 0 か 1 が確定できないビットを表す．$\underline{*}$は掛け算を行った位置を表す．x^i は連続する x の個数が i 個であることを表す．ビット列の長さを L とした時に，上位ビットから $1, 2, \ldots, L$ 番目とインデックスを付け，$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L$ と表す．さらに，系列を上位ビット側から見ていって，1 番目の $\underline{\mathbf{1}}$ のインデックスを m_1，2 番目の $\underline{\mathbf{1}}$ のインデックスを m_2，... と定義する．さらに，$\underline{\mathbf{1}}$ の数を M とする．

- **前処理： SM→$\underline{\mathbf{x}}$，S→ x**
  SM は掛け算ビットを意味し，S は 2 乗算ビットを意味する．
- **Rule 0 :** $\underline{\mathbf{x}}\rightarrow \underline{1}$
  掛け算ビットを$\underline{1}$とする．
- **Rule 1** : Algorithm 4 を参照.
- **Rule 2** : Algorithm 5 を参照.
- **Rule 3** : Algorithm 6 を参照.

ただし，Rule 2 は下位から上位までの順で適用する．それ以外は上位から下位まで適用する．

Algorithm 4 Rule 1

Input: ビット列 (Rule 0 完了)

Output: ビット列 (Rule 1 完了)

m_1 に着目

if 最上位ビットからビットを見た際に, ある j_2 に対して, $\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{x}^{w-1-j_2}$ $(1 \leq j_2 \leq w-1)$ が存在する

then $\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{x}^{w-1-j_2} \rightarrow \mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{0}^{w-1-j_2}$

for $i = 1$ **to** $M - 1$

  m_i と m_{i+1} に着目

  if ある j_2 に対して, $\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{x}^{w-1-j_2}$ $(1 \leq j_2 \leq w-1)$ が存在する

  then $\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{x}^{w-1-j_2}$

    $\rightarrow \underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{0}^{w-1-j_2}$

end for

Algorithm 5 Rule 2

Input: ビット列 (Rule 1 完了)

Output: ビット列 (Rule 2 完了)

for $i = L$ **down to** $i = 1$

  if $\mathbf{x}_i = \mathbf{1}$

    if $\mathbf{x}_i$ の **1** と, その上位 $w-1$ ビットが

      $\mathbf{x}^{w-1-j_2}\underline{\mathbf{1}}\mathbf{0}^{j_2}\mathbf{1}$ $(1 \leq j_2 \leq w-1)$ である場合

    then $\mathbf{x}^{w-1-j_2}\underline{\mathbf{1}}\mathbf{0}^{j_2}\mathbf{1}$

      $\rightarrow \mathbf{1}\mathbf{x}^{w-j_2-1}\underline{\mathbf{1}}\mathbf{0}^{j_2}\mathbf{1}$

      $i = i - w$

  elif $\mathbf{x}_i = \underline{\mathbf{1}}$

    if $\mathbf{x}_i$ の $\underline{\mathbf{1}}$ と, その上位 $w-1$ ビットが

      $\mathbf{x}^{w-1-j_2}\underline{\mathbf{1}}\mathbf{0}^{j_2}\underline{\mathbf{1}}$ $(1 \leq j_2 \leq w-1)$ である場合

    then $\mathbf{x}^{j_2}\underline{\mathbf{1}}\mathbf{0}^{w-1-j_2}\underline{\mathbf{1}}$

      $\rightarrow \mathbf{1}\mathbf{x}^{j_2-1}\underline{\mathbf{1}}\mathbf{0}^{w-1-j_2}\underline{\mathbf{1}}$

      $i = i - w$

end for

Algorithm 6 Rule 3

Input: ビット列 (Rule 2 完了)
Output: ビット列 (Rule 3 完了)
1 もしくは $\underline{\mathbf{1}}$ が登場するまで，上位ビットを読み取る．
if ある j_2 に対して，$\mathbf{x}^{j_2}\mathbf{1}$ となる部分が存在する
then $\mathbf{x}^{j_2}\mathbf{1} \rightarrow \mathbf{0}^{j_2}\mathbf{1}$
elif $\mathbf{x}^{j_2+w-1}\underline{\mathbf{1}}$
then $\mathbf{x}^{j_2+w-1}\underline{\mathbf{1}} \rightarrow \mathbf{0}^{j_2}\mathbf{x}^{w-1}\underline{\mathbf{1}}$
for $i = 1$ **to** $M - 1$
  if m_i に着目し，ある j_2 に対して，$\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2}\mathbf{1}$ が存在する
  then $\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2}\mathbf{1} \rightarrow \underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{0}^{j_2}\mathbf{1}$
  elif m_i, m_{i+1} に着目し，$\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2+w-1}\underline{\mathbf{1}}$
  then $\underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{x}^{j_2+w-1}\underline{\mathbf{1}} \rightarrow \underline{\mathbf{1}}\mathbf{0}^{j_1}\mathbf{0}^{j_2}\mathbf{x}^{w-1}\underline{\mathbf{1}}$
end for

$w = 4$ の例を挙げる．SMSSSSSSMSSSMSSSSSSMSMSSSSSSSMSSSSSSSM の場合は，

前処理：xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Rule 0：1xxxxx1xx1xxxx11xxxxx1xxxxx1

Rule 1：1xxxxx1xx10xxx11000xx10xxxx1

Rule 2：1xx1xx11x101xx11000xx10xxxx1

Rule 3：1001xx11x101xx11000xx100xxx1

となる．Bernstein らのルールより 2 ビット多く復元できている．

これから，最適性について述べる．ルールの最適性とは，復元可能な上限までビットを復元できることを意味する．SM 時系列と矛盾しないビット系列の全候補について，ビットが全て同じ値になる時，ビットの値を確定する．ルールを適用することにより，復元可能な上限までビットの復元が行われている．提案した新たなビット復元ルールを適用すると，以下の定理 1 が成立する．

定理 1　与えられた SM 時系列に，前処理を行った後，Rule 0-3 を Rule 0，1，2，3 の順に一度ずつ適用すると，復元可能な上限までビットを復元した状態となる．

4.2　ビット復元率の評価

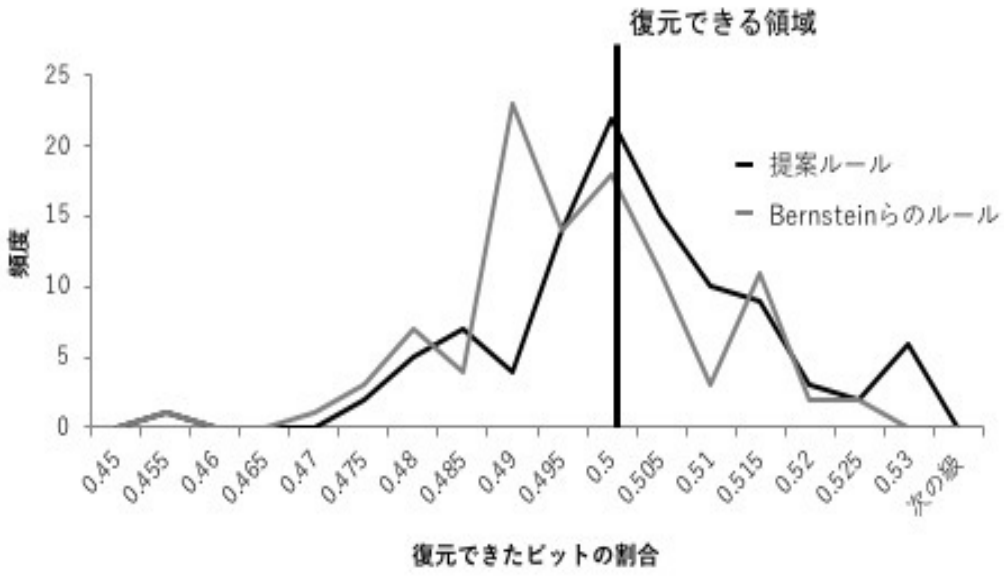

図 4.1. 1024-bit RSA，$w = 4$，復元できたビット割合の分布

表 4.1. 復元できたビットの割合平均値の比較

w	Berstein らのルール	提案したルール	差
3	60.00%	60.80%	0.80%
4	49.44%	49.96%	0.51%
5	41.60%	41.84%	0.24%
6	36.07%	36.19%	0.12%
7	31.70%	31.76%	0.06%

Bernstein らのルールと新たなビット復元ルールを実装した上，w が 3 から 7 までの実験を行った．各 w に対して，2048-bit RSA における 100 回の実験を行った．Bernstein らのルールによるビット復元割合の平均値と新しい提案したルールによるビット復元の平均値は表 4.1 でまとめた．

表 4.1 により，確かに新たなビット復元ルールにより，多くのビットが復元できていることがわかる．提案したルールは最適性を持つので，必ず彼らより多くのビットを復元する．Bernstein らのルールと比較して，新たに復元されるビットは主に Rule 2 によるものである．w の増加により，Bernstein らのルールとの差が小さくなる．これは提案した Rule 2 になる

状況が少ないからである．$w = 4$ の時に，提案したルールで秘密鍵のビットを 49.96% まで復元できた．図 4.1 により，復元できる領域は縦線の右側になる．

さらに，d_p, d_q の SM 時系列が知られた時，提案したルールを用い，1024-bit CRT-RSA における鍵全体復元の実験を 100 回実行した．Bernstein らと同じ基準で，秘密鍵候補が 1,000,000 個以上になる場合，アルゴリズムは停止し，秘密鍵を復元できなかったとみなす．実験結果は，d_p, d_q を復元できたビットの割合の平均値が 50% 以上になる確率は 49% であり，鍵全体が復元できた確率は 29% であった．同じ基準で比較する為に，Bernstein らのルールも同じ設定で 100 回の実験を実行した．実験結果は，d_p, d_q を復元できたビットの割合の平均値が 50% 以上になる確率は 25% であり，鍵全体が復元できた確率は 25% であった．したがって，提案したルールによって，鍵全体の復元率が 25% から 29% まで向上した．これは，提案したルールによる復元割合の平均値は閾値 50% に非常に近くなったことによる．

第 5 章

新たな鍵復元アルゴリズム

5.1 Random Sampling アルゴリズム

5.1.1 モチベーション

Bernstein らは，ビット復元ルールを適用した後，Heninger-Shacham アルゴリズムを用いて鍵復元を行う方法を提案している．Heninger-Shacham アルゴリズムは，既知ビットが多くなればなるほど，高い確率で攻撃に成功する．4 章では，確率 1 では復元できるビットを全て復元したビット復元ルールを提案した．残りのビットは，確率 1 で復元できないビットである．まず，ビット復元ルールを適用後に，残りの未知ビットの状況を調べる．

残りの未知ビットが 1 か 0 となる確率の評価について，理論的な評価は非常に困難であるので，実験で求めることにした．そのため，SM 時系列に矛盾しない全てのビット列を生成する Random Sampling アルゴリズムを提案する．

残りの未知ビットは全て 1 と 0 ともなるわけではないため，一様ランダムにサンプリングすることは簡単にできない．例えば，$w = 2$，ssmssm 列が与えられた時に，ビット復元ルールを適用した後の列は x$\underline{1}$x$\underline{1}$になる．しかし，SM 時系列と矛盾しないビット列は，1101，1111，0101，合計 3 通りしかない．ここで，一つ一つ w ビットであるブロックを単位として注目したアルゴリズムを提案する．

5.1.2 アルゴリズム

本節では，SM 時系列と矛盾しない全てのビット列を一様ランダムにサンプルするアルゴリズムを説明する．以下，$w = 4$ の場合を具体例として説明する．また，ある特定な場合では，ビット列が取りうるパータンを場合の数と呼ぶ．まず，SM 時系列に対して，前処理と Rule 0 を行う．ここで，$\underline{1}$の数を M とする．

はじめに，最下位のブロックについて，ブロックの位置ごとで場合の数を保存する．ブロックの位置は w 通りであるため，保存される場合の数も w 通りになる．また，ブロック内の未知なビットは全て 2 通りを持つ．

また，選んだブロックについて，

- 最上位ビットを 1 とする．
- $\underline{1}$以下のビットを全て 0 とする．
- 上記以外のビットをランダムに決定する．

さらに，ブロックとブロックの間のビットを全て 0 とする．

具体的に，$w = 4$ の場合，最下位の$\underline{1}$について，

$$
\begin{aligned}
1\mathrm{xx}\underline{1} &\quad \rightarrow 1 \times 2^2 = 4 \text{ 通り} \\
1\mathrm{x}\underline{1}0 &\quad \rightarrow 1 \times 2 = 2 \text{ 通り} \\
1\underline{1}00 &\quad \rightarrow 1 \text{ 通り} \\
\underline{1}000 &\quad \rightarrow 1 \text{ 通り}
\end{aligned}
$$

を保存する．

次に，下位ビット側から i 番目の$\underline{1}$について，$\underline{1}$の位置によっての場合の数が全てカウントしたことを前提とする．下位ビット側から $i+1$ 番目の$\underline{1}$を含むブロックについて考察する．このとき，ブロックの位置ごとに，

- 重なり合わない直下位のブロックの位置を全て求め，場合の数の総和を求める．
- 現在のブロックの場合の数を求める．
- 以上の二数の積を保存する．

具体的に，$w = 4$ の場合，下位ビット側から 1,2 番目の$\underline{1}$が，$\underline{1}\mathrm{xx}\underline{1}\mathrm{x}$ のように与えられていたとする．このとき，最下位の$\underline{1}$について，

$$
\begin{aligned}
1\mathrm{x}\underline{1}0 &\rightarrow 2 \text{ 通り} \\
1\underline{1}0 &\rightarrow 1 \text{ 通り} \\
\underline{1}0 &\rightarrow 1 \text{ 通り}
\end{aligned}
$$

の場合の数が保存されている．このとき，

- 下位ビット側から 2 番目のブロックを $1\mathrm{xx}\underline{1}$とした場合，つまり，$(1\mathrm{xx}\underline{1})\mathrm{xx}\underline{1}\mathrm{x}$ について，直下位のブロックの場合の数の総和は，
  - $(1\mathrm{xx}\underline{1})1\mathrm{x}\underline{1}0 \rightarrow 2$ 通り
  - $(1\mathrm{xx}\underline{1})01\underline{1}0 \rightarrow 1$ 通り
  - $(1\mathrm{xx}\underline{1})00\underline{1}0 \rightarrow 1$ 通り

  の合計 4 通りである．$(1\mathrm{xx}\underline{1}) \rightarrow 4$ 通りなので，16 通りの 16 を保存する．
- 下位ビット側から 2 番目のブロックを $1\mathrm{x}\underline{1}0$ とした場合，$(1\mathrm{x}\underline{1}0)\mathrm{x}\underline{1}\mathrm{x}$ について，直下位のブロックの場合の数の総和は，
  - $(1\mathrm{x}\underline{1}0)1\underline{1}0 \rightarrow 1$ 通り

- (1x$\underline{1}$0)0$\underline{1}$0→1 通り

の合計 2 通りである．(1x$\underline{1}$0)→2 通りなので，4 通りの 4 を保存する．

- 下位ビット側から 2 番目のブロックを 1$\underline{1}$00 とした場合，(1$\underline{1}$00)$\underline{1}$x について，直下位のブロックの場合の数の総和は，
  - (1$\underline{1}$00)$\underline{1}$0→1 通り

  の合計 1 通りである．(1$\underline{1}$00)→2 通りなので，1 通りの 1 を保存する．
- 下位ビット側から 2 番目のブロックを$\underline{1}$000 とした場合，直下位のブロックは全て重なり合い，条件を満たすものは存在しないので，0 通りの 0 を保存する．

以上を最上位の$\underline{1}$まで繰り返す．

次に，最上位の$\underline{1}$を含むブロックについて，値が 0 でないブロックが k 個 $(k \leq w)$ あり，それぞれのブロックに，$X_1, X_2, \ldots, X_k$ が保存されていたとする．このとき，ブロックを $X_i / \left(\sum_{j=1}^{k} X_j\right)$ の確率で選ぶ．

ここで，上位側から i 番目の$\underline{1}$を含むブロックが選ばれているとする．このとき，上位側から i 番目の$\underline{1}$を含むブロックと重なりを持たず，値が 0 でないブロックが，k 個 $(k \leq w)$ あり，それぞれのブロックに，$X_1, X_2, \ldots, X_k$ が保存されていたとする．このとき，ブロックを $X_i / \left(\sum_{j=1}^{k} X_j\right)$ の確率で選ぶ．

以上を，最下位の$\underline{1}$まで繰り返す．

以上が Random Sampling アルゴリズムである．上記のアルゴリズムをまとめると，以下の通りである．

Algorithm 7 Random Sampling

Input：w，SM 時系列

Output：SM 時系列と矛盾しないビット列

Step 1：場合の数を求める

from 最下位ブロック **to** 最上位ブロック

重なり合わない直下位のブロックの位置を全て求め，場合の数の総和を求める．

現在のブロックの場合の数を求める．

以上の二数の積を保存する．

Step 2：ブロックの位置の確定

from 最上位ブロック **to** 最下位ブロック

直上位ブロックと重なりを持たず，値が 0 でないブロックの値を呼び出し，$X_1, X_2, \ldots, X_k$ とする．

ブロックを $X_i / \left(\sum_{j=1}^{k} X_j \right)$ の確率で選ぶ．

選んだブロックについて，

最上位ビットを 1 とする．

1以下のビットを全て 0 とする．

上記以外のビットをランダムに決定する．

ブロックの最上位ビットと直上位ブロックの最下位ビットの間のビットを全て 0 とする．

5.1.3　各ビットが 0 となる確率の評価

各ビットが 0 となる確率を評価するため，2048-bit RSA の鍵に対して，$w = 4$ の時，同じ SM 列で 10,000 回ランダムサンプリングを行い，0 となる回数を調べた．分布は図 5.1，数値は表 5.1 の通りである．実験結果から，全ての未知ビットは 1/2 の確率で 1 か 0 になるとは限らない．特に，0%～10% の確率で 0 になるビット数は 20 個であり，約ビット列の 2% である．それに対して，70%～100% の確率で 0 になるビットの数は非常に少ない．

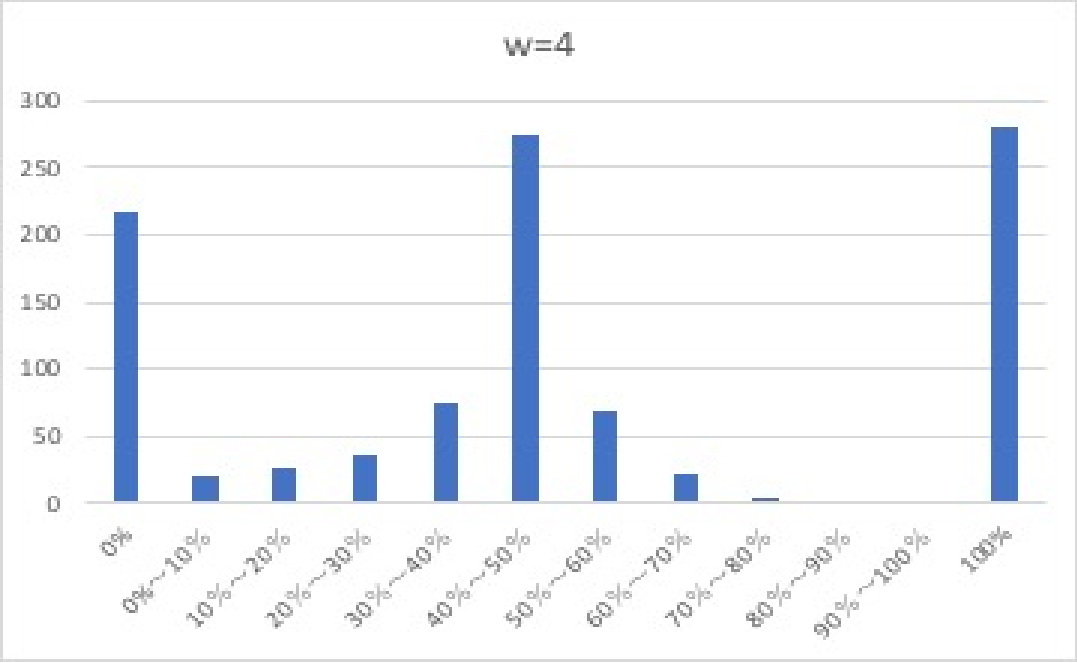

図 5.1. 0 になる確率に応じたビット数の分布（2048-bit RSA，$w = 4$）

未確定ビットの個数は，525 個であるため，未確定ビットが完全にランダムであるならば，残存しているエントロピーの値は，525 となる．しかし，分布に偏りがあり，これより小さくなることが期待される．実際に，実験結果に基づき，未確定ビットに残存しているエントロピーの値を評価した．その結果は 485 であった．これより，実質的に 52.5% 分の情報を入手できたとみなすことができる．

5.2　新たな鍵復元アルゴリズム

前節の結果により，ビット復元ルールを適用した後，1 と 0 となる確率に偏りがあることが判明した．すでに値が確定したビットだけでなく，低い確率で 0 になるビットを 1 に設定する方法で，既知ビットを増やすことより，鍵復元率を上げるアルゴリズムを考える．しかし，新たに 1 と設定したビットが，全て本当に 1 である確率は極めて低く，鍵全体の復元成功率も低くなる．しかし，1 ビットは 0 に設定することにより，1 ビットの反転が許容され，反転を許

表 5.1. 0 になる確率に応じたビット数の分布（2048-bit RSA，$w = 4$）

0 になる確率	ビットの数
0%	217
0%～10%	20
10%～20%	26
20%～30%	37
30%～40%	75
40%～50%	274
50%～60%	68
60%～70%	22
70%～80%	3
80%～90%	0
90%～100%	0
100%	279

容する確率は全て 1 である確率の倍になる．成功確率を高くできることが期待できる．

5.2.1　アルゴリズム

本節では，新たな鍵復元アルゴリズムを説明する．まず，得られた秘密鍵の SM 時系列に，4 章で提案したビット復元ルールを適用する．次に，SM 時系列を用い，Random Sampling を行う．残りの未知ビットのうち，0 になる確率があるしきい値以下のビットを 1 に設定する．ビット復元ルールによる復元したビットと，未知ビットのうち，1 に設定したビットを含め，既知ビットとして扱う．既知ビットを用い，Heninger-Shacham アルゴリズムを実行する．鍵候補がある個数を超える場合，実験は失敗する．Heninger-Shacham アルゴリズムが探った鍵は CRT-RSA の関係式に満足さない場合，1 ビット反転した物と 2 ビットまで反転する物も試す．例えば，4 ビットが特定された場合，まず，1111 を試す．成功しなかった場合，次に，1 ビットの反転を許容する 1110，1101，1011，0111 を試す．成功しなかった場合，次に，2 ビットの反転を許容する 1100，1001，1010，0011，0101，0110 を試す．

Random Sampling アルゴリズムにより，0 となる確率がしきい値以下のビットを特定した上で，以下の三つの戦略に基づく実験を行う．

戦略 1：

特定したビットを全て 1 に設定する．**Step 4** まで実行することに対応する．

戦略 2：

特定したビットを最大 1 ビットを 0 に設定，ほかは 1 に設定する．**Step 6** まで実行することに対応する．

戦略 3：

特定したビットを最大 2 ビットを 0 に設定，ほかは 1 に設定する．**Step 8** まで実行することに対応する．

Algorithm 8 提案した鍵復元アルゴリズム

Input：d_p, d_q の SM 時系列，K，L，しきい値 P

Output：真の秘密鍵 or 失敗

Step 1：4 章が提案したビット復元ルールを SM 時系列に適用

Step 2：**Step 1** で得られた列に Random Sampling を K 回行い，0 となる確率が P 以下のビットを特定

Step 3：**Step 2** で特定したビットを全て 1 に設定

Step 4：**Step 3** で得られた列に Heninger-Shacham アルゴリズムを適用

　　if 秘密鍵の候補が L 個を超える

　　失敗を出力

　　else 秘密鍵の候補 (KEY_1，KEY_2, $\cdots$，KEY_M) を全て保存

　　if KEY_i が CRT-RSA の関係式に満たす

　　KEY_i を出力

　　else 失敗を出力

Step 5：**Step 2** で特定したビットのうち，1 つを 0 に設定，他は 1 に設定.

設定した 0 の位置によって，**Step 5** と **Step 6** の繰り返しが発生する.

Step 6：**Step 5** で得られた列に Heninger-Shacham アルゴリズムを適用

　　if 秘密鍵の候補が L 個を超える

　　失敗を出力

　　else 秘密鍵の候補 (KEY_1，KEY_2, $\cdots$，KEY_M) を全て保存

　　if KEY_i が CRT-RSA の関係式に満たす

　　KEY_i を出力

　　else 失敗を出力

Step 7：**Step 2** で特定したビットのうち，2 つを 0 に設定，他は 1 に設定.

設定した 0 の位置によって，**Step 7** と **Step 8** の繰り返しが発生する.

Step 8：**Step 7** で得られた列に Heninger-Shacham アルゴリズムを適用

　　if 秘密鍵の候補が L 個を超える

　　失敗を出力

　　else 秘密鍵の候補 (KEY_1，KEY_2, $\cdots$，KEY_M) を全て保存

　　if KEY_i が CRT-RSA の関係式に満たす

　　KEY_i を出力

　　else 失敗を出力

5.2.2 鍵復元の成功確率の評価

まず，しきい値を定める．1024-bit RSA に対して，$w = 4$ の時に，ビット復元ルールを適用した後，10,000 回の Random Sampling により洗い出した残り未知ビットに対して，戦略 1，戦略 2，戦略 3 を行なった．各しきい値に対して，100 回実験を行った．鍵候補が $L = 1,000,000$ 以上の時，実験を失敗とする．

結果は表 5.2 の通りである．特に，

戦略 1 での鍵復元成功確率 =

推測が成功した時の鍵復元成功確率 × 戦略 1 での推測に成功する確率

戦略 2 での鍵復元成功率 =

推測が成功した時の鍵復元成功確率 × 戦略 2 での推測に成功する確率

戦略 3 での鍵復元成功率 =

推測が成功した時の鍵復元成功確率 × 戦略 3 での推測に成功する確率

が成り立っている．

表 5.2. 鍵復元の成功確率（1024-bit RSA，$w = 4$）

しきい値	15%	12%	10%	7%	5%	0%
推測が成功した時の鍵復元成功確率	0.98	0.89	0.84	0.67	0.45	0.27
戦略 1 での推測に成功する確率	0.15	0.27	0.35	0.63	0.83	1
戦略 1 での鍵復元成功確率	0.15	0.23	0.29	0.42	0.41	0.27
戦略 2 での推測に成功する確率	0.37	0.57	0.66	0.96	0.98	1
戦略 2 での鍵復元成功率	0.36	0.51	0.55	0.64	0.44	0.27
戦略 3 での推測に成功する確率	0.61	0.81	0.92	0.99	1	1
戦略 3 での鍵復元成功率	0.60	0.72	0.77	0.66	0.45	0.27

表 5.3. 鍵復元の成功確率の比較（1024-bit RSA，$w = 4$）

アルゴリズム	鍵全体の復元率
Bernstein らのルール [1] + Heninger-Shacham アルゴリズム	25%
提案したルール+ Heninger-Shacham アルゴリズム	29%
新たな鍵復元アルゴリズム	77%

しきい値を 0% と設定した時は，元の手法，提案ルールを適用した後，Heninger-Shacham アルゴリズムを用いて得られた鍵復元率と一致する．しきい値を小さくすると，推測が成功した時の鍵復元成功確率は下がるが，戦略での推測に成功する確率は上がる．故に，戦略での鍵復元成功確率が最適となる時のしきい値が存在する．

実験により，各戦略の最適となるしきい値は以下である．

- 戦略 1：7%（鍵復元率：0.42）
- 戦略 2（1 ビットまで反転）：7%（鍵復元率：0.64）
- 戦略 3（2 ビットまで反転）：10%（鍵復元率：0.77）

戦略 1 は提案したルール +Heninger-Shacham アルゴリズムより 13% を向上し，戦略 2 は戦略 1 より 22% を向上し，戦略 3 は戦略 2 より 13% を向上した．Bernstein らのルール＋ Heninger-Shacham アルゴリズム，提案したルール＋ Heninger-Shacham アルゴリズムとの比較は，表 5.3 でまとめた．提案方式は，既存の方式よりも高い確率で鍵全体の復元に成功している．しかし，計算時間は増加している．しきい値より特定されているビットの数を l とすると，戦略 1 では Heninger-Shacham アルゴリズムを最大 1 回適用する．1 ビットの反転を許容する時，Heninger-Shacham アルゴリズムを最大 l 回適用する．2 ビットの反転を許容する時，Heninger-Shacham アルゴリズムを最大 $l(l-1)/2$ 回適用する．上記の実験で計測した l の大きさの平均値は下記の表 5.4 でまとめた．このアイデアの拡張として，k ビットの反転を許容するアルゴリズムを考えることができる．これにより，成功確率を高くすることができる．その一方で，繰り返し回数が ${}_lC_k$ 回必要であり，計算時間は増大する．例えば，しきい値を 10% に設定する時，l は約 19 となり，$1 + 19 + 19 \times 18/2 = 191$，つまり 191 倍の計算時間が必要になる．191 倍の計算時間と引き換えに，29% から 77% の成功確率の向上を得た．

表 5.4. 特定されているビットの数

しきい値	15%	12%	10%	7%	5%	0%
l	30.92	22.78	18.78	12.34	8.53	0

第 6 章

結論

本研究は，サイドチャネル攻撃により，Sliding Window 法から漏洩した秘密鍵の SM 時系列を用い，秘密鍵復元の研究を行った．Bernstein らの提案した SM 時系列から，秘密鍵を部分的に復元するアルゴリズムを改良し，4 章では，新たなビット復元ルールを提案した．新たなルールは Bernstein らより多くのビットを復元できる．さらに，最大限まで復元可能なビットを復元できる最適性を持つ．二つのルールを比較するため，Bernstein らのルール＋ Heninger-Shacham アルゴリズムと提案したルール＋ Heninger-Shacham アルゴリズム，二つの秘密鍵全体の復元手法を実装した．$w = 4$，1024-bit RSA 暗号において，各手法は 100 回を行った．その結果，秘密鍵全体の復元率は，前者が 25% となり，後者は 29% となった．後者は優れているとも言える．

次に，ビット復元ルールを適用した後，残り未知ビットが 0 か 1 になる確率を評価した．そのため，元の SM 時系列と矛盾しないビット列をランダムサンプリングできる Random Sampling アルゴリズムを提案した．$w = 4$ の時，2048-bit RSA の秘密鍵の SM 時系列に対して，数値実験を行った結果，残り未知ビットの分布に偏りがあることを判明した．そして，この事実をうまく利用し，新たな秘密鍵復元アルゴリズムを提案した．具体的に，残り未知ビットの中，0 となる確率がしきい値以下のビットを 1 とし，既知情報として Heninger-Shacham アルゴリズムを適用する．さらに，1 ビットまでの反転と 2 ビットまでの反転を許容するアルゴリズムも提案した．その結果，ベストとなる秘密鍵の復元率は 77% になった．これは前段落で提示した二つの手法よりはるかに優れている．

謝辞

本研究は，指導教員の國廣准教授と東京大学大学院情報理工学系研究科の大西　健斗さんとの共同研究となり，お二方よりたくさんの支援とアドバイスを受けました．心より深く感謝しております．

参考文献

[1] Bernstein, D.J., Breitner, J., Genkin, D., Bruinderink, L.G., Heninger, N., Lange, T., van Vredendaal, C., Yarom, Y., "Sliding Right into Disaster: Left-to-Right Sliding Windows Leak," CHES 2017, LNCS, vol. 10529, pp. 555–576 (2017).

[2] RSA Laboratories: PKCS#1 v2.1: RSA Cryptography

[3] Gandolfi, K., Mourtel, C., Olivier, F., "Electromagnetic Attacks: Concrete Results," CHES 2001. LNCS, vol. 2162, pp. 252–261(2001).

[4] Heninger, N., Shacham, H., "Reconstructing RSA Private Keys from Random Key Bits," CRYPTO 2009, LNCS, vol. 5677, pp. 1–17 (2009).

[5] Kocher, P.C., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO 1996, LNCS, vol. 1109, pp. 104–113 (1996).

[6] Kocher, P., Jaffe, J., Jun, B., "Differential Power Analysis," CRYPTO 1999, LNCS, vol. 1666, pp. 388–397 (1999).

[7] Kunihiro, N., Shinohara, N., Izu, T., "Recovering RSA Secret Keys from Noisy Key Bits with Erasures and Errors," In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 180–197 (2013).

[8] Menezes, A.J.,van Oorschot, P.C., Vanstone, S.A., "Handbook of Applied Cryptography," CRC Press, Boca Raton, Florida (1996).

[9] Miller, V., "Use of elliptic curves in cryptography," In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–428 (1986).

[10] Paterson, K. G., Polychroniadou, A., Sibborn, D. L., "A Coding-Theoretic Approach to Recovering Noisy RSA Keys," In: Wang, X., Sako, K. (eds.) ASIACRYPT. LNCS, vol. 7658, pp. 386–403 (2012).

[11] 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).

[12] Serge Lang, "Algebraic Number Theory, Addison-Wesley Publishing Company," pp. 43 (1970).

[13] Yarom, Y., Falkner, K., "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," In: USENIX 2014, pp. 719–732 (2014).

[14] Yarom, Y., Benger, N., “Recovering OpenSSL ECDSA nonces Using the Flush+Reload Cache Side-Channel Attack,” Cryptology ePrint Archive, Report 2014/140 (2014).