平成 30 年度

修士論文

準同型署名の弱安全性から強安全性への効率的な変換

47-176093　伊藤宗一郎

指導教員　國廣昇　准教授

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

準同型署名とは，メッセージ同士の演算結果に対応する署名を，それぞれのメッセージに対応する署名同士の演算により生成することのできる署名方式である．準同型署名に対する安全性には，選択的安全性と適応的安全性，弱安全性と強安全性という二つの軸がある．一つ目の軸については，現時点で提案されている全ての方式は，選択的安全性よりも強い安全性である適応的安全性を実現している．しかし，二つ目の軸については，近年まで強安全性を満たす方式は存在しなかった．ここで，強安全性とは，攻撃者が署名の偽造に使う関数において，チャレンジャに署名クエリを行なっていない入力を含む関数も認める安全性である．Catalano ら（ACNS, 2018）は，弱適応的安全性しか満たさない従来の準同型署名を，強適応的安全性を満たす準同型署名に変換するジェネリックな方法を構成した．これに対し，本稿では，Catalano らの変換で冗長な箇所があることを指摘し，彼らの方式よりも効率のいい変換を提案する．次に，Gorbunov らによる格子に基づく弱適応的安全な準同型署名方式（STOC, 2015）に対しては，さらに効率的な強適応的安全への変換が可能であることを示す．この変換は，うまく公開鍵を使い回す工夫を導入することにより，全体の公開鍵サイズを小さくすることに成功している．

また，Gorbunov らによる準同型署名の改良も行なった．具体的には，彼らの方式では計算モデルとしてサーキットを用いていたが，我々は計算モデルを branching program に変更した．これにより，安全性をより難しい SIS 問題に帰着させることが可能となる．一方で，この方式は弱適応的安全性を満たさず，弱選択的安全性しか満たしていないという欠点がある．これにより，上で構成した安全性強化は使えないため，この方式に対する強安全性への強化として，我々は新しく，弱選択的安全性から強選択的安全性への変換を構成した．

キーワード　準同型署名，強安全性，弱安全性，格子，branching program

目次

第 1 章

導入

1.1 背景

近年，クラウド技術などの発達により，大規模なデータの収集と活用が可能になっている．それにより，データの機密性や認証性を保ったまま演算を行うことのできる，準同型性を持った暗号化方式や署名方式の研究が盛んに行われている．

準同型署名とはデータ同士の演算結果に対応する署名を，それぞれのデータに対応する署名同士の演算で生成することのできる署名方式である．これを用いると，ユーザがデータ同士の演算を外部機関に委託するときに，その計算が正しく行われたかどうかを検証することができる．

まず，あるユーザ A が，データの演算を外部機関に委託する際に，どのように準同型署名が用いられるかを示す．ユーザ A は，はじめに公開鍵，検証鍵，秘密鍵という 3 つの鍵を生成する．そして，それらの鍵を用いてそれぞれのデータに対して署名を生成し，署名とデータの組と，計算してほしい関数を外部機関に与える．これを受け取った外部機関は，データ同士の演算だけでなく，署名同士の演算も行い，新しい署名を生成する．そして，データの演算結果と，新しい署名の両方をユーザ A に返す．これに対し，ユーザ A は，公開鍵同士の演算を行い，新しい公開鍵を生成する．そして，新しい公開鍵を用いて，外部機関から受け取った値が正しく計算された値かどうかを検証する．ここで，外部機関がデータと署名に対し正しく演算を行なっていた場合には，新しい署名は演算後のデータに対応する署名になっており，ユーザ A による検証を通るという性質を持っている．逆に，外部機関が正しく演算を行なっていなかった場合には，ユーザ A による検証を通らないという安全性を持っているので，ユーザ A は外部機関が正しく演算を行ったかどうか検証することができる．

データ同士の計算を自分で行うことと比較して，計算を外部機関に委託して，正しく計算したことを準同型署名を用いて確認することのメリットは複数ある．まず，検証による計算量はデータ同士の演算よりも小さくなる．また，検証にはデータの値を用いないので，データを外部機関に送った後は保存しておく必要がない．以上より，データの量が大規模な場合には，準同型署名は非常に有用である．

1.2　準同型署名の評価基準

この準同型署名において，データ間の演算の自由度，安全性の強弱，安全性帰着をどの困難性仮定に行っているか，という 3 点が重要である．

最初の準同型署名は [JMSW02] で提案された．この方式では，データ間の線形演算のみが可能であった．その後，[BF11a] や [CFW14] は多項式演算が可能な方式を提案した．そして，Gorbunov ら [GVW15] は任意の関数の計算が可能な完全準同型署名を提案した．Gorbunov らの方式は，現時点で構成されている準同型署名方式の中で最も演算の自由度が高い．

一方で，安全性を高める研究も行われてきた．準同型署名の安全性には二つの軸がある．一つ目の軸は，選択的安全性か適応的安全性かというものであり，二つ目の軸は，弱安全性か強安全性かというものである．これにより，準同型署名の安全性は弱選択的安全性，強選択的安全性，弱適応的安全性，強適応的安全性の 4 種類がある．

選択的安全性と適応的安全性の違いは，攻撃者とチャレンジャの間で行われる安全性ゲームの違いにある．選択的安全性における安全性ゲームでは，攻撃者はチャレンジャから公開鍵や検証鍵を受け取る前にしか署名クエリを行えない．一方，適応的安全性における安全性ゲームでは，攻撃者はチャレンジャから公開鍵や検証鍵を受け取った後に署名クエリを行える．よって，適応的安全性の方が選択的安全性よりも強い安全性である．現時点で提案されている方式は全て，適応的安全性を実現しているか，適切な変換を施すことで適応的安全性を実現するようになるかのいずれかである．例えば，[GVW15] の方式は，元々は選択的安全性しか満たしていないが，彼らは同論文中で適応的安全性への変換も構成しており，それを用いることで適応的安全性を実現することが可能となる．

弱安全性と強安全性の違いは，攻撃者の出力に対し，どこまでを偽造として認めるかというところにある．準同型署名に対する安全性は，[BF11a] らが初めて定義を行なったが，この定義では，偽造とは，正しく計算を行なった値とは異なる値でチャレンジャに受理されるものである．また，この定義では，攻撃者が出力する関数に含まれる入力は全て署名クエリを行なったものでなければならないという制約がある．この制約を取り払うために，[Fre12] は，攻撃者が出力する関数に，署名クエリを行なっていない入力が含まれるものも偽造として認める定義を提案した．ここで，攻撃者が出力する関数の入力に署名クエリを行なっていない入力が含まれている場合，正しい出力をどう定義するかという問題がある．この問題に対処するために，[Fre12] は，well-defined program を定義した．これは，攻撃者が署名クエリしていないデータの値により，出力値が変化しない関数のことである．このとき，攻撃者がクエリしていない入力の値に関わらず，well-defined program の出力値は一定なので，それを正しい値と定義することができる．これにより，準同型署名の偽造として，二つの種類が生じた．一つは，攻撃者が出力した関数が well-defined program で，かつ正しい値ではない値で受理されるものである．もう一つは，攻撃者が出力した関数が well-defined program ではないときに受理されるものである．しかし，この定義にも問題が残っている．それは，ある関数が well-defined program か否かを決定するのは一般に多項式時間では行えないというものである．この欠点

を解決する新しい安全性定義を，Catalano ら [CFN18] が提案した．彼らが提案した安全性は，[BF11a] の定義した偽造に加え，攻撃者が出力した関数が well-defined program であるか否かに関わらず，署名クエリしていないデータを含む関数のときに受理されるもの全てを偽造として認めるものである．この安全性が強安全性と呼ばれ，それに対して [BF11a] の定義した安全性は弱安全性と呼ばれる．強安全性は [Fre12] が提案した安全性よりも真に強く，検証は多項式時間で可能である．また，[CFN18] は同時に，弱適応的安全性しか満たさない従来の準同型署名を，強適応的安全性を満たす準同型署名に変換するジェネリックな変換も構成している．この方式は，元の準同型署名 Π と同時に，もう一つ別の準同型署名 Π_+ を並列に実行するもので，両者は同じ大きさのサーキットについて演算を行うので，公開鍵のサイズや演算の計算量などが最大で 2 倍まで増加する．

最後に，安全性帰着をどの困難性仮定に行っているかという点も重要である．例えば，困難性仮定として強 RSA 仮定を用いている [CFW11] や多重線型 Diffie-Hellman 仮定を用いている [CFW14] などの方式は，量子計算機への耐性を持っていない．それに対し，Gorbunov らの方式 [GVW15] は，量子計算機を用いても効率的に解く方法はいまだに見つかっていない SIS 問題を困難性仮定として用いている．ただし，彼らの方式では剰余 q はセキュリティパラメータ λ に対して準指数オーダーの大きさが必要である．一般に，剰余 q の大きさが大きくなるほど SIS 問題を解くことが簡単になってしまうので，q の大きさが多項式オーダーの SIS 問題と比較すると，簡単な SIS 問題に安全性帰着を行なっているという欠点がある．

1.3 我々の貢献

我々の貢献は主に 4 点である．

1 点目は，[CFN18] の方式よりも効率的な，弱適応的安全性しか満たさない準同型署名を，強適応的安全性を満たす準同型署名に変換するジェネリックな変換の構成である．具体的には，元の準同型署名 Π と同時に実行する準同型署名 Π_+ で計算を行うサーキットが，Π で計算を行うサーキットよりも小さい方式を提案して，同じ安全性を満たすことを示した．これにより，演算の計算量や演算後の署名サイズの増加を抑えることができる．

2 点目は，弱選択的安全性しか満たさない準同型署名を，強選択的安全性を満たす準同型署名に変換するジェネリックな変換の構成である．現時点で構成されている方式は全て適応的安全性を実現しているので，この変換は今まで研究されてこなかったが，今回我々は選択的安全性しか満たさない方式[*1]を新しく構成した．よって，その方式に対する強安全性への強化として，この変換を構成した．

3 点目は，弱適応的安全性を満たす特定の方式に対する，上記のジェネリック変換よりもさらに効率的な強適応的安全への変換の構成である．ここで，特定の方式とは，現時点で唯一の完全準同型署名であり，弱適応的安全性を満たす方式 [GVW15] のことである．変換の効率化

[*1] [GVW15] が構成した変換を用いると適応的安全を満たすことが可能であるが，一方で計算可能な関数のクラスに大きな制約を設けてしまうという欠点がある．

は二つのアイデアからなる．一つ目は，Π_+ で計算するサーキットが加算ゲートのみにより構成されていることに着目することで，Π_+ で用いるパラメータを Π で用いるものよりも小さくすることができるというものである．これにより，演算後の署名サイズなどを圧縮できる．二つ目は，公開鍵の構造をうまく使うことにより，Π_+ で用いる公開鍵を，Π で用いる公開鍵から計算することができるというものである．これにより，変換を通して増加していた公開鍵のサイズを一定に保つことが可能になる．

4 点目は，Gorbunov らの方式の改良である．具体的には，剰余 q がセキュリティパラメータ λ に対して準指数オーダーの大きさが必要だった Gorbunov らの方式を改良し，q が λ に対して多項式オーダーの大きさとなる方式を構成した．これは，Gorbunov らの方式が計算モデルとして用いていたサーキットではなく，branching program を計算モデルとして用いることで実現できる．ただし，このアイデアは，同じように q が λ に対して多項式オーダーの大きさである準同型暗号を構成した先行研究 [GV15] ですでに言及されており，我々の構成はそれを定式化したものである．また，2 点目の貢献の箇所で述べた通り，この方式は弱選択的安全性しか満たしておらず，かつ [GVW15] が構成した選択的安全性から適応的安全性への変換を適用すると計算を行うことできる関数のクラスに強い制約を設けるので，変換を行うと方式の実用性が著しく落ちてしまうという欠点がある．ただし，適応的安全への変換を行わない場合にも，2 点目の貢献である，弱選択的安全から強選択的安全へのジェネリックな変換を用いることで，強安全性を実現することができる．

1.4　本稿の構成

2 章では記法や基本的な定義の紹介を行い，3 章では準同型署名とその安全性の定義を行う．4 章では，貢献の 1 点目である，弱適応的安全性から強適応的安全性へのジェネリックな安全性強化の構成と，2 点目の弱選択的安全性から強選択的安全性へのジェネリックな安全性強化の構成を行う．5 章では貢献の 3 点目である，[GVW15] の方式に対する安全性強化のより効率的な手法を紹介する．最後に，6 章では，貢献の 4 点目である，[GVW15] の方式の改良を行う．

第 2 章

準備

記号の準備 $\lambda \in \mathbb{N}$ はセキュリティパラメータを表す． $\mathbf{SD}(X, Y)$ は，二つの確率変数 X と Y の統計距離を表す．特に断りのない場合．log の底は 2 とする．$n \times m$ 行列 $\mathbf{A}$ に対し，ノルムを $\|\mathbf{A}\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{m} |a_{ij}|$ と定義する．ある関数 $\epsilon : \mathbb{N} \to \mathbb{R}$ が negligible であるとは，任意の定数 $c \ge 0$ に対してある $n_0 \in \mathbb{N}$ が存在し，$n \ge n_0$ に対して $\epsilon(n) = O(n^{-c})$ が成り立つことを意味し negligible な関数の集合を negl とする．一方，ある関数が多項式で表現できるとき， $\mathsf{poly}(\cdot)$ と表す. また，関数 f を $f : \{0,1\}^{\le N} \to \{0,1\}$ であると書くとき，f は N ビット以下の入力を受け取り 1 ビットの出力を返す関数を意味する．最後に，集合 S に対して，$s \leftarrow S$ は S から一様ランダムに s を選ぶことを表す．

2.1 基本的な定義

まず，最小エントロピーと Small Integer Solution（SIS）困難性仮定を定義する．SIS 困難性仮定は [GVW15] の準同型署名方式や，6 章で構成する方式が困難性仮定として用いている．

定義 1 (最小エントロピー [DRS04])**.** 確率変数 X の最小エントロピーは以下のように定義される．

$$\mathbf{H}_\infty(X) := -\log(\max_x \Pr[X = x]).$$

相関関係のある確率変数 Y の値がわかっているときの確率変数 X の条件付き最小エントロピーは以下のように定義される．

$$\mathbf{H}_\infty(X|Y) := -\log(\mathop{\mathbf{E}}_{y \leftarrow Y}[\max_x \Pr[X = x|Y = y]]).$$

Y の値が与えられたときに X の値を当てられる確率は最大で $2^{-\mathbf{H}_\infty(X|Y)}$ となる．

補題 1 ([DRS04])**.** X, Y を任意の確率変数とし， Y の確率空間を $\mathcal{Y}$ とすると，$\mathbf{H}_\infty(X|Y) \ge \mathbf{H}_\infty(X) - \log(|\mathcal{Y}|)$ となる．

定義 2 (SIS 困難性仮定 [Ajt96])**.** 各パラメータ $n = n(\lambda), m = m(\lambda), q = q(\lambda), \beta = \beta(\lambda)$ が λ によって決まるとき, $\mathsf{SIS}(n, m, q, \beta)$ 困難性仮定とは， 任意の多項式時間アルゴリズム $\mathcal{A}$ に

対して，以下の確率が $\mathsf{negl}(\lambda)$ になるという仮定である．

$$\Pr[\mathbf{A}\cdot\mathbf{u}=\mathbf{0}\wedge\mathbf{u}\neq\mathbf{0}\wedge\|\mathbf{u}\|_\infty\leq\beta:\mathbf{A}\leftarrow\mathbb{Z}_q^{n\times m},\mathbf{u}\leftarrow\mathcal{A}(1^\lambda,\mathbf{A})]$$

2.2　計算モデル

次に，準同型署名において計算する関数を表す際に用いる labeled program と，6 章で構成する方式が計算モデルとして用いる branching program を定義する．

Labeled program とは，入力にラベルを付与することで，関数そのものの情報だけでなく，どのデータをどの入力として扱うかを定めながら計算を行うかという情報も与えたものである．

定義 3 (Labeled Program[GW13])**.** Labeled program $\mathcal{P}=(f,(\tau_1,\ldots,\tau_k))$ は，関数 $f:\mathcal{X}^k\to\mathcal{X}$ と，それぞれの入力に対応するラベル $\tau_i\in\mathbb{N}$ からなる．複数の labeled program $\{\mathcal{P}_i=(f_i,T_i)\}_{i\in[k]}$（ただし $f_i:\mathcal{X}^{k_i}\to\mathcal{X},k_i\leq k,T_i\subset\mathbb{N}^{k_i}$）と関数 $g:\mathcal{X}^k\to\mathcal{X}$ に対して，$\mathcal{P}^*=g(\mathcal{P}_1,\ldots,\mathcal{P}_k)$ は，記号の濫用ではあるが，$\mathcal{P}^*=(g(f_1,\ldots,f_k),\bigcup_{i\in[k]}T_i)$ を意味する．また，特に g_{id} を恒等写像として $\mathcal{I}_\tau:=(g_{\mathrm{id}},\tau)$ となる labeled program を，$\tau\in\mathbb{N}$ の identity program と言う．全ての labeled program $\mathcal{P}=(f,(\tau_1,\ldots,\tau_k))$ は，identity program を用いて，次のように書き表わせる $\mathcal{P}=f(\mathcal{I}_{\tau_1},\ldots,\mathcal{I}_{\tau_k})$．

次に，branching program の定義であるが，ここでは特に，permutation branching program と呼ばれるものの定義を行う．

定義 4 (Permutatoin Branching Programs[BV14])**.** 幅 w，長さ L，入力空間 $\{0,1\}^\ell$ の permutation branching program BP は L 組の $\{(\mathsf{var}(t),\sigma_{t,0},\sigma_{t,1})\}_{t\in[L]}$ で表される．ここで，

- $\sigma_{t,0}$，$\sigma_{t,1}$ は $[w]$ から $[w]$ への置換.
- $\mathsf{var}:[L]\to[\ell]$.

入力が $x=(x_1,\ldots,x_\ell)\in\{0,1\}^\ell$ のとき，branching program BP は以下のように出力を計算する．初期状態を $\eta_0=1$ として，t ステップ目の状態を $\eta_t\in[w]$ で表す．このとき，η_t は以下のように再帰的に計算できる．

$$\eta_t=\sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1})$$

そして，$\eta_L=1$ のとき出力は $\mathsf{BP}(x)=1$ となり，それ以外は 0 となる．

[BV14] のように， 状態をベクトル $\mathbf{v}_t\in\{0,1\}^w$ で表すこともできる．t ステップ目での状態が $\eta_t=i$ のとき，$\mathbf{v}_t[i]=1$ かつ $\forall j\in[w]\wedge j\neq i,\mathbf{v}_t[j]=0$ とする．初期状態は $\eta_0=1$ なので，$\mathbf{v}_0=[1,0,\ldots,0]$ となる．この状態ベクトルに関して，$\mathbf{v}_t[i]=1$ となるのは以下の 2 つの場合のどちらか一方が成立するときである．

- $\mathbf{v}_{t-1}[\sigma_{t,0}^{-1}(i)]=1$ and $x_{\mathsf{var}}(t)=0$

- $\mathbf{v}_{t-1}[\sigma_{t,1}^{-1}(i)] = 1$ and $x_{\mathsf{var}}(t) = 1$

この関係を使って，状態ベクトルの更新式は以下のように書き表わせる．

$$\mathbf{v}_t[i] = \mathbf{v}_{t-1}[\sigma_{t,0}^{-1}(i)] \cdot (1 - x_{\mathsf{var}(t)}) + \mathbf{v}_{t-1}[\sigma_{t,1}^{-1}(i)] \cdot x_{\mathsf{var}(t)}$$

Branching program について，以下の定理が成り立つ．

定理 1 (バリントンの定理 [Bar89])**.** 幅が *5* の *permutation branching program* が計算可能な関数のクラスは，クラス $\mathcal{NC}^1$ と等価である．

上の定理は，幅が 5 の permutation branching program がクラス $\mathcal{NC}^1$ の関数を多項式時間で計算可能であるということと同時に，クラス $\mathcal{NC}^2$ 以上の関数は多項式時間で計算できないということを示している．また，[Bar89] は，両者の等価性を示す際に，実際にサーキットを幅が 5 の permutation branching program に変換する方法も示している．

2.3 格子アルゴリズム

最後に，準同型署名を構成するときに用いるアルゴリズムの紹介をする．
以下のアルゴリズムは，署名を生成する際に用いられる．

定理 2 (サンプリング・アルゴリズム [GPV08, AP09, MP12])**.** 格子パラメータ $n, q, m,$ ノルムの上限値 β が，$m = O(n \log q)$ かつ $\beta = O(n\sqrt{\log q})$ を満たしているとする．このとき，以下の性質を持つ多項式時間アルゴリズムの組 $(\mathsf{TrapGen}, \mathsf{Sample}, \mathsf{SamplePre})$ が存在する．

- $\mathsf{TrapGen}(1^\lambda) \to (\mathbf{A}, \mathsf{td})$: セキュリティパラメータ λ を入力にとり，トラップドア生成アルゴリズムはランク n の行列 $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ とトラップドア td を出力する．
- $\mathsf{Sample}(\mathbf{A}) \to \mathbf{U}$: 行列 $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ を入力にとり, サンプリングアルゴリズムは行列 $\mathbf{U} \in \mathbb{Z}_q^{m \times k}$ を出力する．
- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td})$: 行列 $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$，ターゲット行列 $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$，トラップドア td を入力にとり，原像サンプリングアルゴリズムは行列 $\mathbf{U} \in \mathbb{Z}_q^{m \times k}$ を出力する．
- 上記のアルゴリズムは以下の性質を持つ．$(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ としたとき，
  1. $\mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A})$．このとき，$\|\mathbf{U}\|_\infty \leq \beta$
  2. $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$，$\mathbf{U} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td})$ としたとき， $\mathbf{AU} = \mathbf{V}$ かつ $\|\mathbf{U}\|_\infty \leq \beta$ となる．
  3. $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda), \mathbf{A}' \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A}), \mathbf{V} = \mathbf{AU}, \mathbf{V}' \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{U}' \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}', \mathsf{td})$ に対して以下が成立する．

$$\mathbf{A} \approx \mathbf{A}' \ \textit{and}\ (\mathbf{A}, \mathsf{td}, \mathbf{U}, \mathbf{V}) \approx (\mathbf{A}, \mathsf{td}, \mathbf{U}', \mathbf{V}').$$

- 条件を満たす任意の n, m, q に対して，ある行列 $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ と，$\mathbf{V} \in \mathbb{Z}_q^{n \times k}$（$k$ は任意の自然数）を入力にとる決定的多項式時間アルゴリズム $\mathbf{G}^{-1}$ が存在し，出力 $\mathbf{R} = \mathbf{G}^{-1}(\mathbf{V})$ は $\mathbf{GR} = \mathbf{V}$ かつ $\mathbf{R} \in \{0,1\}^{m \times k}$ を満たす．

次に，署名同士や公開鍵同士の準同型演算を行う際に用いられるアルゴリズムを紹介する．これらのアルゴリズムは，入力として受け取る計算モデルによって出力される行列のノルムが異なる．

サーキットを用いるものは以下のようになる．

定理 3 (準同型演算 [GSW13])**.** 格子パラメータ n, q, m, ノルムの上限値 β，サーキットの深さの上限値 d，メッセージ長 ℓ が，$m = O(n \log q)$ かつ $\beta \cdot 2^{\tilde{O}(d)} < q$ を満たしているとする．このとき，以下の性質を満たす多項式時間アルゴリズム $(\mathsf{EvalPKC}, \mathsf{EvalUC})$ が存在する．

- $\mathsf{EvalPKC}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C) \to \mathbf{V}_C$: 行列 $\mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$ とサーキット $C : \{0,1\}^\ell \to \{0,1\}$ を入力にとり，行列 $\mathbf{V}_C \in \mathbb{Z}_q^{n \times m}$ を出力する．
- $\mathsf{EvalUC}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), C) \to \mathbf{U}_C$: $\mathbf{V}_i \in \mathbb{Z}_q^{n \times m}, x_i \in \{0,1\}$ かつ $\mathbf{U}_i \in \mathbb{Z}_q^{m \times m}$ を満たす行列とメッセージの組 $(\mathbf{V}_i, x_i, \mathbf{U}_i)$ と，サーキット $C : \{0,1\}^\ell \to \{0,1\}$ を入力にとり，行列 $\mathbf{U}_C \in \mathbb{Z}_q^{m \times m}$ を出力する．
- $\forall i \in [\ell]$ に対して $\mathbf{A}\mathbf{U}_i + x_i \cdot \mathbf{G} = \mathbf{V}_i$ かつ $\|\mathbf{U}_i\|_\infty \leq \beta$ を満たす任意の行列 $\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$，$\mathbf{U}_1, \ldots, \mathbf{U}_\ell \in \mathbb{Z}_q^{m \times m}$，任意の入力 $x_1, \ldots, x_\ell \in \{0,1\}$ と，深さが d 以下の任意のサーキット $C : \{0,1\}^\ell \to \{0,1\}$ に対して，$\mathbf{V}_C \leftarrow \mathsf{EvalPKC}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C)$，$\mathbf{U}_C \leftarrow \mathsf{EvalUC}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell \mathbf{U}_\ell), C)$ とするとき，$\mathbf{A}\mathbf{U}_C + C(x) \cdot \mathbf{G} = \mathbf{V}_C$ かつ $\|\mathbf{U}_C\|_\infty \leq \beta \cdot 2^{\tilde{O}(d)} < q$ が成立する．

注意 1. メッセージ空間を $\{0,1\}$ ではなく，$\mathbb{Z}_q$ としても同様のアルゴリズムは構成可能である．しかし，その場合は EvalUC の出力である $\mathbf{U}_C$ のノルムが非常に大きくなるので，効率が大幅に悪くなってしまうという欠点がある．ただし，加算ゲートのみにより構成されるサーキットを入力に取る場合は，その欠点が解消される．加算ゲートのみにより構成されているサーキット $C^+ : \mathbb{Z}_q^\ell \to \mathbb{Z}_q$ と，$\forall i \in [\ell]$ に対して $\mathbf{A}\mathbf{U}_i + x_i \cdot \mathbf{G} = \mathbf{V}_i$ かつ $\|\mathbf{U}_i\|_\infty \leq \beta$ を満たす任意の行列 $\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$，$\mathbf{U}_1, \ldots, \mathbf{U}_\ell \in \mathbb{Z}_q^{m \times m}$，任意の入力 $x_1, \ldots, x_\ell \in \mathbb{Z}_q$ に対して，$\mathbf{V}_{C^+} \leftarrow \mathsf{EvalPKC}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C^+)$，$\mathbf{U}_{C^+} \leftarrow \mathsf{EvalUC}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell \mathbf{U}_\ell), C^+)$ とするとき，$\mathbf{A}\mathbf{U}_{C^+} + C^+(x) \cdot \mathbf{G} = \mathbf{V}_{C^+}$ かつ $\|\mathbf{U}_{C^+}\|_\infty \leq \beta \cdot |C^+|$ が成立する．サーキットサイズ $|C^+|$ は λ に対して多項式オーダーの大きさなので，$\mathbf{U}_{C^+}$ のノルムは，入力の行列のノルムに比べてさほど大きくなっていない．

一方，branching program を用いる場合の以下のようになる．

定理 4 (GV Homomorphic Operations[GV15])**.** 格子パラメータ n, q, m, ノルムの上限値 β，長さの上限値 L，メッセージ長 ℓ が，$m = O(n \log q)$ かつ $\beta \cdot O(mL) \leq q$ を満たしているとする．このとき，以下の性質を満たす多項式時間アルゴリズム $(\mathsf{EvalPKBP}, \mathsf{EvalUBP})$ が存在する．

- $\mathsf{EvalPKBP}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, BP) \to \mathbf{V}_{BP}$: 行列 $\mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$ と branching program $BP : \{0,1\}^\ell \to \{0,1\}$ を入力にとり，行列 $\mathbf{V}_{BP} \in \mathbb{Z}_q^{n \times m}$ を出力する．

注意 2. [GV15] では，引数に状態ベクトル $\mathbf{v}_0[i]$ に対応する行列 $\mathbf{V}_{0,i}$ も取っていたが，全て零行列などで初期化すると予め統一しておけば不要である．また，これは $\mathsf{EvalUBP}$ でも同様である．

- $\mathsf{EvalUBP}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), BP) \to \mathbf{U}_{BP}$: $\mathbf{V}_i \in \mathbb{Z}_q^{n\times m}, x_i \in \{0,1\}$ かつ $\mathbf{U}_i \in \mathbb{Z}_q^{m\times m}$ を満たす行列とメッセージの組 $(\mathbf{V}_i, x_i, \mathbf{U}_i)$ と，branching program $BP : \{0,1\}^\ell \to \{0,1\}$ を入力にとり，行列 $\mathbf{U}_{BP} \in \mathbb{Z}_q^{m\times m}$ を出力する．
- $\mathbf{A}\mathbf{U}_i + x_i \cdot \mathbf{G} = \mathbf{V}_i$ かつ $\|\mathbf{U}_i\|_\infty \le \beta\ \forall i \in [\ell]$ を満たす任意の行列 $\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n\times m}$, 任意の入力 $x_1, \ldots, x_\ell \in \{0,1\}$，任意の行列 $\mathbf{U}_1, \ldots, \mathbf{U}_\ell \in \mathbb{Z}_q^{m\times m}$ と，長さが L 以下の任意の branching program $BP : \{0,1\}^\ell \to \{0,1\}$ に対して，$\mathbf{V}_{BP} \leftarrow \mathsf{EvalPKBP}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, BP)$，$\mathbf{U}_{BP} \leftarrow \mathsf{EvalUBP}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell\, \mathbf{U}_\ell), BP)$ のとき，$\mathbf{A}\mathbf{U}_{BP} + BP(x) \cdot \mathbf{G} = \mathbf{V}_{BP}$ かつ $\|\mathbf{U}_{BP}\|_\infty \le \beta \cdot O(mL) \le q$ が成立する．

以上から，計算モデルを branching program にするとノルムの増加が抑えられることがわかる．6 章ではこれを利用して [GVW15] の方式よりも効率的な準同型署名を構成している．

第 3 章

準同型署名

次に，準同型署名のシンタクスなどの定義を行う．なお，本論文では，簡単のためデータセットが一つの場合のみを考える．

定義 5 (準同型署名)**.** ある $N \in \mathbb{N}$ に対してラベル空間を $\mathcal{T} = [N]$，メッセージ空間を $\mathcal{X}$，関数空間を $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ とする，ここで，$\mathcal{F}_\lambda$ は $\mathcal{X}^{\leq N}$ から $\mathcal{X}$ への関数である．このとき，準同型署名は以下の *6* つのアルゴリズムからなる．

- $\mathsf{PrmsGen}(1^\lambda, 1^N) \to \mathsf{pk}$: セキュリティパラメータ λ とラベル空間の大きさ N を入力にとり，公開鍵生成アルゴリズムは公開鍵 pk を出力する．
- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: セキュリティパラメータ λ を入力にとり，鍵生成アルゴリズムは検証鍵 vk と秘密鍵 sk を出力する．
- $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x) \to \sigma$: 公開鍵 pk，秘密鍵 sk，ラベル $\tau \in \mathcal{T}$，メッセージ $x \in \mathcal{X}$ を入力にとり，署名アルゴリズムは署名 σ を出力する．
- $\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (F, (\tau_1, \ldots, \tau_k))) \to \mathsf{pk}_F$: 公開鍵 pk と labeled program $\mathcal{P} \in \mathcal{F} \times \mathcal{T}^k$ を入力にとり，公開鍵演算アルゴリズムは公開鍵 pk_F を出力する．
- $\mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, F, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k))) \to \hat{\sigma}$: 公開鍵 pk, 検証鍵 vk, 関数 $F \in \mathcal{F}$, メッセージと署名の k 個の組 (x_i, σ_i) を入力にとり，署名演算アルゴリズムは署名 $\hat{\sigma}$ を出力する．
- $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) \to \{0, 1\}$: 公開鍵 pk'，検証鍵 vk，メッセージ $x \in \mathcal{X}$，署名 σ を入力にとり，検証アルゴリズムは 0 または 1 を出力する．

署名正当性. 署名アルゴリズムが正しく動作することを要求する．より正確には，全ての $\lambda \in \mathbb{N}, \tau \in \mathcal{T}, x \in \mathcal{X}, \mathsf{pk} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda), \sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$ に対して

$$\Pr[(\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma)) = 1] = 1$$

となることである．ただし，$\mathsf{pk}' \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{I}_\tau)$ とする．

演算正当性. 署名演算アルゴリズム，公開鍵演算アルゴリズムが正しく動作すること

を要求する．より正確には，全ての $\lambda \in \mathbb{N}$, $F \in \mathcal{F}$, $\mathsf{pk} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ に対して，$\mathsf{Verify}(\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}_i), \mathsf{vk}, x_i, \sigma_i) = 1$ を満たす k 個の組 $\{(\mathcal{P}_i, = (\mathcal{I}_i, \tau_i), x_i, \sigma_i)\}_{i=1,\ldots,k}$ が与えられたとき

$$\Pr[(\mathsf{Verify}(\mathsf{pk}_F, \mathsf{vk}, \hat{x}, \hat{\sigma}) = 1] = 1$$

となることである．ただし，$\hat{x} = F(x_1, \ldots, x_k), \mathsf{pk}_F \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = F(\mathcal{P}_1, \ldots, \mathcal{P}_k))$, $\hat{\sigma} \leftarrow \mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, F, (x_1, \sigma_1), \ldots (x_k, \sigma_k))$ とする．

安全性：偽造不可能性．準同型署名の安全性について，弱安全性と強安全性の二つを定義する．これらの安全性は，行う安全性ゲームは同じだが，攻撃者の出力が偽造として認められる条件 COND が異なる．

注意 3. 以下で我々が定義する弱適応的安全性は，[CFN18] では semi-adaptive security という名前が用いられていたことに注意されたい．

$\Pi = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Verify})$ を，準同型署名とする．このとき，攻撃者 $\mathcal{A}$ に対して安全性ゲームを以下の手順で行う．

セットアップ： チャレンジャは公開鍵 $\mathsf{pk} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$ の生成と，検証鍵と秘密鍵 $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ の生成を行う．そして，公開鍵 pk と検証鍵 vk を $\mathcal{A}$ に送る．最後に，集合 T を $T = \phi$ と初期化する．

署名クエリ： 攻撃者 $\mathcal{A}$ は適応的に $(\tau, x) \in \mathcal{T} \times \mathcal{X}$ の形式のクエリを送ることができる．クエリに対して，チャレンジャは以下のように対応する．

- $(\tau, \cdot)$ の形式のクエリが初めての場合[*2]，チャレンジャは署名 $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$ を計算し，集合 T を $T \leftarrow T \cup (\tau, x)$ のように更新し，署名 σ を $\mathcal{A}$ に送る．
- $(\tau, x) \in T$ の場合，チャレンジャは以前送った署名と同じものを送る．
- それ以外の場合，つまり $x' \neq x$ となる x' について $(\tau, x') \in T$ のとき，チャレンジャはクエリを無視する．

偽造： $\mathcal{A}$ は $(\mathcal{P}^* = (F^*, (\tau_1^*, \ldots, \tau_n^*)), x^*, \sigma^*)$ の組を出力する．このとき，以下の両方が成立するとき，$\mathcal{A}$ がゲームに勝利すると表現する．

- $\mathsf{Verify}(\mathsf{pk}_{F^*}, \mathsf{vk}, x^*, \sigma^*) = 1$（ただし，$\mathsf{pk}_{F^*} \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^*)$）
- 条件 COND を満たす．

定義 6 (弱適応的安全性)**.** 上の安全性ゲームにおける条件 COND が，以下の条件を満たす場合，弱適応的安全であると言う．

Type 1: $\forall i \in [n], \exists(\tau_i^*, x_i) \in T$ かつ $x^* \neq F^*(x_1, \ldots, x_n)$.

以後，上の条件を満たす偽造を Type 1 forgery という．また，$\mathbf{Adv}^{\mathsf{w\text{-}ad\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \Pr[\mathcal{A}$ が安全性ゲームに勝つ] とする．このとき，Π が弱適応的安全であるとは，任意の多項式

[*2] つまり，$\forall(\tau', x') \in T, \tau' \neq \tau$ の場合

時間アルゴリズム $\mathcal{A}$ に対して， $\mathbf{Adv}^{\mathsf{w\text{-}ad\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ であることをいう．

弱適応的安全性は $\mathcal{A}$ が最終的に出力する labeled program の全てのラベルについてクエリしていることを要求している． これは攻撃者に制限を設けているので，安全性としては弱いものであると言える．次に，この制限を取り払った，より強い安全性を考える．

定義 7 (強適応的安全性)．上の安全性ゲームにおける条件 COND が，以下の条件のどちらかを満たす場合，強適応的安全であると言う．
Type 1: $\forall i \in [n], \exists(\tau_i^*, x_i) \in T$ かつ $x^* \neq F^*(x_1, \ldots, x_n)$.
Type 2: $\exists j \in \{1, ..., n\}, (\tau_j^*, \cdot) \notin T$.
先ほどと同様に，Type 2 の条件を満たす偽造を Type 2 forgery という．また，
$\mathbf{Adv}^{\mathsf{s\text{-}ad\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \Pr[\mathcal{A}$ が安全性ゲームに勝つ確率$]$ とする．このとき，Π が強適応的安全であるとは，任意の多項式時間アルゴリズム $\mathcal{A}$ に対して， $\mathbf{Adv}^{\mathsf{s\text{-}ad\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ であることをいう．

次に，適応的ではなく選択的な安全性を定義する．この場合，安全性ゲームにおいて，攻撃者はセットアップのはじめにクエリするラベルとメッセージの組を全てチャレンジャに送る必要がある．

セットアップ： 攻撃者 $\mathcal{A}$ はラベルとメッセージの組の集合 $T = ((\tau_1, x_1), \ldots, (\tau_n, x_n)) \subset \mathcal{T} \times \mathcal{X}$ をチャレンジャに送る．ここで，$n = \mathsf{poly}(\lambda)$ とする． チャレンジャは $\exists i, j \in [n], \tau_i = \tau_j$ のとき 0 を出力する[*3] ．それ以外の場合，チャレンジャは公開鍵，秘密鍵と署名鍵を生成する $\mathsf{pk} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$，$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$．最後に，$\mathcal{A}$ に公開鍵 pk と検証鍵 vk を送る．

署名生成： チャレンジャは受け取った全てのクエリ $(\tau, x) \in T$ に対する署名 $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$ を計算し，署名 σ を $\mathcal{A}$ に送る．

偽造： $\mathcal{A}$ は $(\mathcal{P}^* = (F^*, (\tau_1^*, \ldots, \tau_n^*)), x^*, \sigma^*)$ の組を出力する．このとき，以下の両方が成立するとき，$\mathcal{A}$ がゲームに勝利すると表現する．

- $\mathsf{Verify}(\mathsf{pk}_{F^*}, \mathsf{vk}, x^*, \sigma^*) = 1$（ただし，$\mathsf{pk}_{F^*} \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^*)$）
- 条件 COND を満たす．

定義 8 (弱選択的安全性)．上の安全性ゲームにおける条件 COND が，以下の条件を満たす場合，弱選択的安全であると言う．
Type 1: $\forall i \in [n], \exists(\tau_i^*, x_i) \in T$ かつ $x^* \neq F^*(x_1, \ldots, x_n)$.
また，$\mathbf{Adv}^{\mathsf{w\text{-}sel\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \Pr[\mathcal{A}$ が安全性ゲームに勝つ$]$ とする．このとき，Π が弱選択的安全であるとは，任意の多項式時間アルゴリズム $\mathcal{A}$ に対して， $\mathbf{Adv}^{\mathsf{w\text{-}sel\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ であることをいう．

[*3] これは，先ほどと同様攻撃者は一つのラベルについて一つのメッセージに対する署名しか得られないことを意味する．

定義 9 (強選択的安全性)**.** 上の安全性ゲームにおける条件 COND が，以下の条件のどちらかを満たす場合，強選択的安全であると言う．

Type 1: $\forall i \in [n], \exists(\tau_i^*, x_i) \in T$ かつ $x^* \neq F^*(x_1, \ldots, x_n)$.

Type 2: $\exists j \in \{1, ..., n\}, (\tau_j^*, \cdot) \notin T$.

さらに $\mathbf{Adv}^{\mathsf{s\text{-}sel\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \Pr[\mathcal{A}$ が安全性ゲームに勝つ$]$ とする．このとき，Π が強選択的安全であるとは，任意の多項式時間アルゴリズム $\mathcal{A}$ に対して， $\mathbf{Adv}^{\mathsf{s\text{-}ad\text{-}UF}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ であることをいう．

第 4 章

弱安全から強安全へのジェネリックな安全性強化の提案

準同型署名の弱安全から強安全への強化は，弱適応的安全から強適応的安全への強化と，弱選択的安全から強選択的安全への強化の 2 種類がある．本章では前者の変換と後者の変換の両方を紹介する．

4.1 弱適応的安全から強適応的安全へのジェネリックな安全性強化

弱適応的安全から強適応的安全への安全性強化については，先行研究 [CFN18] が存在する．しかし，我々はその変換よりも効率的な変換を構成した．以下では，まず先行研究を簡単に紹介し，次に我々の変換を紹介する．

4.1.1 先行研究 [CFN18]

[CFN18] で示された変換方法は，関数空間が深さ d 以下のサーキット $\mathcal{C}$ で弱適応的安全な準同型署名と，メッセージ空間が $\mathbb{Z}_p$（$p > 2^d$）の弱適応的安全な線形準同型署名を用いて，関数空間が深さ d 以下のサーキット $\mathcal{C}$ で強適応的安全な準同型署名に変換するというものであった．この方式では，一つ目の方式でサーキット C を計算すると同時に，二つ目の方式で C とサイズが同じサーキットで，ゲートを全て加算にしたものを計算している．

4.1.2 提案方式

次に，我々の構成した変換を示す．この変換は，関数空間が深さ d 以下のサーキット $\mathcal{C}$ で弱適応的安全な準同型署名と，メッセージ空間が $\{0,1\}$ を含む 弱適応的安全な線形準同型署名を用いて，関数空間が深さ d 以下のサーキット $\mathcal{C}$ で強適応的安全な準同型署名に変換する．一つ目の準同型署名を Π，二つ目の準同型署名を Π_+ とし，最終的に構成される準同型署名を $\hat{\Pi}$

とする．最後に，$n \in \mathbb{N}$ に対してサーキット C_n^+ を定義する．このサーキットは入力 $\{0,1\}^n$ に対して，全ての入力を一回ずつ並列に足し合わせていくサーキットで，サイズが $n-1$ であるものとする．

提案方式の構成

- $\hat{\Pi}.\mathsf{PrmsGen}(1^\lambda, 1^N) \to \hat{\mathsf{pk}}$: 2 つの公開鍵生成アルゴリズム $\mathsf{pk} \leftarrow \Pi.\mathsf{PrmsGen}(1^\lambda, 1^N)$, $\mathsf{pk}_+ \leftarrow \Pi_+.\mathsf{PrmsGen}(1^\lambda, 1^N)$ を実行した後，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk}, \mathsf{pk}_+)$ を出力する．
- $\hat{\Pi}.\mathsf{KeyGen}(1^\lambda) \to (\hat{\mathsf{vk}}, \hat{\mathsf{sk}})$: 2 つの鍵生成アルゴリズム $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$, $(\mathsf{vk}_+, \mathsf{sk}_+) \leftarrow \Pi_+.\mathsf{KeyGen}(1^\lambda)$ を実行した後，検証鍵と秘密鍵の組 $(\hat{\mathsf{vk}}, \hat{\mathsf{sk}}) := ((\mathsf{vk}, \mathsf{vk}_+), (\mathsf{sk}, \mathsf{sk}_+))$ を出力する．
- $\hat{\Pi}.\mathsf{Sign}(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}, \tau, x) \to \hat{\sigma}$: 2 つの署名アルゴリズム $\sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$, $\sigma^+ \leftarrow \Pi_+.\mathsf{Sign}(\mathsf{pk}_+, \mathsf{sk}_+, \tau, 0)$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma^+)$ を出力する．
- $\hat{\Pi}.\mathsf{PrmsEval}(\hat{\mathsf{pk}}, \mathcal{P} = (C, (\tau_1, \ldots, \tau_k))) \to \hat{\mathsf{pk}}_C$: 2 つの公開鍵演算アルゴリズム $\mathsf{pk}_C \leftarrow \Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (C, (\tau_1, \ldots, \tau_k)))$, $\mathsf{pk}_{C_k^+} \leftarrow \Pi_+.\mathsf{PrmsEval}(\mathsf{pk}_+, \mathcal{P}_+ = (C_k^+, (\tau_1, \ldots, \tau_k)))$ を計算したあと，公開鍵 $\hat{\mathsf{pk}}_C := (\mathsf{pk}_C, \mathsf{pk}_{C_k^+})$ を出力する．
- $\hat{\Pi}.\mathsf{SigEval}(\hat{\mathsf{pk}}, \hat{\mathsf{vk}}, C, ((x_1, \hat{\sigma}_1), \ldots, (x_k, \hat{\sigma}_k))) \to \hat{\sigma}$: 入力の署名 $\hat{\sigma}_i = (\sigma_i, \sigma_i^+)$ に対して，2 つの署名演算アルゴリズム $\sigma \leftarrow \Pi.\mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, C, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k)))$, $\sigma^+ \leftarrow \Pi_+.\mathsf{SigEval}(\mathsf{pk}_+, \mathsf{vk}_+, C_k^+, ((0, \sigma_1^+), \ldots, (0, \sigma_k^+)))$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma_+)$ を出力する．
- $\hat{\Pi}.\mathsf{Verify}(\hat{\mathsf{pk}}', \hat{\mathsf{vk}}, x, \hat{\sigma}) \to \{0,1\}$: 入力の公開鍵 $\hat{\mathsf{pk}}' = (\mathsf{pk}', \mathsf{pk}'_+)$，署名 $\hat{\sigma} = (\sigma, \sigma^+)$ に対して，$\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$ かつ $\Pi_+.\mathsf{Verify}(\mathsf{pk}'_+, \mathsf{vk}_+, 0, \sigma^+) = 1$ の場合のみ，1 を出力する．

先行研究との比較

我々の方式と先行研究の方式の最大の違いは，Π_+ で SigEval や PrmsEval などを行う際に計算を行うサーキットの大きさにある．我々の方式では，追加で計算を行うサーキット C_k^+ のサイズは，元のサーキット C の入力ビット数 k と等しい．一方，先行研究の場合では，追加で計算を行うサーキットのサイズは，元のサーキット C のサイズ $|C|$ と等しい．これにより，例えば [CFW14] や [GVW15] のようにゲートごとに計算を行う準同型署名の場合，先行研究の変換を用いると Π_+ の PrmsEval や SigEval の計算量が $O(|C|)$ に比例するが，我々の変換では $O(k)$ にしか比例しない．また，[BF11a] や [GVW15] などの方式のように，演算後の署名サイズがサーキットサイズに依存する方式では，署名サイズも減少する．

また，メッセージ空間に対する制約も異なる．我々の方式では，Π_+ のメッセージ空間は $\{0,1\}$ を含むものであればなんでもよいが，先行研究では $p > 2^d$ となる p に対してメッセージ空間を $\mathbb{Z}_p$ としなければならない．よって，我々の方式では Π_+ のメッセージ空間に対する制約が緩いので，より広い範囲の準同型署名方式を，Π_+ として用いることができるようになる．例えば，[BF11b] の方式を [CFN18] の変換には用いることができないが，我々の変換に

は用いることができる．

安全性証明

この $\hat{\Pi}$ に対して，以下の定理が成立する．

定理 5. Π が関数空間が深さ d 以下のサーキット $\mathcal{C}$ で弱適応的安全な準同型署名であり，Π_+ がメッセージ空間が $\mathbb{Z}_2$ の弱適応的安全な線形準同型署名のとき，$\hat{\Pi}$ は関数空間が深さ d 以下のサーキット $\mathcal{C}$ で強適応的安全な準同型署名である．

証明. $\hat{\Pi}$ は明らかに署名正当性と演算正当性を満たすので，安全性証明だけを行う．

$\mathbf{Adv}^{\mathsf{s\text{-}ad\text{-}UF}}_{\mathcal{A},\hat{\Pi}}(\lambda)$ を，ある攻撃者 $\mathcal{A}$ が $\hat{\Pi}$ の強適応的安全性を破る確率，E_i を攻撃者 $\mathcal{A}$ が $\hat{\Pi}$ の Type i forgery を生成する事象とすると，以下の式が成立する．

$$\mathbf{Adv}^{\mathsf{s\text{-}ad\text{-}UF}}_{\mathcal{A},\hat{\Pi}}(\lambda) = \Pr[\mathsf{E}_1] + \Pr[\mathsf{E}_2]$$

この二つの項がどちらも negigible な関数であることを示すために，補題 2 と補題 3 の二つを示す．

補題 2. Π が弱適応的安全な準同型署名のとき，$\hat{\Pi}$ の *Type 1 forgery* を偽造できる多項式時間アルゴリズム $\mathcal{A}$ は存在しない．

証明. 上の補題を示すために，$\hat{\Pi}$ の Type 1 forgery を偽造できる攻撃者 $\mathcal{A}$ が存在するとき，$\mathcal{A}$ を用いて Π の Type 1 forgery を偽造できる攻撃者 $\mathcal{B}$ を構成する方法を示す．

セットアップ： Π のチャレンジャから pk, vk を受け取り，$\mathcal{B}$ は公開鍵 $\mathsf{pk}_+ \leftarrow \Pi_+.\mathsf{PrmsGen}(1^\lambda)$ と，検証鍵と秘密鍵 $(\mathsf{vk}_+, \mathsf{sk}_+) \leftarrow \Pi_+.\mathsf{KeyGen}(1^\lambda)$ を生成する．そして，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk}, \mathsf{pk}_+)$ と検証鍵 $\hat{\mathsf{vk}} = (\mathsf{vk}, \mathsf{vk}_+)$ を $\mathcal{A}$ に送る．

署名クエリ： $\mathcal{A}$ から署名クエリ (τ, x) を受け取ると，$\mathcal{B}$ は署名 $\sigma^+ \leftarrow \Pi_+.\mathsf{Sign}(\mathsf{pk}_+, \mathsf{sk}_+, \tau, 0)$ を生成する．また，Π のチャレンジャに署名クエリ (τ, x) を送り，署名 σ を受け取る．そして，署名の組 $\hat{\sigma} = (\sigma, \sigma^+)$ を $\mathcal{A}$ に送る．

偽造： 最後に，$\mathcal{A}$ が $\hat{\Pi}$ の Type 1 forgery である $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ を出力した後，$\mathcal{B}$ は $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \sigma)$ を出力する．

$\mathcal{B}$ の出力が Π の Type 1 forgery であることを示す．$\mathcal{A}$ の出力である $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ は $\hat{\Pi}$ の Type 1 forgery なので，$\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$（ただし，$\mathsf{pk}' \leftarrow \Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)))$）を満たす．また，$\forall i \in [k], \exists(\tau_i^*, x_i) \in T$，かつ $x^* \neq C(x_1, \ldots, x_k)$ も満たす．よって，$\mathcal{B}$ の出力 $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \sigma)$ は Π の Type 1 forgery になっている．

以上より，$\Pr[\mathsf{E}_1] = \mathbf{Adv}^{\mathsf{w\text{-}ad\text{-}UF}}_{\mathcal{B},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ となる．

□

補題 3. Π_+ が弱適応的安全な準同型署名のとき，$\hat{\Pi}$ の *Type 2 forgery* を偽造できる多項式時間アルゴリズム $\mathcal{A}$ は存在しない．

証明. 上の補題を示すために，$\hat{\Pi}$ の Type 2 forgery を偽造できる攻撃者 $\mathcal{A}$ が存在するとき，$\mathcal{A}$ を用いて Π_+ の Type 1 forgery を偽造できる攻撃者 $\mathcal{B}$ を構成する方法を示す．

セットアップ： Π_+ のチャレンジャから $\mathsf{pk}_+, \mathsf{vk}_+$ を受け取り，$\mathcal{B}$ は公開鍵 $\mathsf{pk} \leftarrow \Pi.\mathsf{PrmsGen}(1^\lambda)$ と，検証鍵と秘密鍵 $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$ を生成する．そして，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk}, \mathsf{pk}_+)$ と検証鍵 $\hat{\mathsf{vk}} = (\mathsf{vk}, \mathsf{vk}_+)$ を $\mathcal{A}$ に送る．

署名クエリ： $\mathcal{A}$ から署名クエリ (τ, x) を受け取ると，$\mathcal{B}$ は署名 $\sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$ を生成する．また，Π_+ のチャレンジャに署名クエリ $(\tau, 0)$ を送り，署名 σ^+ を受け取る．そして，署名の組 $\hat{\sigma} = (\sigma, \sigma^+)$ を $\mathcal{A}$ に送る．

偽造： $\mathcal{A}$ は $\hat{\Pi}$ の Type 2 forgery $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ を出力する．ここで，集合 $(\tau_1, \ldots, \tau_k)$ に含まれ，かつ $\mathcal{A}$ が署名クエリを行なっていないラベルの集合を $\mathcal{J}$ とする．このとき，Type 2 forgery の定義より，この集合は空でない．よって，$\mathcal{B}$ は $\mathcal{J}$ の中から一つのラベル $\hat{\tau}$ を選び，Π_+ のチャレンジャに署名クエリ $(\hat{\tau}, 1), \{(\tau, 0)\}_{\tau \in \mathcal{J} \wedge \tau \neq \hat{\tau}}$ を行う．そして，$\mathcal{B}$ は $(\mathcal{P}^* = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)), 0, \sigma^+)$ を出力する．

$\mathcal{B}$ の出力が Π_+ の Type 1 forgery であることを示す．$\mathcal{A}$ の出力 $(\mathcal{P}^* = (C, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ は $\hat{\Pi}$ の Type 2 forgery なので，$\Pi.\mathsf{Verify}(\mathsf{pk}'_+, \mathsf{vk}_+, 0, \sigma^+) = 1$（ただし，$\mathsf{pk}'_+ \leftarrow \Pi_+.\mathsf{PrmsEval}(\mathsf{pk}_+, \mathcal{P}_+ = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)))$ を満たす．次に，$\mathcal{B}$ は $\tau_1^*, \ldots \tau_k^*$ の全てのラベルを Π_+ のチャレンジャにクエリしている．最後に，$C_k^+(x_1, \ldots, x_k) \neq 0$ を示す．いま $\hat{\tau}$ に対応するメッセージだけが 1 であり，他のラベルに対応するメッセージの値は 0 なので，$C_k^+(x_1, \ldots, x_k) = 1 \neq 0$ となる．よって，$\mathcal{B}$ の出力 $(\mathcal{P}^* = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)), 0, \sigma^+)$ は Π_+ の Type 1 forgery になっている．

以上より，$\Pr[\mathsf{E}_2] = \mathbf{Adv}^{\mathsf{w\text{-}ad\text{-}UF}}_{\mathcal{B}, \Pi_+}(\lambda) = \mathsf{negl}(\lambda)$ となる． □

以上，補題 2 と補題 3 より $\Pr[\mathsf{E}_1]$ と $\Pr[\mathsf{E}_2]$ が $\mathsf{negl}(\lambda)$ であることが示されたので，定理 5 が示される． □

4.2 弱選択的安全から強選択的安全へのジェネリックな安全性強化

弱選択的安全性から強選択的安全性への変換については，先行研究は存在しない．現在までに構成されている準同型署名方式はすべて適応的安全を実現しているので，この変換が必要でなかったからである．しかし，我々は 6 章で [GVW15] の方式を基にして，新しい準同型署名方式を構成する．この方式は [GVW15] の方式よりも難しい SIS 問題に安全性帰着を行えるという長所があるものの，弱選択的安全性しか満たしていないという欠点がある．よって，この方式にも使える変換として，弱選択的安全性から強選択的安全性への変換をここで示す．

4.2.1　提案方式

関数空間が $\mathcal{F}$ に対する弱選択的安全な準同型署名を Π とおく．また，メッセージ空間が $\mathbb{Z}_p$（$p > N$）の弱選択的安全な線形準同型署名を Π_+ とおく．この Π と Π_+ を用いて，関数空間が $\mathcal{F}$ に対する強選択的安全な準同型署名 $\hat{\Pi}$ を構成する．

提案方式の構成

- $\hat{\Pi}.\mathsf{PrmsGen}(1^\lambda, 1^N) \to \hat{\mathsf{pk}}$: 2 つの公開鍵生成アルゴリズム $\mathsf{pk} \leftarrow \Pi.\mathsf{PrmsGen}(1^\lambda, 1^N)$, $\mathsf{pk}_+ \leftarrow \Pi_+.\mathsf{PrmsGen}(1^\lambda, 1^N)$ を実行した後，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk}, \mathsf{pk}_+)$ を出力する．
- $\hat{\Pi}.\mathsf{KeyGen}(1^\lambda) \to (\hat{\mathsf{vk}}, \hat{\mathsf{sk}})$: 2 つの鍵生成アルゴリズム $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$, $(\mathsf{vk}_+, \mathsf{sk}_+) \leftarrow \Pi_+.\mathsf{KeyGen}(1^\lambda)$ を実行した後，検証鍵と秘密鍵の組 $(\hat{\mathsf{vk}}, \hat{\mathsf{sk}}) := ((\mathsf{vk}, \mathsf{vk}_+), (\mathsf{sk}, \mathsf{sk}_+))$ を出力する．
- $\hat{\Pi}.\mathsf{Sign}(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}, \tau, x) \to \hat{\sigma}$: 2 つの署名アルゴリズム $\sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$, $\sigma^+ \leftarrow \Pi_+.\mathsf{Sign}(\mathsf{pk}_+, \mathsf{sk}_+, \tau, 0)$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma^+)$ を出力する．
- $\hat{\Pi}.\mathsf{PrmsEval}(\hat{\mathsf{pk}}, \mathcal{P} = (F, (\tau_1, \ldots, \tau_k))) \to \hat{\mathsf{pk}}_F$: 2 つの公開鍵演算アルゴリズム $\mathsf{pk}_F \leftarrow \Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (F, (\tau_1, \ldots, \tau_k)))$, $\mathsf{pk}_{C_k^+} \leftarrow \Pi_+.\mathsf{PrmsEval}(\mathsf{pk}_+, \mathcal{P}_+ = (C_k^+, (\tau_1, \ldots, \tau_k)))$ を計算したあと，公開鍵 $\hat{\mathsf{pk}}_F := (\mathsf{pk}_F, \mathsf{pk}_{C_k^+})$ を出力する．
- $\hat{\Pi}.\mathsf{SigEval}(\hat{\mathsf{pk}}, \hat{\mathsf{vk}}, F, ((x_1, \hat{\sigma}_1), \ldots, (x_k, \hat{\sigma}_k))) \to \hat{\sigma}$: 入力の署名 $\hat{\sigma}_i = (\sigma_i, \sigma_i^+)$ に対して，2 つの署名演算アルゴリズム $\sigma \leftarrow \Pi.\mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, F, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k)))$, $\sigma^+ \leftarrow \Pi_+.\mathsf{SigEval}(\mathsf{pk}_+, \mathsf{vk}_+, C_k^+, ((0, \sigma_1^+), \ldots, (0, \sigma_k^+)))$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma_+)$ を出力する．
- $\hat{\Pi}.\mathsf{Verify}(\hat{\mathsf{pk}}', \hat{\mathsf{vk}}, x, \hat{\sigma}) \to \{0, 1\}$: 入力の公開鍵 $\hat{\mathsf{pk}}' = (\mathsf{pk}', \mathsf{pk}'_+)$，署名 $\hat{\sigma} = (\sigma, \sigma^+)$ に対して，$\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$ かつ $\Pi_+.\mathsf{Verify}(\mathsf{pk}'_+, \mathsf{vk}_+, 0, \sigma^+) = 1$ の場合のみ，1 を出力する．

安全性証明

$\hat{\Pi}$ に対して，以下の定理が成立する．

定理 6. Π が関数空間 $\mathcal{F}$ に対する弱選択的安全な準同型署名であり，Π_+ がメッセージ空間が $\mathbb{Z}_p$（$p > N$）の弱適応的安全な線形準同型署名のとき，$\hat{\Pi}$ は関数空間 $\mathcal{F}$ に対する強選択的安全な準同型署名である．

$\hat{\Pi}$ は明らかに署名正当性と演算正当性を満たすので，安全性のみを示す．$\mathbf{Adv}^{\mathsf{s\text{-}sel\text{-}UF}}_{\mathcal{A},\hat{\Pi}}(\lambda)$ をある攻撃者 $\mathcal{A}$ が $\hat{\Pi}$ の stong selective 安全性を破る確率，E_i を攻撃者 $\mathcal{A}$ が $\hat{\Pi}$ の Type i forgery を偽造する事象とすると，以下の式が成立する．

$$\mathbf{Adv}^{\mathsf{s\text{-}sel\text{-}UF}}_{\mathcal{A},\hat{\Pi}}(\lambda) = \Pr[\mathsf{E}_1] + \Pr[\mathsf{E}_2]$$

この二つの項がどちらも negligible な関数であることを示すために，以下の二つの補題を示す．

補題 4. Π が弱選択的安全な準同型署名のとき，$\hat{\Pi}$ の *Type 1 forgery* を偽造できる PPT アルゴリズム $\mathcal{A}$ は存在しない．

証明. 上の補題を示すために，$\hat{\Pi}$ の Type 1 forgery を偽造できる攻撃者 $\mathcal{A}$ が存在するとき，$\mathcal{A}$ を用いて Π の Type 1 forgery を偽造できる攻撃者 $\mathcal{B}$ を構成する方法を示す．

セットアップ： $\mathcal{A}$ から $n \leq N$ を満たす n に対して $T = ((\tau_1, x_1), \ldots, (\tau_n, x_n)) \subset \mathcal{T} \times \mathcal{X}$ を受け取った後，$\mathcal{B}$ は T を Π のチャレンジャに送る．次に，Π のチャレンジャから pk, vk を受け取り，$\mathcal{B}$ は公開鍵 $\mathsf{pk}_+ \leftarrow \Pi_+.\mathsf{PrmsGen}(1^\lambda)$ と，検証鍵と秘密鍵 $(\mathsf{vk}_+, \mathsf{sk}_+) \leftarrow \Pi_+.\mathsf{KeyGen}(1^\lambda)$ を生成する．最後に，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk}, \mathsf{pk}_+)$ と検証鍵 $\hat{\mathsf{vk}} = (\mathsf{vk}, \mathsf{vk}_+)$ を $\mathcal{A}$ に送る．

署名生成： まず，$\mathcal{B}$ は Π のチャレンジャから，送った全てのクエリ $(\tau, x) \in T$ に対応する署名 σ を受け取る．次に，受け取った全てのクエリ $(\tau, x) \in T$ に対応する署名 $\sigma^+ \leftarrow \Pi_+.\mathsf{Sign}(\mathsf{pk}_+, \mathsf{sk}_+, \tau, 0)$ を生成する．最後に，受け取った全てのクエリ $(\tau, x) \in T$ に対応する署名の組 $\hat{\sigma} = (\sigma, \sigma^+)$ を $\mathcal{A}$ に送る．

偽造： 最後に，$\mathcal{A}$ が $\hat{\Pi}$ の Type 1 forgery である $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ を出力した後，$\mathcal{B}$ は $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \sigma)$ を出力する．

$\mathcal{B}$ の出力が Π の Type 1 forgery であることを示す．$\mathcal{A}$ の出力である $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ は $\hat{\Pi}$ の Type 1 forgery なので，$\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$（ただし，$\mathsf{pk}' \leftarrow \Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)))$），かつ $\forall i \in [k], \exists (\tau_i^*, x_i) \in T$，かつ $x^* \neq BP(x_1, \ldots, x_k)$ を満たす．よって，$\mathcal{B}$ の出力 $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \sigma)$ は Π の Type 1 forgery になっている．

以上より，$\Pr[\mathsf{E}_1] = \mathbf{Adv}^{\mathsf{w\text{-}sel\text{-}UF}}_{\mathcal{B},\Pi}(\lambda) = \mathsf{negl}(\lambda)$ となる． □

補題 5. Π_+ がメッセージ空間が $\mathbb{Z}_p$（$p > N$）の弱選択的安全な線形準同型署名のとき，$\hat{\Pi}$ の *Type 2 forgery* を偽造できる PPT アルゴリズム $\mathcal{A}$ は存在しない．

証明. 上の補題を示すために，$\hat{\Pi}$ の Type 2 forgery を偽造できる攻撃者 $\mathcal{A}$ が存在するとき，$\mathcal{A}$ を用いて Π_+ の Type 1 forgery を偽造できる攻撃者 $\mathcal{B}$ を構成する方法を示す．

セットアップ： $\mathcal{A}$ から $n < N$ を満たす n に対して $T = ((\tau_1, x_1), \ldots, (\tau_n, x_n)) \subset \mathcal{T} \times \mathcal{X}$ を受け取った後，$T' = ((\tau_1, 0), \ldots, (\tau_n, 0), \{(\tau', 1)\}_{(\tau', \cdot) \notin T \wedge \tau' \in \mathcal{T}})$ を Π_+ のチャレンジャに送る[*4]．次に，Π_+ のチャレンジャから $\mathsf{pk}_+, \mathsf{vk}_+$ を受け取り，$\mathcal{B}$ は公開鍵 $\mathsf{pk} \leftarrow \Pi.\mathsf{PrmsGen}(1^\lambda)$ と，秘密鍵と検証鍵 $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ を生成する．最後に，公開鍵 $\hat{\mathsf{pk}} = (\mathsf{pk} = \mathsf{pk}_+)$ と検証鍵 $\hat{\mathsf{vk}} = (\mathsf{vk}, \mathsf{vk}_+)$ を $\mathcal{A}$ に送る．

署名生成： まず，$\mathcal{B}$ は Π_+ のチャレンジャから，送った全てのクエリ $(\tau, x) \in T'$ に対応する署名 σ_+ を受け取る．次に，受け取った全てのクエリ $(\tau, x) \in T$ に対応する署名 $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$ を生成する．最後に，受け取った全てのクエリ $(\tau, x) \in T$ に対応

[*4] ラベル空間の大きさは λ に対して多項式オーダーなので，これが可能である．

する署名の組 $\hat{\sigma} = (\sigma, \sigma_+)$ を $\mathcal{A}$ に送る．

偽造：　最後に，$\mathcal{A}$ が $\hat{\Pi}$ の Type 2 forgery である $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ を出力した後，$\mathcal{B}$ は $(\mathcal{P}^* = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)), 0, \sigma^+)$ を出力する．

$\mathcal{B}$ の出力が Π_+ の Type 1 forgery であることを示す．

$\mathcal{A}$ の出力である $(\mathcal{P}^* = (F, (\tau_1^*, \ldots, \tau_k^*)), x^*, \hat{\sigma} = (\sigma, \sigma^+))$ は $\hat{\Pi}$ の Type 2 forgery なので，以下の二点を満たす．まず，$\Pi_+.\mathsf{Verify}(\mathsf{pk}'_+, \mathsf{vk}_+, 0, \sigma^+) = 1$（ただし，$\mathsf{pk}'_+ \leftarrow \Pi_+.\mathsf{PrmsEval}(\mathsf{pk}_+, \mathcal{P} = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)))$ である．次に，$\mathcal{J} \subset (\tau_1^*, \ldots, \tau_k^*)$ かつ $\forall \tau^* \in \mathcal{J}, (\tau^*, \cdot) \notin T$ となる空でない集合 $\mathcal{J}$ が存在する．

ここで，　$\mathcal{J}$ に属するラベルに対応するメッセージは全て 1 なので，$C_k^+(x_1, \ldots, x_k) > 0$ となる．また，ラベル空間の上限は N なので，$C_k^+(x_1, \ldots, x_k) \leq N$ である．よって，$C_k^+(x_1, \ldots, x_k) \neq 0 \mod p$ となる．

最後に，$\mathcal{B}$ は全てのラベルについてクエリを行なっているので，$\forall i \in [k], \exists (\tau_i^*, x_i) \in T'$ である．よって，$\mathcal{B}$ の出力 $(\mathcal{P}^* = (C_k^+, (\tau_1^*, \ldots, \tau_k^*)), 0, \sigma^+)$ は Π_+ の Type 1 forgery になっている．

以上より，$\Pr[\mathsf{E}_2] = \mathbf{Adv}^{\mathsf{w\text{-}sel\text{-}UF}}_{\mathcal{B}, \Pi_+}(\lambda) = \mathsf{negl}(\lambda)$ となる．　□

以上，補題 4 と補題 5 より，$\Pr[\mathsf{E}_1]$ と $\Pr[\mathsf{E}_2]$ が $\mathsf{negl}(\lambda)$ であることが示されたので，定理 6 が示される．

第 5 章

格子の代数構造を用いたより効率的な安全性強化の提案

この章では，現時点で存在する唯一の完全準同型署名であり，格子ベースで構成された弱適応的安全性を満たす Gorbunov ら [GVW15] の方式に対して，弱適応的安全から強適応的安全への安全性強化を行う．そして，格子の構造をうまく利用することで，この変換が前章で示したジェネリック変換よりも効率よく行えることを示す．

効率化に関するアイデアは二つである．一つ目は，Π_+ の $\mathsf{SigEval}, \mathsf{PrmsEval}$ では加算ゲートのみにより構成されているサーキットしか計算を行わないことに着目することで，Π_+ で用いるパラメータを小さいものに設定することができるというものである．これにより，Π_+ における署名や公開鍵のサイズを小さくできる．二つ目は，異なるパラメータを用いる Π と Π_+ において，お互いの剰余 q と q_+ に対して，$q/q_+ \in \mathbb{N}$ という関係を設定しておくことで，Π_+ の公開鍵を Π の公開鍵から計算することができるというものである．これにより，$\mathsf{PrmsGen}$ で Π_+ の公開鍵を生成する必要がなくなり，全体の公開鍵サイズが減少する．以下の 5.1 節では GVW 方式の構成と一つ目のアイデアについて説明し，次の 5.2 節では安全性の強化と二つ目のアイデアについて説明する．

5.1 GVW 方式

まずはじめに，従来の弱適応的安全性をもつ，パラメータ $\mathbf{P} = (n, m, q, d, \beta, \beta_{max}, \beta_{\mathsf{SIS}})$ を用いる GVW の準同型署名方式の構成を示す[*5]．次節以降で，これを Π として用いる．メッセージ空間は $\{0,1\}$ を含み，関数空間は深さ d 以下のサーキット $\mathcal{C}$ である．また，任意のサーキット $C : \{0,1\}^k \to \{0,1\}$ と行列 $\mathbf{V}_i \in \mathbb{Z}_q^{n \times m}$ に対して，$C' : \{0,1\}^{knm\lceil \log q \rceil} \to \{0,1\}^{nm\lceil \log q \rceil}$ を $C'(\mathsf{bin}(\mathbf{V}_1)^{*6}, \ldots, \mathsf{bin}(\mathbf{V}_k)) = \mathsf{bin}(\mathsf{EvalPKC}(\mathbf{V}_1, \ldots, \mathbf{V}_k, C))$ を満たすサーキットとする．

以下の準同型署名方式は，$\mathsf{SIS}(n, m, q, \beta_{\mathsf{SIS}})$ 困難性仮定が成立するとき，弱適応的安全性を

[*5] 同論文中で構成している，選択的安全性から適応的安全性への変換を施した後の方式である．また，シンタクスが若干異なるので，本論文に合わせている．

[*6] ここで，bin とは入力をあらかじめ定められた方法でビット分解する関数とする．

満たすことが知られている．

- $\Pi.\mathsf{PrmsGen}(1^\lambda, 1^N) \to \mathsf{pk}$: 公開鍵生成アルゴリズムは $\forall i \in [N]$ に対して，行列 $\vec{\mathbf{V}}_i = (\mathbf{V}_{i,1,1,1}, \ldots, \mathbf{V}_{i,n,m,\lceil \log q \rceil}) \leftarrow (\mathbb{Z}_q^{n\times m})^{nm\lceil \log q \rceil}$ をサンプルし，公開鍵 $\mathsf{pk} = (\vec{\mathbf{V}}_1, \ldots, \vec{\mathbf{V}}_N)$ を出力する．
- $\Pi.\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: 鍵生成アルゴリズムは 2 つの行列とトラップドアの組 $(\mathbf{A_1}, \mathsf{td}_1) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $(\mathbf{A_2}, \mathsf{td}_2) \leftarrow \mathsf{TrapGen}(1^\lambda)$ を生成し，検証鍵 $\mathsf{vk} = (\mathbf{A_1}, \mathbf{A_2}) \in (\mathbb{Z}_q^{n\times m})^2$，秘密鍵 $\mathsf{sk} = (\mathbf{A_1}, \mathbf{A_2}, \mathsf{td}_1, \mathsf{td}_2) \in (\mathbb{Z}_q^{n\times m})^2 \times (\mathbb{Z}_q^{m\times m})^2$ を出力する．
- $\Pi.\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x) \to \sigma$: 署名アルゴリズムは行列 $\mathbf{T}_\tau \leftarrow \mathbb{Z}_q^{n\times m}$ をランダムに選び，$\mathbf{R}_\tau \leftarrow \mathsf{SamplePre}(\mathbf{A_2}, \mathbf{T}_\tau - x\cdot\mathbf{G}, \mathsf{td}_2)$ を計算する．次に，$\forall i \in [n], j \in [m], k \in \lceil \log q \rceil$ に対して，$\vec{\mathbf{U}}_\tau \leftarrow \mathsf{SamplePre}(\mathbf{A_1}, \vec{\mathbf{V}}_\tau - \mathsf{bin}(\mathbf{T}) \otimes \cdot\mathbf{G}, \mathsf{td}_1)$ を計算する．最後に，署名 $\sigma = (\mathbf{T}_\tau, \mathbf{R}_\tau, \vec{\mathbf{U}}_\tau) \in \mathbb{Z}_q^{n\times m} \times (\mathbb{Z}_q^{m\times m})^{1+nm\lceil \log q \rceil}$ を出力する．
- $\Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (C, (\tau_1, \ldots, \tau_k))) \to \mathsf{pk}_C$: 公開鍵演算アルゴリズムは公開鍵同士の準同型演算 $\vec{\mathbf{V}}_{C'} \leftarrow \mathsf{EvalPKC}(\vec{\mathbf{V}}_{\tau_1}, \ldots, \vec{\mathbf{V}}_{\tau_k}, C')$ を計算し，公開鍵 $\mathsf{pk}_C = \vec{\mathbf{V}}_{C'} \in \mathbb{Z}_q^{n\times m}$ を出力する．
- $\Pi.\mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, C, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k))) \to \hat{\sigma}$: 署名演算アルゴリズムは，署名 $\sigma_i = (\mathbf{T}_i, \mathbf{R}_i, \vec{\mathbf{U}}_i)$ を分割し，$\mathbf{T}^* \leftarrow \mathsf{EvalPKC}(\mathbf{T}_1, \ldots, \mathbf{T}_k, C)$,
  $\mathbf{R}^* \leftarrow \mathsf{EvalUC}((\mathbf{T}_1, x_1, \mathbf{R}_1), \ldots, (\mathbf{T}_k, x_k, \mathbf{R}_k), C)$,
  $\vec{\mathbf{U}}^* \leftarrow \mathsf{EvalUC}((\vec{\mathbf{V}}_1, \mathsf{bin}(\mathbf{T}_1), \vec{\mathbf{U}}_1), \ldots, (\vec{\mathbf{V}}_k, \mathsf{bin}(\mathbf{T}_k), \vec{\mathbf{U}}_k), C')$ を計算する．最後に，署名 $\hat{\sigma} = (\mathbf{T}^*, \mathbf{R}^*, \vec{\mathbf{U}}^*) \in \mathbb{Z}_q^{n\times m} \times (\mathbb{Z}_q^{m\times m})^{1+nm\lceil \log q \rceil}$ を出力する．
- $\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) \to \{0, 1\}$: 検証アルゴリズムは $\mathsf{pk}' = \vec{\mathbf{V}}'$ とし，署名 $\sigma = (\mathbf{T}, \mathbf{R}, \vec{\mathbf{U}})$ を分割する．そして，以下の条件が成り立つときのみ 1 を出力する．
  - $\mathbf{A_2}\mathbf{R} + x\cdot\mathbf{G} = \mathbf{T}$，$\mathbf{A_1}\vec{\mathbf{U}} + \mathsf{bin}(\mathbf{T}) \otimes \mathbf{G} = \vec{\mathbf{V}}'$
  - $\|\mathbf{R}\|_\infty \le \beta_{max}$，$\|\vec{\mathbf{U}}\|_\infty \le \beta_{max}$

以上の方式が正当性を満たすためのパラメータの条件は，$m = O(n\log q), \beta = O(n\sqrt{\log q}), \beta_{max} = \beta m^{\tilde{O}(d\log n)}, q > \beta_{\mathsf{SIS}} = (2m+1)\beta_{max}$ となることである．このとき，他の条件が成立することは明らかであるため，$\|\vec{\mathbf{U}}\|_\infty \le \beta_{max}$ を満たすことを示す．行列の乗算を計算するサーキットは，$\mathcal{NC}^1$ 回路で構成できるので，$2nm\log q$ 個の入力に対して，深さ $\tilde{O}(\log n)$ で構成できる．元のサーキット C の深さの上限が d なので，サーキット C' の深さの上限は $\tilde{O}(d\log n)$ である．深さ c のサーキットに対して，ノルムは最大で m^c 倍になるので，$\|\vec{\mathbf{U}}\|_\infty \le \beta m^{\tilde{O}(d\log n)} \le \beta_{max}$ となり，正当性を満たす．以上の方式は深さが d 以下の任意のサーキットを対象にしているので，ノルムの増大を抑えるために $\mathbf{T}_\tau$ をビット分解してサーキット C' に対応する署名 $\vec{\mathbf{U}}^*$ を計算する必要があった．しかし，C' が加算ゲートのみを含むサーキットの場合，注意 1 で述べた通り，$\mathbf{T}_\tau$ を $\mathbb{Z}_q$ のまま保持しながら $\vec{\mathbf{U}}^*$ を計算したとしてもノルムの増大を抑えることができる．

よって，我々の一つ目のアイデアは，この性質を利用して，パラメータサイズを削減することである．具体的には，SigEval や PrmsEval で計算するサーキットが加算ゲートの

みから構成される場合，[GVW15] で与えられた構成よりも効率的な構成を与える．メッセージ空間を $\mathbb{Z}_q$ とし，関数空間をサイズが N 以下[*7]で，加算ゲートしか持たないサーキット $\mathcal{C}^+$ とする．また，サーキット $C^+ : (\mathbb{Z}_q)^k \to \mathbb{Z}_q$ に対し，$C^{+'} : \mathbb{Z}_q^{knm} \to \mathbb{Z}_q^{nm}$ を $C^{+'}(\mathsf{dec}(\mathbf{V}_1)^{*8}, \ldots, \mathsf{dec}(\mathbf{V}_k)) = \mathsf{dec}(\mathsf{EvalPKC}(\mathbf{V}_1, \ldots, \mathbf{V}_k, C^+))$ を満たすサーキットとする．そして，上記の GVW 方式の Sign において署名を $\mathbf{T}$ の各成分の各ビットごとに生成していた部分を，各成分ごとに生成するように変更した方式を考える．この方式が正当性を満たすための条件は，パラメータ $\mathbf{P}' = (n, m_+, q_+, N, \beta^+, \beta^+_{max}, \beta^+_{\mathsf{SIS}})$ が，$m_+ = O(n \log q_+), \beta^+ = O(n\sqrt{\log q_+}), \beta^+_{max} = \beta \cdot N, q_+ > \beta^+_{\mathsf{SIS}} = (2m_+ + 1)\beta^+_{max}$ となることである．先ほどと同様，$\|\vec{\mathbf{U}}\|_\infty \leq \beta_{max}$ を満たすことのみ示す．行列の加算は，各成分ごとに足し合わせることで計算できるので，$\vec{\mathbf{U}}^* \leftarrow \mathsf{EvalUC}((\vec{\mathbf{V}}_1, \mathsf{dec}(\mathbf{T}_1), \vec{\mathbf{U}}_1), \ldots, (\vec{\mathbf{V}}_k, \mathsf{dec}(\mathbf{T}_k), \vec{\mathbf{U}}_k), C^{+'})$ としたときの $\vec{\mathbf{U}}^*$ の各成分のノルムは，$\vec{\mathbf{U}}_i$ の各成分のノルムを，$|C^+|$ 回足し合わせたものに等しい．よって，$\|\vec{\mathbf{U}}\|_\infty \leq \beta \cdot N \leq \beta_{max}$ となり，正当性を満たす．このとき，$\mathbf{P}'$ のパラメータは $\mathbf{P}$ のものよりも小さくなっており，これにより署名や公開鍵のサイズが小さくなる．よって，次節以降でこの方式を Π_+ として用いる．

5.2 提案方式の構成

前節で示した Π と Π_+ を用いて，強適応的安全な準同型署名 $\hat{\Pi}$ を構成する．ただし，$\mathbf{P}$ の q と $\mathbf{P}'$ の q_+ は $q/q_+ \in \mathbb{N}$ を満たすとする．

- $\hat{\Pi}.\mathsf{PrmsGen}(1^\lambda, 1^N) \to \hat{\mathsf{pk}}$: 公開鍵生成アルゴリズム $\mathsf{pk} \leftarrow \Pi.\mathsf{PrmsGen}(1^\lambda, 1^N)$ を実行し，公開鍵 $\hat{\mathsf{pk}} = \mathsf{pk}$ を出力する．
- $\hat{\Pi}.\mathsf{KeyGen}(1^\lambda) \to (\hat{\mathsf{vk}}, \hat{\mathsf{sk}})$: 2 つの鍵生成アルゴリズム $(\mathsf{vk}, \mathsf{sk}) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$, $(\mathsf{vk}_+, \mathsf{sk}_+) \leftarrow \Pi_+.\mathsf{KeyGen}(1^\lambda)$ を実行した後，検証鍵と秘密鍵の組 $(\hat{\mathsf{vk}}, \hat{\mathsf{sk}}) := ((\mathsf{vk}, \mathsf{vk}_+), (\mathsf{sk}, \mathsf{sk}_+))$ を出力する．
- $\hat{\Pi}.\mathsf{Sign}(\hat{\mathsf{pk}}, \hat{\mathsf{sk}}, \tau, x) \to \hat{\sigma}$: まずはじめに，$\forall i \in [N], \forall j \in [n], \forall k \in [m_+], \forall r \in [n], \forall s \in [m_+]$ に対して，$(\mathbf{V}^+_{i,j,k})_{r,s} = (\mathbf{V}_{i,j,k,1})_{r,s} \mod q_+$ となる $\vec{\mathbf{V}}^+_i = (\vec{\mathbf{V}}_{i,1,1} \ldots, \vec{\mathbf{V}}_{i,n,m}) \in (\mathbb{Z}_{q_+}^{n \times m_+})^{nm_+}$ を求め，$\mathsf{pk}_+ = (\vec{\mathbf{V}}^+_1, \ldots, \vec{\mathbf{V}}^+_N)$ とする（以後，この操作を（$*$）で示す）．次に，2 つの署名アルゴリズム $\sigma \leftarrow \Pi.\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$, $\sigma^+ \leftarrow \Pi_+.\mathsf{Sign}(\mathsf{pk}_+, \mathsf{sk}_+, \tau, 0)$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma^+)$ を出力する．
- $\hat{\Pi}.\mathsf{PrmsEval}(\hat{\mathsf{pk}}, \mathcal{P} = (C, (\tau_1, \ldots, \tau_k))) \to \hat{\mathsf{pk}}_C$: まずはじめに，（$*$）の操作を行う．次に，2 つの公開鍵演算アルゴリズム $\mathsf{pk}_C \leftarrow \Pi.\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (C, (\tau_1, \ldots, \tau_k)))$, $\mathsf{pk}_{C^+_k} \leftarrow \Pi_+.\mathsf{PrmsEval}(\mathsf{pk}_+, \mathcal{P}_+ = (C^+_k, (\tau_1, \ldots, \tau_k)))$ を計算したあと，公開鍵 $\hat{\mathsf{pk}}_C := (\mathsf{pk}_C, \mathsf{pk}_{C^+_k})$ を出力する．
- $\hat{\Pi}.\mathsf{SigEval}(\hat{\mathsf{pk}}, \hat{\mathsf{vk}}, C, ((x_1, \hat{\sigma}_1, \ldots, (x_k, \hat{\sigma}_k)))) \to \hat{\sigma}$: まずはじめに，（$*$）の

[*7] Π_+ で計算するサーキットのサイズは Π 計算するサーキットの入力に等しいので，N より大きくなることはない．

[*8] ここで，dec とは入力の行列を成分ごとに分解する関数とする．

操作を行う．次に，入力の署名 $\hat{\sigma}_i = (\sigma_i, \sigma_i^+)$ に対して，2 つの署名演算アルゴリズム $\sigma \leftarrow \Pi.\mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, C, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k)))$，$\sigma_+ \leftarrow \Pi_+.\mathsf{SigEval}(\mathsf{pk}_+, \mathsf{vk}_+, C_k^+, ((0, \sigma_1^+), \ldots, (0, \sigma_k^+)))$ を計算した後，署名 $\hat{\sigma} := (\sigma, \sigma^+)$ を出力する．

- $\hat{\Pi}.\mathsf{Verify}(\hat{\mathsf{pk}}', \hat{\mathsf{vk}}, x, \hat{\sigma}) \to \{0, 1\}$: 入力の公開鍵 $\hat{\mathsf{pk}}' = (\mathsf{pk}', \mathsf{pk}'_+)$，署名 $\hat{\sigma} = (\sigma, \sigma^+)$ に対して，$\Pi.\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$ かつ $\Pi_+.\mathsf{Verify}(\mathsf{pk}'_+, \mathsf{vk}_+, 0, \sigma^+) = 1$ の場合のみ，1 を出力する．

5.3　ジェネリック変換との比較

我々の方式と，ジェネリック変換との効率の比較を行う．まず，ジェネリック変換では $|\hat{\mathsf{pk}}| = |\mathsf{pk}| + |\mathsf{pk}_+|$ であったが，我々の方式では Π_+ の公開鍵は Π の公開鍵から計算できるので，$|\hat{\mathsf{pk}}| = |\mathsf{pk}|$ である．よって，公開鍵のサイズが減少している．

また，Π_+ のパラメータも小さくなっている．我々の方式において，Π_+ のパラメータを大まかに評価すると，$m_+ = n(\log N + \log n), q_+ = n^3 N$ となる．一方，ジェネリック変換の場合の Π_+ のパラメータを評価すると，$m'_+ = n \log N \log n \log(n \log N), q'_+ = n^{\log N \log(n \log N)}$ となる．このパラメータの大きさの違いにより，演算後の署名サイズなどが小さくなる．具体的には，我々の方式における演算後の署名サイズの最大値 σ_{max} は，$\sigma_{max} = n m_+^3 (\log N + \log n)$ となるが，ジェネリック変換の場合の演算後の署名サイズの最大値 σ'_{max} は，$\sigma'_{max} = n(m'_+)^3 \log q'_+ (\log m'_+)^2 (\log N)$ となる．したがって，

$$\frac{\sigma'_{max}}{\sigma_{max}} = \min(\log N (\log n)^{10}, (\log N)^5 (\log n)^6)$$

となり，署名サイズが減少している．

5.3.1　安全性の証明

定理 7. Π が関数空間が深さ d 以下のサーキット $\mathcal{C}$ で弱適応的安全な準同型署名であり，Π_+ が関数空間がサイズ N 以下で加算ゲートのみにより構成されるサーキット $\mathcal{C}^+$ で弱適応的安全な準同型署名であるとすると，$\hat{\Pi}$ は関数空間が深さ d 以下のサーキット $\mathcal{C}$ で強適応的安全な準同型署名である．

正当性については，5.1 節で述べたので省略する．

次に，安全性の証明であるが，上の方式と 4 章で示したジェネリックな方式との違いは，公開鍵を 1 組しか作らないという一点だけなので，証明はほぼ同じである．よって，以下の二つの補題により安全性が証明される．ただし，補題 6 は補題 2 と同様に示せるので証明は省略する．

補題 6. Π_+ が関数空間がサイズ N 以下で加算ゲートのみにより構成されるサーキット $\mathcal{C}^+$ で弱適応的安全な準同型署名のとき，$\hat{\Pi}$ の *Type 1 forgery* を偽造できる多項式時間アルゴリズ

ム $\mathcal{A}$ は存在しない.

補題 7. Π_+ が関数空間がサイズ N 以下で加算ゲートのみにより構成されるサーキット $\mathcal{C}^+$ で弱適応的安全な準同型署名のとき，$\hat{\Pi}_+$ の *Type 2 forgery* を偽造できる多項式時間アルゴリズム $\mathcal{A}$ は存在しない.

証明. この補題の証明と，補題 3 の証明の唯一の違いは，Π_+ のチャレンジャから $\mathsf{pk}_+ = (\vec{\mathbf{V}}_1^+, \ldots, \vec{\mathbf{V}}_N^+) \in (\mathbb{Z}_{q_+}^{n\times m_+})^{nm_+}$ を受け取った後, $\mathsf{pk} = (\vec{\mathbf{V}}_1, \ldots, \vec{\mathbf{V}}_N) \in (\mathbb{Z}_q^{n\times m})^{nm\lceil \log q\rceil}$ を生成しなければならないことだけである．これは，以下のようにして生成できる．まず，$1 \le k \le m_+$ のとき，$\forall i \in [N], \forall j \in [n]$ に対して，$\hat{\mathbf{V}}_{i,j,k,1} \leftarrow \mathbb{Z}_{q/q_+}^{n\times m_+}, \tilde{\mathbf{V}}_{i,j,k} \leftarrow \mathbb{Z}_q^{n\times(m-m_+)}$ をサンプルし[*9]，$\mathbf{V}_{i,j,k,1} = (\mathbf{V}_{i,j,k}^+ + q_+\hat{\mathbf{V}}_{i,j,k} || \tilde{\mathbf{V}}_{i,j,k}) \in \mathbb{Z}_q^{n\times m}$ とする．ここで，$\mathbf{V}_{i,j,k}^+$ は $(\mathbb{Z}_{q_+}^{n\times m_+})$ 上の一様ランダムな分布に従っているので，$\mathbf{V}_{i,j,k,1}$ は $\mathbb{Z}_q^{n\times m}$ 上の一様ランダムな分布に従うことに注意されたい．次に，$m_+ < k \le m$ のとき，$\forall i \in [N], \forall j \in [n], 2 \le \forall \ell \le [\lceil \log q\rceil]$ に対して，$\mathbf{V}_{i,j,k,\ell} \leftarrow \mathbb{Z}_q^{n\times m}$ とする.

以上により生成された $\mathsf{pk} = (\vec{\mathbf{V}}_1, \ldots, \vec{\mathbf{V}}_N)$ は $(\mathbb{Z}_q^{n\times m})^{nm\lceil \log q\rceil}$ 上の一様ランダムな分布に従っており，$\Pi_+.\mathsf{KeyGen}(1^\lambda)$ で生成された pk と同じ分布となる．また，$\forall i \in [N], \forall j \in [n], \forall k \in [m_+], \forall r \in [n], \forall s \in [m_+]$ に対して，$(\mathbf{V}_{i,j,k}^+)_{r,s} = (\mathbf{V}_{i,j,k,1})_{r,s} \mod q_+$ となる．よって，あとは補題 3 と同様にして示せる. □

[*9] $\hat{\mathbf{V}}_{i,j,k}$ のサンプルは $q/q_+ \in \mathbb{N}$ なのでこれが可能である

第 6 章

多項式剰余を実現する準同型署名の構成

前章で示した GVW 方式には，剰余 q の大きさが λ に対して準指数オーダーの大きさが必要であるという欠点があった．よって，本章では，多項式剰余を実現する準同型署名の構成を示す．この構成は，計算モデルとして branching program を用いることで，剰余 q の大きさが多項式オーダーの SIS 問題に帰着することが可能となる．ただし，この構成は計算モデルとして branching program を用いていること以外は [GVW15] のものとほとんど同じであり，またこのアイデア自身は [GV15] で示されている．

6.1 提案方式

ラベル空間のサイズを $N = \mathsf{poly}(\lambda)$ とする． メッセージ空間 $\mathcal{X}$ は $\{0,1\}$ を含むものとする．$\mathcal{BP}_\lambda : \{0,1\}^{\leq N} \to \{0,1\}$ を長さが $L_{max} = \mathsf{poly}(\lambda)$ 以下，幅が 5 以下[*10]の branching program の集合とし，関数空間を $\mathcal{BP} = \{\mathcal{BP}_\lambda\}_{\lambda\in\mathbb{N}}$ とする．

各パラメータを，$n = \mathsf{poly}(\lambda), m = O(n\log q), \beta = O(n\log q), \beta_{max} \geq \beta\cdot(2mL_{max}+1), \beta_{SIS} > \beta_{max}\cdot(2m+1), q = \mathsf{poly}(\lambda) > \beta_{SIS}$ と設定する．

このとき，準同型署名方式 $\Pi = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Verify})$ は以下のように構成される．

- $\mathsf{PrmsGen}(1^\lambda, 1^N) \to \mathsf{pk}$: 公開鍵生成アルゴリズムは行列 $\mathbf{V}_1, \ldots, \mathbf{V}_N \leftarrow \mathbb{Z}_q^{n\times m}$ をサンプルし，公開鍵 $\mathsf{pk} = (\mathbf{V}_1, \ldots, \mathbf{V}_N)$ を出力する．
- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: 鍵生成アルゴリズムは行列とトラップドアの組 $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ を生成し，検証鍵 $\mathsf{vk} = \mathbf{A} \in \mathbb{Z}_q^{n\times m}$，秘密鍵 $\mathsf{sk} = (\mathbf{A}, \mathsf{td}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{m\times m}$ を出力する．
- $\mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x) \to \sigma$: 署名鍵生成アルゴリズムは原像 $\mathbf{U}_\tau \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}_\tau - x\cdot\mathbf{G}, \mathsf{td})$ をサンプルし，署名 $\sigma = \mathbf{U}_\tau \in \mathbb{Z}_q^{m\times m}$ を出力する．

[*10] 定理 1 より，幅が 5 あればクラス $NC1$ の関数が計算できる．

- $\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (BP, (\tau_1, \ldots, \tau_k))) \rightarrow \mathsf{pk}_{BP}$: 公開鍵 pk と labeled program $\mathcal{P} \in \mathcal{BP} \times \mathcal{T}^k$ を入力に，公開鍵演算アルゴリズムは公開鍵同士の準同型演算を行い $\mathbf{V}_{BP} \leftarrow \mathsf{EvalPKBP}(\mathbf{V}_{\tau_1}, \ldots, \mathbf{V}_{\tau_k}, BP)$，公開鍵 $\mathsf{pk}_{BP} = \mathbf{V}_{BP} \in \mathbb{Z}_q^{n \times m}$ を出力する．
- $\mathsf{SigEval}(\mathsf{vk}, BP, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k))) \rightarrow \hat{\sigma}$: 入力が $BP \notin \mathcal{BP}$ もしくは $\exists i \in [k], (x_i, \sigma_i) \notin \mathcal{X} \times \mathbb{Z}_q^{m \times m}$ ならば，署名アルゴリズムは停止する．それ以外の場合，公開鍵 pk，検証鍵 vk，branching program $BP \in \mathcal{BP}$，k 個のメッセージと署名の組 $(x_i, \sigma_i = \mathbf{U}_i) \in \mathcal{X} \times \mathbb{Z}_q^{m \times m}$ を入力に，$\forall i \in [k]$ に対して $\tilde{\mathbf{V}}_i = \mathbf{A}\mathbf{U}_i + x_i \cdot \mathbf{G}$ とする．そして，署名同士の準同型演算 $\mathbf{U}_{BP} \leftarrow \mathsf{EvalUBP}((\tilde{\mathbf{V}}_1, x_1, \mathbf{U}_1), \ldots, (\tilde{\mathbf{V}}_k, x_k, \mathbf{U}_k), BP)$ を行い，署名 $\hat{\sigma} = \mathbf{U}_{BP} \in \mathbb{Z}_q^{m \times m}$ を出力する．
- $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) \rightarrow \{0, 1\}$: $\sigma \notin \mathbb{Z}_q^{m \times m}$ の場合，検証アルゴリズムは停止する．それ以外の場合，公開鍵 $\mathsf{pk}' = \mathbf{V}_{BP} \in \mathbb{Z}_q^{n \times m}$，検証鍵 $\mathsf{vk} = \mathbf{A} \in \mathbb{Z}_q^{n \times m}$，メッセージ $x \in \mathcal{X}$，署名 $\sigma = \mathbf{U} \in \mathbb{Z}_q^{m \times m}$ を入力として受け取り，検証アルゴリズムは $\mathbf{A}\mathbf{U} = \mathbf{V}_{BP} - x \cdot \mathbf{G} \wedge \|\mathbf{U}\|_\infty \leq \beta_{max}$ のときのみ 1 を出力する．

まず，上で構成した準同型署名について，署名正当性と演算正当性が成り立つことを示す．

署名正当性． 任意の $\lambda \in \mathbb{N}$, ラベル $\tau \in \mathcal{T}$, メッセージ $x \in \mathcal{X}$ に対して，$\sigma = \mathbf{U}_\tau \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, \tau, x)$, $\mathbf{A}\mathbf{U}_\tau = \mathbf{V}_\tau - x \cdot \mathbf{G}$ かつ $\mathbf{V}_{BP} = \mathsf{pk}' \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{I}_\tau) = \mathsf{EvalPKBP}(\mathbf{V}_\tau, g_{\mathrm{id}}) = \mathbf{V}_\tau$ となるので，$\mathbf{A}\sigma = \mathbf{V}_{BP} - x \cdot \mathbf{G}$ を満たす．また，$\mathbf{U}_\tau \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}_\tau - x \cdot \mathbf{G}, \mathsf{td})$, $\|\mathbf{U}_\tau\|_\infty \leq \beta \leq \beta_{max}$ となるので，以下の式が成り立つ．

$$\Pr[(\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma)) = 1] = 1$$

演算正当性． 任意の $\lambda \in \mathbb{N}$, メッセージ $x_i \in \mathcal{X}$, ラベル $\tau_i \in \mathcal{T}$, $BP \in \mathcal{BP}$ と $\mathsf{Verify}(\mathsf{PrmsEval}(\mathsf{pk}, \mathcal{I}_{\tau_i}), vk, x_i, \sigma_i) = 1$ を満たす署名 σ_i に対して，$\mathbf{V}_{BP} = \mathsf{pk}_{BP} \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P} = (BP, (\tau_1, \ldots, \tau_k))) = \mathsf{EvalPKBP}(\mathbf{V}_{\tau_1}, \ldots, \mathbf{V}_{\tau_k}, BP)$, $\hat{\sigma} \leftarrow \mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, BP, (x_1, \sigma_1), \ldots (x_k, \sigma_k)) = \mathsf{EvalUBP}((\tilde{\mathbf{V}}_1, x_1, \mathbf{U}_1), \ldots, (\tilde{\mathbf{V}}_k, x_k, \mathbf{U}_k), BP)$, $\hat{x} = BP(x_1, \ldots, x_k)$ としたとき，定理 4 より $\mathbf{A}\hat{\sigma} = \mathbf{V}_{BP} - x \cdot \mathbf{G}$ かつ $\|\hat{\sigma}\|_\infty \leq \beta \cdot (2mL_{max} + 1) \leq \beta_{max}$ となる．よって，以下の式が成り立つ．

$$\Pr[(\mathsf{Verify}(\mathsf{pk}_{BP}, vk, \hat{x}, \hat{\sigma}) = 1] = 1$$

次に，安全性について以下の定理が成り立つことを示す．

定理 8. $\mathsf{SIS}(n, m, q, \beta_{SIS})$ 困難性仮定が成り立つとき，Π は弱選択的安全な準同型署名である．

証明. Π が弱選択的安全でないとする．ここで定義より，ある攻撃者 A が存在して，下の安全性ゲームを行ったとき，$\mathsf{pk}_{BP^*} \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^* = (BP^*, (\tau_1^*, \ldots, \tau_k^*))$ に対して $\mathsf{Verify}(\mathsf{pk}_{BP^*}, vk, x^*, \sigma^*) = 1$ かつ $x^* \neq BP^*(x_1, \ldots, x_n)$ となる $(\mathcal{P}^* = (BP^*, (\tau_1^*, \ldots, \tau_n^*)), x^*, \sigma^*)$ を出力できる確率が λ に対して non-negligible となる．

セットアップ：　攻撃者 $\mathcal{A}$ が集合 $T = ((\tau_1, x_1), \ldots, (\tau_n, x_n)) \subset \mathcal{T} \times \mathcal{X}$ をチャレンジャに送る．チャレンジャは $\exists i, j \in [n], \tau_i = \tau_j$ のとき動作を停止する．それ以外の場合，チャレンジャは行列 $(\mathbf{V}_1, \ldots, \mathbf{V}_N) \leftarrow \mathbb{Z}_q^{n \times m}$ をサンプルし，公開鍵を $\mathsf{pk} = (\mathbf{V}_1, \ldots, \mathbf{V}_N)$. とする．次に，チャレンジャは行列とトラップドアの組 $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ をサンプルし，検証鍵と秘密鍵を $\mathsf{vk} = \mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathsf{sk} = (\mathbf{A}, \mathsf{td}) \in \mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^{m \times m}$ とする．最後に，公開鍵 pk と検証鍵 vk を $\mathcal{A}$ に送る．

署名生成：　チャレンジャは受け取った全てのクエリ $(\tau, x) \in T$ に対応する署名 $\sigma = \mathbf{U}_\tau \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}_\tau - x \cdot \mathbf{G}, \mathsf{td})$ を計算し，それらを $\mathcal{A}$ に送る．

偽造：　$\mathcal{A}$ は $(\mathcal{P}^*, x^*, \sigma^*)$ の組を出力する．

次に，上の安全性ゲームから，チャレンジャの動作を少し変更した新しい安全性ゲームを定義する．

セットアップ：　攻撃者 $\mathcal{A}$ が集合 $T = ((\tau_1, x_1), \ldots, (\tau_n, x_n)) \subset \mathcal{T} \times \mathcal{X}$ をチャレンジャに送る．チャレンジャは $\exists i, j \in [n], \tau_i = \tau_j$ のとき動作を停止する．それ以外の場合，チャレンジャは行列 $\mathbf{A}' \leftarrow \mathbb{Z}_q^{n \times m}$ をサンプルし，$\|\mathbf{U}'\|_\infty \leq \beta$ を満たす N 個の行列 $(\mathbf{U}'_1, \ldots, \mathbf{U}'_N) \leftarrow \mathbb{Z}_q^{m \times m}$ をサンプルする．次に，チャレンジャは以下の計算を行う．

- $(\tau, x) \in T$ に対して，$\mathbf{V}'_\tau = \mathbf{A}'\mathbf{U}'_\tau + x \cdot \mathbf{G}$.
- $(\tau, \cdot) \notin T \wedge \tau \in [N]$ に対して，$b \leftarrow \{0, 1\}$ として，$\mathbf{V}'_\tau = \mathbf{A}'\mathbf{U}'_\tau + b \cdot \mathbf{G}$.

最後に，$\mathsf{pk} = (\mathbf{V}'_1, \ldots, \mathbf{V}'_N)$, $\mathsf{vk} = \mathbf{A}'$ として，pk, vk を $\mathcal{A}$ に送る．

署名生成：　チャレンジャは受け取った全てのクエリ $(\tau, x) \in T$ に対して $\sigma = \mathbf{U}'_\tau$ とし，σ を $\mathcal{A}$ に送る．

偽造：　$\mathcal{A}$ は $(\mathcal{P}^*, x^*, \sigma^*)$ の組を出力する．

この新しいゲームにおいて，チャレンジャは署名を先にランダムに選んでいるので，署名を生成する秘密鍵を必要としない．ここで，攻撃者 $\mathcal{A}$ が前者の安全性ゲームで得る情報は $(\mathbf{A}, \{\mathbf{U}_{\tau_i}\}_{(\tau_i, \cdot) \in T}, \{\mathbf{V}_i\}_{i \in [N]})$ であり，後者の安全性ゲームで得る情報は $(\mathbf{A}', \{\mathbf{U}'_{\tau_i}\}_{(\tau_i, \cdot) \in T}, \{\mathbf{V}'_i\}_{i \in [N]})$ であるが，定理 2 よりこの二つの組は統計的に識別不可能である．よって，前者の安全性ゲームで署名の偽造に non-negligible な確率で成功する攻撃者は，後者の安全性ゲームでも署名の偽造に non-negligible な確率で成功する．

次に，その攻撃者を用いて，SIS 困難性仮定を破る攻撃者 $\mathcal{B}$ を構成する．まず，SIS 問題として行列 $\mathbf{A} \in \mathbb{Z}_q^{m \times m}$ を受け取った $\mathcal{B}$ は，その行列を $\mathbf{A}'$ として，$\mathcal{A}$ と後者の安全性ゲームを行う．これが可能なのは，前述の通り，このゲームにおいてチャレンジャは署名を生成する秘密鍵を必要としないからである．

このとき，$\mathcal{A}$ の出力 $(\mathcal{P}^* = (BP^*, (\tau_1^*, \ldots, \tau_n^*)), x^*, \sigma^*)$ は以下の式を満たす．

$$\mathbf{A}\sigma^* + x^* \cdot \mathbf{G} = \mathbf{V}$$

ただし，$\mathbf{V} = \mathsf{pk}_{BP^*} \leftarrow \mathsf{PrmsEval}(\mathsf{pk}, \mathcal{P}^* = (BP^*, (\tau_1^*, \ldots, \tau_k^*)))$ である．

一方，$\sigma \leftarrow \mathsf{SigEval}(\mathsf{pk}, \mathsf{vk}, BP^*, ((x_1, \sigma_1), \ldots, (x_k, \sigma_k))$ を計算すると，演算正当性より以

下の式が成立する．

$$\mathbf{A}\sigma + BP^*(x_1, \ldots, x_n) \cdot \mathbf{G} = \mathbf{V}$$

上の二式の両辺を引いて $\hat{\mathbf{U}} = \sigma - \sigma^*, \hat{x} = x^* - BP^*(x_1, \ldots, x_n)$ とすると，

$$\mathbf{A}\hat{\mathbf{U}} = \hat{x} \cdot \mathbf{G}$$

ここで，$x^* \neq BP^*(x_1, \ldots, x_n)$ より，$\hat{x} = x^* - BP^*(x_1, \ldots, x_n) \neq 0$ であり，また，$\|\sigma\|_\infty, \|\sigma^*\|_\infty \leq \beta_{max}$ より $\|\hat{\mathbf{U}}\|_\infty \leq 2\beta_{max}$ である．次に，$\mathcal{B}$ は m 次元ベクトルをサンプルし $\mathbf{r} \leftarrow \{0,1\}^m$，$\mathbf{z} = \mathbf{A}\mathbf{r} \in \mathbb{Z}_q^n, \mathbf{r}' = \mathbf{G}^{-1}(\hat{x}^{-1} \cdot \mathbf{z})$ とする．ここで，

$$\mathbf{A}(\hat{\mathbf{U}}\mathbf{r}' - \mathbf{r}) = \hat{x} \cdot \mathbf{G} \cdot \mathbf{G}^{-1}(\hat{x}^{-1} \cdot \mathbf{z}) - \mathbf{z} = \mathbf{0}.$$

となるので，$\mathbf{u} = \hat{\mathbf{U}}\mathbf{r}' - \mathbf{r} \in \mathbb{Z}_q^m$ とすると，$\mathbf{A}\mathbf{u} = \mathbf{0}$ かつ $\|\mathbf{u}\|_\infty \leq (2m+1)\beta_{max} < \beta_{\mathsf{SIS}}$ を満たす．

最後に，$\mathbf{u} \neq \mathbf{0}$ であること，つまり $\hat{\mathbf{U}}\mathbf{r}' \neq \mathbf{r}$ であることを，条件付き最小エントロピーを用いて示す．$\mathbf{r}'$ は $\mathbf{z} = \mathbf{A}\mathbf{r}$ の値に依存しているので，$\mathbf{H}_\infty(\mathbf{r}|\mathbf{r}') \geq \mathbf{H}_\infty(\mathbf{r}|\mathbf{A}\mathbf{r})$ となる．補題 1 より，S を $\mathbf{z}$ の値域として，$\mathbf{H}_\infty(\mathbf{r}|\mathbf{A}\mathbf{r}) \geq \mathbf{H}_\infty(r) - \log(|S|)$ が成り立つ．$\mathbf{z} \in \mathbb{Z}_q^n$ より，$|S| = q^n$ であり， $\log(|S|) = \log(q^n) = n \log q$ である．一方，$\mathbf{H}_\infty(r) = -\log 2^{-m} = m$ なので，

$$\mathbf{H}_\infty(\mathbf{r}|\mathbf{r}') \geq \mathbf{H}_\infty(\mathbf{r}|\mathbf{A}\mathbf{r}) \geq \mathbf{H}_\infty(r) - \log(S) \geq m - n \log q = \omega(\log \lambda).$$

定義 1 で述べたとおり，確率変数 Y の値が与えられたとき，それに関連する確率変数 X の値を当てる確率は $2^{-\mathbf{H}_\infty(X|Y)}$ 以下なので，$\Pr[\mathbf{r} = \hat{\mathbf{U}}\mathbf{r}'] \leq 2^{-\mathbf{H}_\infty(\mathbf{r}|\mathbf{r}')} \leq 2^{-\omega \log(\lambda)} = \mathsf{negl}(\lambda)$ である．

以上より，Π の弱選択的安全性を non-negligible の確率で破ることのできる攻撃者が存在すると仮定すると，SIS 困難性仮定を破る攻撃者が存在することを示した．よって，SIS 困難性仮定の下では，Π は弱選択的安全性をもつ． □

6.2 安全性の強化

この方式は，GVW 方式とは違い，選択的安全から適応的安全への変換を行うと計算可能な関数のクラスに大きな制限をかけてしまう．現時点では計算モデルが branching program のままで適応的安全へ変換する方法を構成できておらず，変換のためにはまず計算を行う関数をサーキットで表して，[GVW15] が構成した適応的安全への変換を行い，定理 1 を用いてサーキットを branching program に変換するという手順を行う必要がある．しかし，5.2 節で示したように，適応的安全への変換を行うためには，入力にとるサーキット C よりも深さの大きいサーキット C' に対して準同型演算を行う必要があった．一般には，サーキット C' の深さは C の深さの $O(\log \lambda)$ 倍になる．よって，計算を行うサーキット C がクラス $\mathcal{NC}^1$ のとき，C' はクラス $\mathcal{NC}^2$ となる．ここで, 定理 1 より，クラス $\mathcal{NC}^2$ 以上のサーキットにバリントンの変換を用いると，branching program の長さは多項式オーダーを超えるので，計算が多項

式時間で終わらない．よって，計算を行うサーキット C' はクラス $\mathcal{NC}^1$ 以下でなければならず，そのためには入力のサーキット C の深さは定数オーダーでなければならない．このように，適応的安全への変換を行うと計算を行う関数に強い制約をかけてしまうので，状況によって変換を行わない場合が考えられる．その場合は，4.2 節で示した，弱選択的安全から強選択的安全への変換を用いることで，強安全性を実現することが可能である．

第 7 章

結論

我々は，準同型署名の弱安全性から強安全性への変換として，適応的な場合と選択的な場合において，ジェネリックなものを構成した．特に，前者の変換は先行研究 [CFN18] よりも効率的なものになっており，方式によっては SigEval と PrmsEval の計算量や，SigEval を行なった後の署名サイズが減少する．また，後者の変換については先行研究がなく，我々が初めて構成した．

次に，格子の構造をうまく利用することで，弱適応的安全を満たす [GVW15] の方式に対しては，ジェネリックなものよりもさらに効率的な強適応的安全への変換を行えることも示した．この変換は，一方の準同型署名方式で用いる公開鍵を，もう一方の準同型署名方式で用いる公開鍵から生成することにより，全体の公開鍵サイズが減少する．また，前者の方式で用いるパラメータを後者のものより小さい値に設定できることを示した．これに従うことで，SigEval を行なった後の署名サイズなどがさらに減少する．

最後に，[GVW15] の方式を改良し，計算モデルを branching program にすることで多項式剰余の準同型署名方式を構成した．これは [GVW15] の方式とは異なり弱選択的安全しか満たさないという欠点があるものの，我々が初めて構成した弱選択的安全から強選択的安全への変換を用いることで，強安全性を実現することができる．

謝辞

本研究を進めるにあたり，指導教員の國廣昇准教授には大変お世話になりました．厚く感謝を申し上げます．また，同研究室で博士課程に在籍中の勝又秀一さんにも多くのアドバイスをいただきました．心より感謝いたします．

参考文献

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, pages 99–108. ACM, 1996.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, pages 75–86. IBFI Schloss Dagstuhl, 2009.

[Bar89] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168. Springer, 2011.

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, pages 1–16. Springer, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *ITCS*, pages 1–12. ACM, 2014.

[CFN18] Dario Catalano, Dario Fiore, and Luca Nizzardo. On the security notions for homomorphic signatures. In *ACNS*, pages 183–201. Springer, 2018.

[CFW11] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Adaptive pseudo-free groups and applications. In *EUROCRYPT*, pages 207–223. Springer, 2011.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, pages 371–389. Springer, 2014.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540. Springer, 2004.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, pages 697–714. Springer, 2012.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. ACM, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92. Springer, 2013.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: efficient abe for branching programs. In *ASIACRYPT*, pages 550–574. Springer, 2015.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, pages 469–477. ACM, 2015.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, pages 301–320. Springer, 2013.

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer, 2002.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718. Springer, 2012.