暗号回路・ハッシュ回路に対する
スキャンベース攻撃に関する研究

Scan-based Side-channel Attacks
against Cryptographic and Hash Function
Integrated Circuits

2020年7月

於久 太祐

Daisuke OKU

暗号回路・ハッシュ回路に対する スキャンベース攻撃に関する研究

Scan-based Side-channel Attacks against Cryptographic and Hash Function Integrated Circuits

2020年7月

早稲田大学 大学院基幹理工学研究科

情報理工・情報通信専攻 情報システム設計研究

於久 太祐

Daisuke OKU

目次

図目次

表目次

第 1 章

序論

1.1 本論文の背景と意義

今日では，インターネットの発達とともに様々なものが電子化され，盛んに様々な情報通信が行われている．買い物は実際に店舗に行かず，インターネット上にあるオンラインショップですることが増えている．オンラインショッピングではインターネット上で決済することができる．駅やコンビニをはじめとする店舗では切符や現金は電子マネーを使い電子決済することができる．このような便利な使い方ができる一方で，近年ではインターネットや情報ネットワークの発達とともに個人情報漏えいインシデントが増加している [1]．重要な個人情報を含む通信を行う時，第三者が盗聴，傍受または改ざんする可能性がある．インターネットの発達は生活を便利にする一方で，個人情報などの秘密情報を守る手段である情報セキュリティ技術の発達も必要となっている．

情報セキュリティ技術の中に暗号技術がある．送受信間で暗号技術を使い秘密情報を暗号化することで第三者による盗聴・傍受が困難になる．そのため，秘密情報を守る上で暗号化は重要な技術である．暗号化の主な方式には共通鍵暗号方式と公開鍵暗号方式がある．共通鍵暗号方式では暗号化と復号化する際に同じ鍵を使う．共通鍵暗号方式には暗号化する単位を 64bit や 128bit などのある単位毎に分けるブロック暗号と 1bit や 1Byte 毎に逐次的に暗号化するストリーム暗号の 2 種類がある．ブロック暗号には DES [2]，DES を 3 回行う Tripul DES，AES [3]，Camellia [4]，LED [5]，CLEFIA [6]，MISTY1 [7] がある．ストリーム暗号には Trivium [8]，MUGI [9]，線形帰還シフトレジスタ（LFSR: Linear Feedback Shift Register）に基づいたストリーム暗号 [10]，MICKEY [11]，Grain [12]，KCipher-2 [13] がある．公開鍵暗号方式は暗号化と復号化で異なる鍵を使う．公開する鍵である公開鍵，自身が持つ公開しない鍵である秘密鍵がある．メッセージを暗号化

するは受信者の公開鍵で暗号化し，受信者は自身の秘密鍵で復号化する．公開鍵暗号には RSA [14] と楕円曲線暗号（ECC: Elliptic Curve Cryptography）[15] がある．

また，情報セキュリティ技術の中には暗号技術の他にメッセージなどの情報を認証する技術もある．MAC（Message Authentication Code）と呼ばれるメッセージ認証符号を使い，正しいか認証する．MAC の 1 種としてハッシュ関数を使った HMAC（Hash-based MAC）[64] がある．HMAC はメッセージと秘密鍵をハッシュ関数に与え，メッセージと秘密鍵から生成されるハッシュ値を認証に使用する．HMAC は使うハッシュ関数により名前が異なる．ハッシュ関数に SHA-256 [65, 66] を使うものを HMAC-SHA-256 [67] と呼ぶ．HMAC-SHA-256 は IPsec や SSL/TLS などのセキュア通信で利用されている [68–73].

オンラインショッピングなどでの決済の処理はソフトウェア上で行われるため，ソフトウェア上での暗号技術の発達が重要である．電子マネー決済をする Suica やクレジットカードは一般に IC（Integrated Circuit）チップが備わっていることから IC カードと呼ばれている．IC カードは金融情報や個人情報などの秘密情報扱うこともあり，個人識別をするために暗号化する IC チップである暗号 LSI（Large Scale Integration）が搭載さている．そのため，改竄や情報漏洩，不正使用等を防ぐ必要がある．秘密情報を安全に保護できる IC カードが求められている．IC チップはハードウェアであり，ソフトウェアと同様にハードウェアでの暗号技術も必要となる．ハードウェアとソフトウェアで大きく異なる点は，ハードウェアは外部から観測することで様々な情報を読み取ることができる点である．そのため，暗号 LSI に対する情報セキュリティ技術はソフトウェアのときとは違った対策が必要となる．より安全な暗号 LSI を設計するためには攻撃手法に対する対策や防御手法が必要である．

暗号 LSI と HMAC を搭載した LSI への攻撃手法としてサイドチャネル攻撃がある．サイドチャネル攻撃は暗号処理装置を外部から物理的観測，計測することで秘密情報を取得する攻撃である．PC やオシロスコープなどの機器を使い，外部から制御・観測をすることで攻撃する．サイドチャネル攻撃にはタイミング攻撃 [16]，故障解析攻撃 [17, 18]，キャッシュ攻撃 [19]，そして，電力差分攻撃 [20–23]，電磁波攻撃 [24]，スキャンベース攻撃 [25–38] が報告されている．

スキャンベース攻撃は回路内部に搭載されているスキャンチェインを利用する攻撃手法である．スキャンチェインは回路の品質を保証するために，製造後に故障検出などのテストをするための仕組みである．スキャンチェインは LSI 内部のレジスタを構成するフリップフロップ（FF）をスキャンフリップフロップ（SFF）にすることで，スキャンチェインを直列に接続する．直列に接続することで外部か

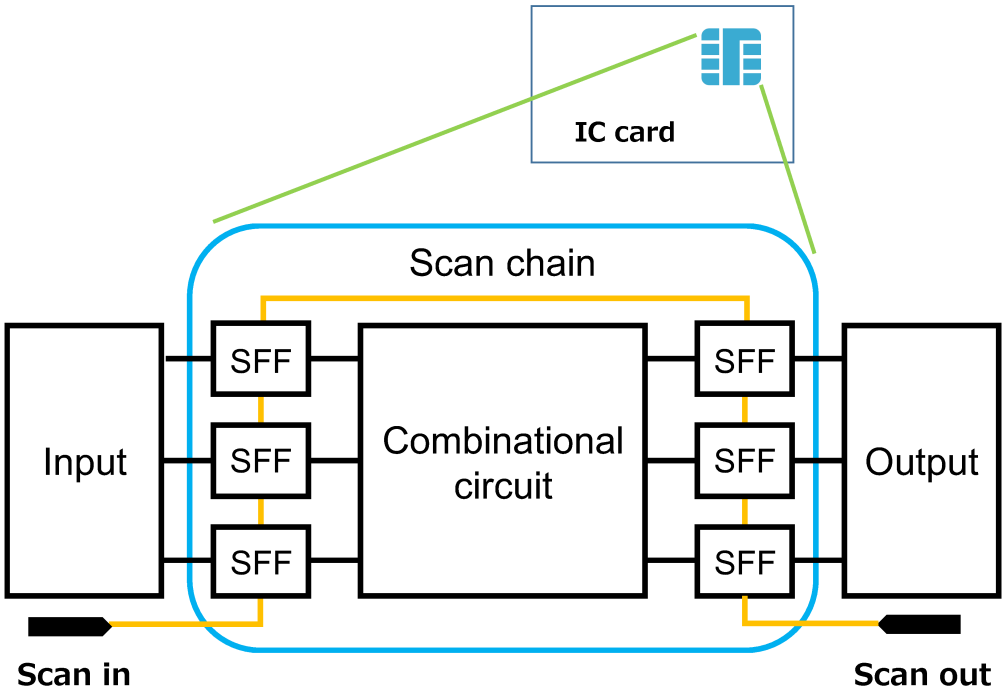


図 1.1: 回路内部にスキャンチェインが存在する IC カードの例.

らレジスタを直接制御，管理できる．制御性と観測性を高める技術であり，最も一般的なテスト容易化技術とされている [39]．図 1.1 に回路内部にスキャンチェインが存在する IC カードを示す．テスト用のスキャンチェインから得られるデータをスキャンデータと呼ぶ．スキャンチェインから容易にスキャンデータを取得できる仕組みを応用し暗号 LSI の秘密鍵を解読する．テストインターフェースとして JTAG と IEEE 1500 がよく使われている [40]．スキャンベース攻撃の概略を図 1.2 に示す．攻撃のシナリオとして，まず，攻撃者は暗号デバイスをなんらかの方法で取得し，攻撃手法に従いスキャンチェインからスキャンデータを取得する．次にスキャンデータから攻撃手法によりスキャンデータから秘密鍵を復元する．復元した秘密鍵使って同じ暗号アルゴリズムの回路を複製することで，不正利用する．文献 [25, 39–41] ではテストインターフェースが悪用される例を示しており，文献 [28, 35] ではスキャンチェインから秘密情報を取得した例を示している．したがって，テストインターフェースを悪用してスキャンチェインへアクセスできれば，攻撃者に秘密情報を取得される可能性がある．

スキャンベース攻撃の先行研究には，ブロック暗号 DES に対する攻撃手法 [25–27]，Triple DES に対する攻撃手法 [26]，AES に対する攻撃手法 [27–30]，Camellia に対する攻撃手法 [31]，LED に対する攻撃手法 [32]，公開鍵暗号 RSA に対する攻撃手法 [27, 33]，ECC に対する攻撃手法 [27, 34]，ストリーム暗号

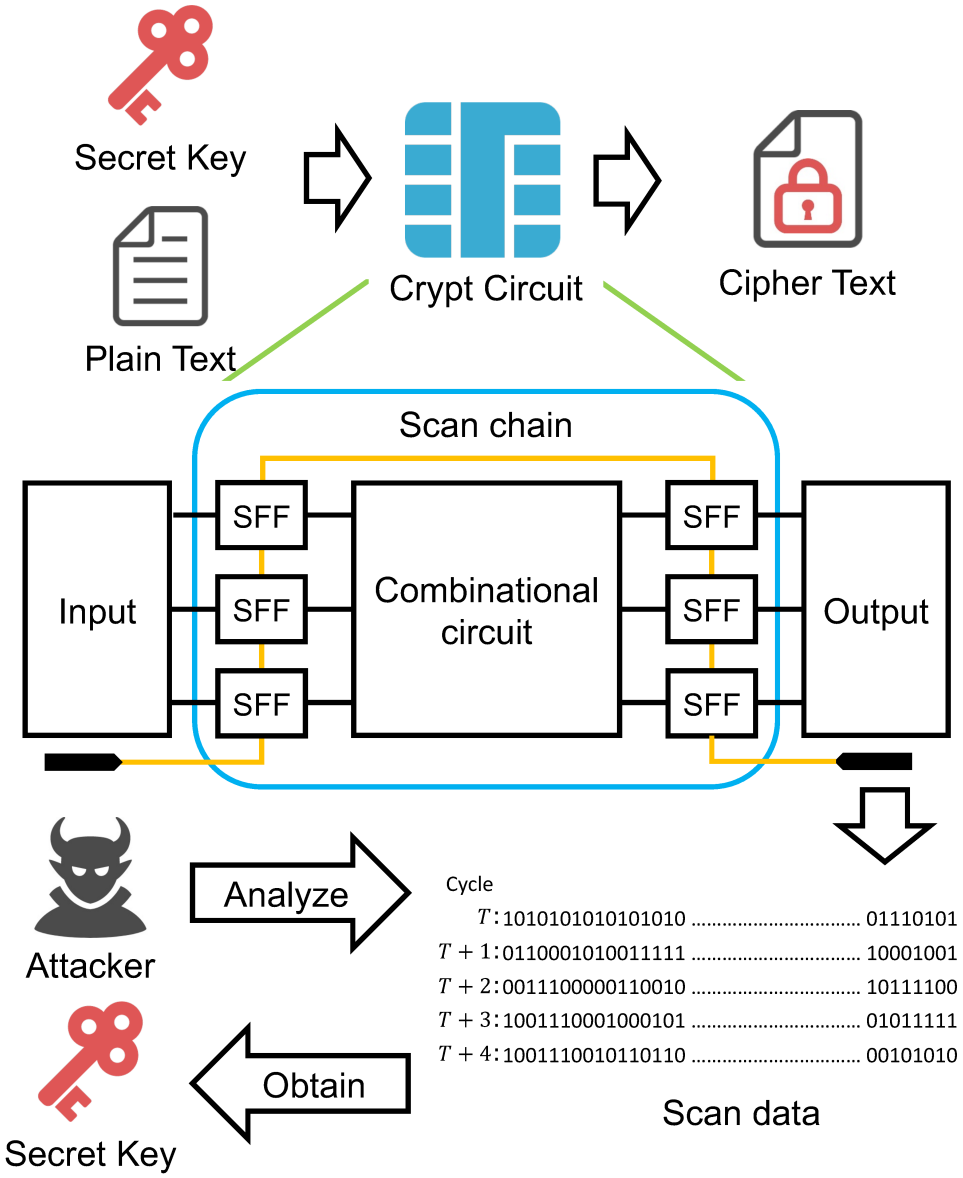


図 1.2: スキャンベース攻撃の概略.

Trivium に対する攻撃手法 [35, 36]，LFSR ベースのストリーム暗号に対する攻撃手法 [37, 38]．また，実機を使用しての暗号回路に対するスキャンベース攻撃の実装実験として，DES 回路を対象とした実験 [42]，Triple DES 回路を対象とした実験 [43]，AES 回路を対象とした実験 [44]，Camellia 回路を対象とした実験 [31]，LED 回路を対象とした実験 [45] が報告されている．スキャンチェインから得られるスキャンデータは秘密情報ではないので，スキャンチェインへのアクセスができる可能性がある．また，スキャンベース攻撃に関する既存研究ではスキャンチェインが残っていることを前提としている．

スキャンベース攻撃に対する防御手法として，提案されている防御手法は以下の

様に 2 種類に分類できる [46].

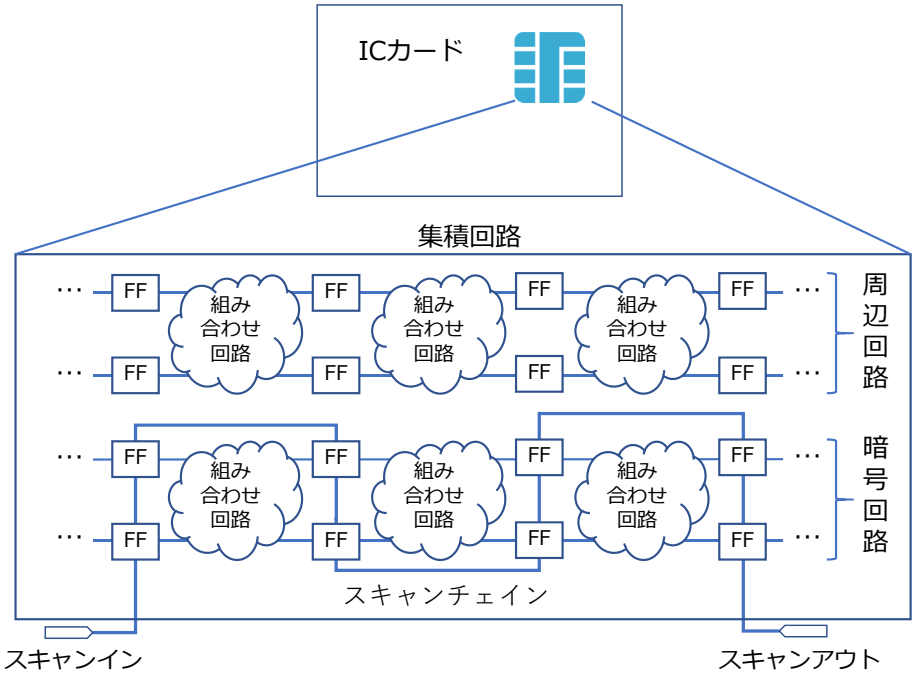
- スキャンチェインの構造を変化させる手法
  インバータや XOR ゲートなどの回路素子をスキャンチェイン上に挿入することでスキャンデータを難読化させる手法 [35, 47–50]，スキャンチェインを複数のサブチェインに分割し動的に変化させる手法 [51, 52]，スキャンチェインを制御する回路を追加する手法 [53–55].
- スキャンチェインの入出力に対してモジュールを追加する手法
  スキャン入力の前段階とスキャン出力の後段階にラッパーモジュール（暗号モジュール [56–59]，マスクモジュール [60]）を追加することで，スキャンチェインへのアクセスを制限する手法.

スキャンチェインの構造を変化させる手法はいづれの防御手法でも回路面積が増加し，テスト性が下がる可能性，スキャンチェインの長さが増えるため，テストにかかる時間が増加する可能性がある．文献 [35, 47, 48] の手法は攻撃手法が存在し [35, 48, 54]，文献 [49, 50] の手法はテスト性が下がるとされている [55]．スキャンチェインを複数のサブチェインに分割し動的に変化させる手法は攻撃が可能であると示されている [55]．なお，防御手法の実装例としてはテスト回路，ブロック暗号（DES，AES，SKINNY，PRESENT），公開鍵暗号（RSA）に限られている．本論文で取り上げる Trivium，HMAC-SHA-256，CLEFIA については，スキャンベース攻撃の可能性ならびにその防御の実装可能性も不明である．

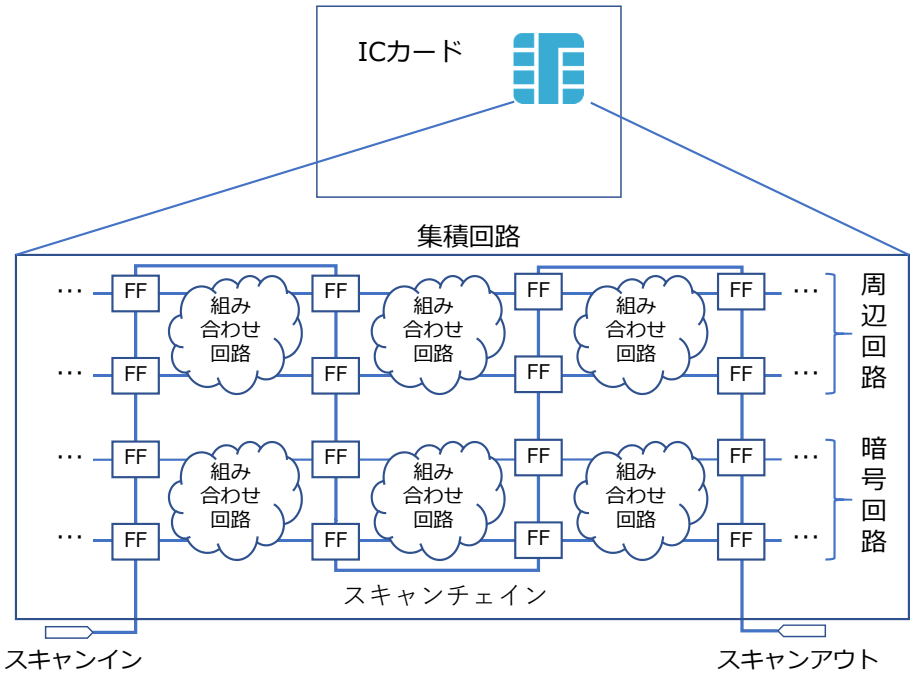
スキャンチェインを制御する回路を追加する手法とスキャンチェインの入出力に対してモジュールを追加する手法はスキャンチェインへの構造を変化させないためテストにかかる時間は変わらないとされている．しかし，防御手法のセキュリティは認証の強度に依存しているため，ラッパーモジュールが攻撃の対象となる可能性がある．また，ラッパーモジュールの品質を保証するためのテスト手法が必要である．さらに回路面積が増大し，認証によるテストにかかる時間の増加の可能性がある上記の議論により，全ての LSI に搭載されていると考えられない.

防御手法とは別にテスト後にスキャンチェインのピンを潰す，テストインターフェイスとスキャンチェインとの接続を切断する手法がある [61]．ただし，デバッグとメンテナンスの障害となる [28]，集束イオンビームやマイクロプローブを使用してテストインターフェイスを再び有効にする可能性がある [62, 63].

本論文では，秘密情報を安全に保護できる集積回路 （IC）を設計するために，攻撃を通して集積回路の脆弱性を調査・解明することを目的としている．先行研究として集積回路から秘密情報を取得するサイドチャネル攻撃が報告されている．本論文では特にスキャンベース攻撃に注目する．スキャンベース攻撃の先行研究



(a) 先行研究で攻撃可能な限定された回路構造.



(b) 提案手法で攻撃可能な回路構造.

図 1.3: 攻撃可能な回路構造の比較.

中には攻撃対象の回路構造を限定しているため，脆弱性を完全には指摘できないという問題点がある．攻撃対象としている回路構造を図 1.3a に示す．図 1.3a の回路構

表 1.1: 先行研究と本研究の位置づけ.

種類	アルゴリズム	構造依存あり	構造依存なし	実装実験
ブロック暗号	DES [2]	[25]	[26, 27]	[42]
	Triple DES		[26]	[43]
	AES [3]	[28]	[27, 29, 30]	[44]
	Camellia [4]		[31]	[31]
	LED [5]		[32]	[45]
	CLEFIA [6]		5 章	
ストリーム暗号	Trivium [8]	[35]	[36]	2 章
	LFSR [10]	[37]	[38]	
公開鍵暗号	RSA [14]		[27, 33]	
	ECC [15]		[27, 34]	
ハッシュ関数	PVG [74]		[75]	
	SHA-256 [65, 66]		3 章，4 章	

造はスキャンチェインに特定のレジスタ (FF) が接続された構造である．一般的に集積回路には暗号回路だけでなく，暗号回路を制御する回路等の複数の周辺回路が存在している．スキャンチェインに回路内の全てのレジスタ (FF) が接続された構造を図 1.3b に示す．また，多くの提案手法は計算機実験のみで検証しているため，実回路へ攻撃できるか不明であることが多い．

これらの問題を解決するために，各暗号方式のアルゴリズムに対して攻撃手法を提案することで実装される暗号方式の脆弱性を解明する．特に本論文では，ストリーム暗号として Trivium を対象としたスキャンベース攻撃実装実験の結果を示し，ハッシュ関数として HMCA-SHA-256 とブロック暗号 CLEFIA を対象としスキャンベース攻撃を提案した．

本論文での貢献点としてハードウェア特有の実装方法の脆弱性を調査するため，提案されている実装方法に着目したスキャンベース攻撃手法を提案していることがあげられる．前提条件を明確にすることで，攻撃の可能性を議論し，実装方法の脆弱性の解明に繋がる．また，回路構造をより柔軟にすることで，一般的な集積回路に近い回路を対象としている．さらに，計算機実験だけでなく提案手法を用いて実回路へ攻撃し，回路の実装方法の脆弱性を調査している．先行研究と本論文の位置づけを表 1.1 にまとめる．

1.2　本論文の概要

本論文では，実装されているストリーム暗号回路に対するスキャンベース攻撃の実験，ハッシュ回路とブロック暗号回路に対するスキャンベース攻撃手法を提案し，計算機評価実験の結果を示す．

以下に本論文の構成を示す．

第 2 章「ストリーム暗号に対するスキャンベース攻撃の実装実験」では，ストリーム暗号を実装した回路に対するスキャンベース攻撃の実装実験の結果を示す．本章で攻撃対象とするストリーム暗号は Trivium である．Trivium は暗号評価プロジェクト eSTREAM で推奨アルゴリズムに認定されており，回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．ブロック暗号に対する実装したスキャンベース攻撃実験は示されいるが，実装したストリーム暗号回路に対するスキャンベース攻撃実験は示されていない．幅広い暗号方式の脆弱性を確認し，実装方法の脆弱性を調査するために選定した．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．今実験で使用するハードウェアアーキテクチャについて説明する．さらに，サイドチャネル攻撃評価標準ボードを用いて，実装した Trivium 暗号回路に対してスキャンベース攻撃する実験の結果を示す．本実験で使用する Trivium 回路はハードウェア記述言語 Verilog-HDL を用いて実装する．本実験では提案されているスキャンベース攻撃手法を用いる．提案手法はスキャンチェイン上に Trivium 暗号回路以外のレジスタが存在しても攻撃が可能な手法である．評価ボード上のレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，スキャンデータを取得できることを確認した．また，得られたスキャンデータを提案手法を用いて解析した結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

第 3 章「連続動作するハッシュ回路に対するスキャンベース攻撃」では，連続動作するハッシュ回路に対するスキャンベース攻撃手法を提案する．連続動作するハッシュ回路として，メッセージ認証符号である HMAC-SHA-256 を生成する回路を対象とする．HMAC（Hash-based Message Authentication Code）は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．本章で攻撃対象

とするハッシュ回路は SHA-256 回路である．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．実アプリケーションを想定して実装した回路も提案されており，実装方法における脆弱性を調査するため選定した．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成する回路である．攻撃の前提条件を示し，連続動作する HMAC-SHA-256 生成回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

第 4 章「連続動作しないハッシュ回路に対するスキャンベース攻撃」では，連続動作しないハッシュ回路に対するスキャンベース攻撃手法を提案する．連続動作しないハッシュ回路として，メッセージ認証符号である HMAC-SHA-256 を生成する回路を対象とする．本章で攻撃対象とするハッシュ回路は SHA-256 である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成しない回路である．攻撃の前提条件を示し，連続動作しない HMAC-SHA-256 生成回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

第 5 章「ブロック暗号に対するスキャンベース攻撃」では，ブロック暗号に対するスキャンベース攻撃を提案する．本章で攻撃対象とするブロック暗号は CLEFIA である．CLEFIA は軽量暗号の国際標準規格 ISO/IEC 29192 に採択されているアルゴリズムであり，AES と互換性がある鍵長，ブロック数を持つ．アルゴリズ

ムをそのまま回路へ実装するだけではなく，より軽量に回路へ実装する手法も提案されており，実装方法における脆弱性を調査するため選定した．平文を暗号化するブロック長は 128 ビットであり，鍵長は 128 ビット，192 ビット，256 ビットの 3 種類ある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．攻撃の前提条件を示し，それぞれの CLEFIA 回路アーキテクチャに対するスキャンベース攻撃手法を提案する．提案手法は多数の入力メッセージを使った内部レジスタの特定，暗号化に使うラウンド鍵の特定の 2 ステップから構成されている．回路から得られるスキャンデータと CLEFIA 回路内のレジスタの対応関係を求め，秘密鍵を復元する．提案手法を使った計算機評価実験の結果，スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

第 6 章「結論」では，本論文全体の内容を総括し，今後の課題をまとめる．

第 2 章

ストリーム暗号に対するスキャンベース攻撃の実装実験

2.1 本章の概要

本章[*1]では，ストリーム暗号を実装した回路に対するスキャンベース攻撃の実験結果を示す．本章で攻撃対象とするストリーム暗号は Trivium である．Trivium は暗号評価プロジェクト eSTREAM で推奨アルゴリズムに認定されており，回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．ブロック暗号に対する実装したスキャンベース攻撃実験は示されいるが，実装したストリーム暗号回路に対するスキャンベース攻撃実験は示されていない．幅広い暗号方式の脆弱性を確認し，実装方法の脆弱性を調査するために選定した．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．今実験ではサイドチャネル攻撃評価標準ボードである SASEBO-GII [76] を用いて，実装した Trivium 回路に対してスキャンベース攻撃する実験の結果を示す．本実験で使用する Trivium 回路はハードウェア記述言語 Verilog-HDL を用いて実装する．スキャンベース攻撃手法として藤代らの手法 [36] を用いる．提案手法はスキャンチェイン上に Trivium 回路以外のレジスタが存在しても攻撃が可能な手法である．評価ボード上のレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，スキャンデータを取得できることを確認した．また，得られたスキャンデータを提案手法を用いて解析し

[*1] 本章は ⟨10⟩, ⟨17⟩ で発表した内容による．

た結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

以下に本章の構成を示す．

2.2 節「ストリーム暗号 Trivium」では，Trivium のアルゴリズムを説明し，今実験で使用するハードウエアアーキテクチャを説明する．Trivium 回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．

2.3 節「Trivium 回路に対するスキャンベース攻撃手法」では，本節では攻撃の前提条件を説明し，実装実験で用いる Trivium 回路に対するスキャンベース攻撃手法である藤代らの提案手法 [36] と Trivium 回路の内部状態を復元する方法を説明する．Trivium に対するスキャンベース攻撃の手法として Agrawal らの手法 [35] も存在する．どちらの手法もシミュレーション上では有効な攻撃手法であるが，実機を使っての有効性は確認されていない．Agrawal らの手法は暗号 LSI 中のスキャンチェイン上に Trivium 回路が使用するレジスタしか無いことを前提としているため，スキャンチェイン上に他のレジスタが存在する場合の暗号 LSI では用いることができないと考えられる．藤代らの提案した手法を使い，内部レジスタとスキャンデータの対応付けをした後，Trivium 回路の内部状態を復元する．

2.4 節「FPGA を使ったスキャンベース攻撃の実装実験」では，実装した Trivium 回路に対するスキャンベース攻撃した実験の結果を示す．シミュレーション上だけでなくハードウェア上に実装し攻撃する．本実験では FPGA 上に Trivium 回路をコンフィギュレーションした．対象アーキテクチャは 2.3 節で説明した Trivium 回路である．スキャンベース攻撃として藤代ら [36] の手法を用いた．SASEBO-GII 上の 1024 ビットのレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，暗号用 FPGA からスキャンデータを取得することができた．また，得られたスキャンデータをスキャンベース攻撃手法を用いて解析した結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

2.5 節「本章のまとめ」では，本章の内容をまとめる．

2.2 ストリーム暗号 Trivium

本節では Trivium のアルゴリズムと実験で使用するハードウエアアーキテクチャを説明する．

2.2.1 Trivium のアルゴリズム

本項では Trivium のアルゴリズム [8] を説明する．Trivium は Cannière らが考案した同期式ストリーム暗号である．暗号評価プロジェクト eSTREAM で推奨アルゴリズムに認定されており，回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．Trivium のハードウェア構造を図 2.1 に示す．四角形の部分が内部状態を示すレジスタであり円形に配置されている．内部シフトレジスタは合計で 288 ビットある．

ブロック暗号に対する実装したスキャンベース攻撃実験は示されいるが，実装したストリーム暗号回路に対するスキャンベース攻撃実験は示されていない．幅広い暗号方式の脆弱性を確認し，実装方法の脆弱性を調査するために選定した．

Trivium アルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズから構成されている．Trivium 暗号回路の概略を図 2.2 に示す．初期化フェーズで内部レジスタを初期値（IV）で初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．Trivium で使用する秘密鍵と IV は 80 ビットである．生成したキーストリームを用いて 2^{64} ビットまでの平文を暗号可能である．復号化する際は，暗号文とキーストリームを排他的論理和する．

初期化フェーズでは 80 ビットの秘密鍵と 80 ビットの IV を用いて 288 個の内部レジスタを初期化する．初期化フェーズのアルゴリズムを Algorithm1 に示す．Algorithm1 では内部レジスタの各ビットを $s_1, \ldots, s_{288}$，秘密鍵の各ビットを $K_1, \ldots, K_{80}$，IV の各ビットを $\mathrm{IV}_1, \ldots, \mathrm{IV}_{80}$ と表し，$\oplus$ は排他的論理和，$\cdot$ は論理積を表す．レジスタに秘密鍵と IV を入力後は内部レジスタの中の特定の 15 個を用いて 3 個のレジスタを更新し，内部レジスタをシフトして内部レジスタの更新を繰り返す．秘密鍵と IV を代入，for 文内は 1 クロックサイクルで実行される．全部を実行するためには $1 + 4 \times 288 = 1153$ クロックサイクル必要となる．

キーストリーム生成フェーズのアルゴリズムを Algorithm2 に示す．キーストリーム生成フェーズでは内部レジスタの中の特定の 15 ビットを用いて 3 個のレジ

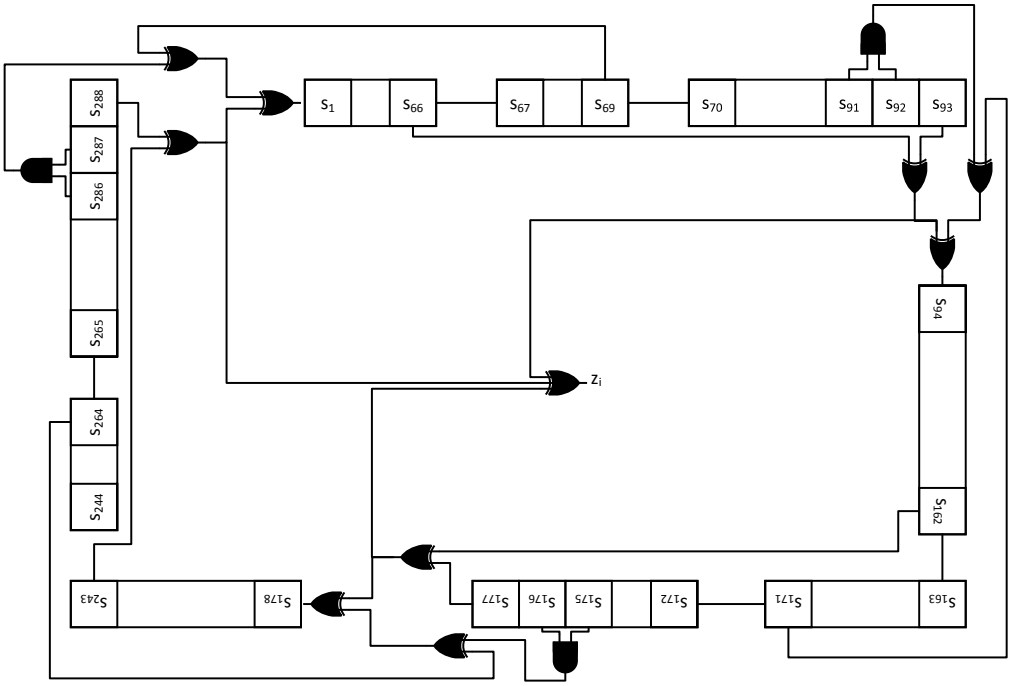


図 2.1: Trivium のハードウェア構造.

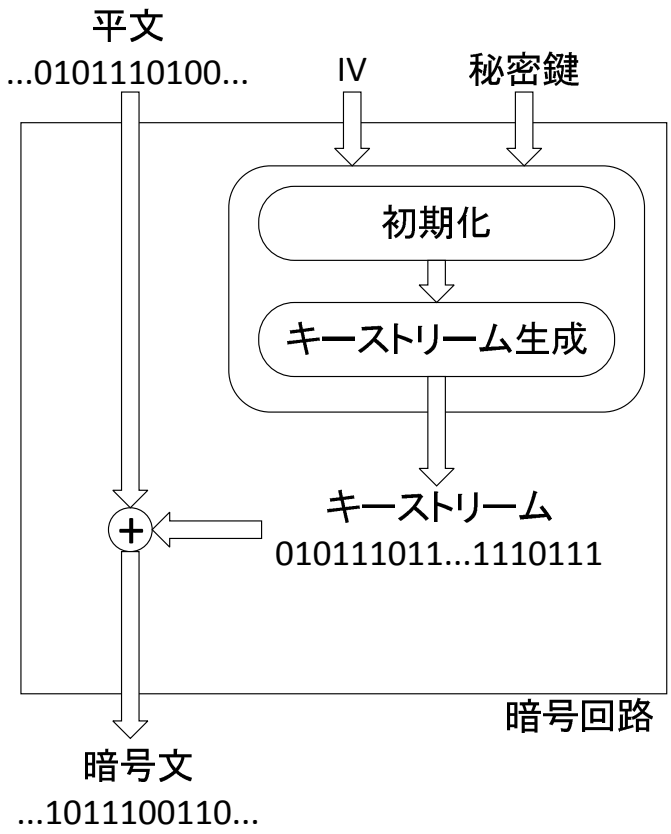


図 2.2: Trivium 暗号回路の概略図.

Algorithm 1 初期化

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, K_2, \ldots, K_{80}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (\mathrm{IV}_1, \mathrm{IV}_2, \ldots, \mathrm{IV}_{80}, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to 4×288 **do**
  $t_1 \leftarrow s_{66} \oplus s_{91} \cdot s_{92} \oplus s_{93} \oplus s_{171}$
  $t_2 \leftarrow s_{162} \oplus s_{175} \cdot s_{176} \oplus s_{177} \oplus s_{264}$
  $t_3 \leftarrow s_{243} \oplus s_{286} \cdot s_{287} \oplus s_{288} \oplus s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

Algorithm 2 キーストリーム生成

for $i = 1$ to N **do**
  $t_1 \leftarrow s_{66} \oplus s_{93}$
  $t_2 \leftarrow s_{162} \oplus s_{177}$
  $t_3 \leftarrow s_{243} \oplus s_{288}$
  $z_i \leftarrow t_1 \oplus t_2 \oplus t_3$
  $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
  $t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}$
  $t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for

スタを更新し，1 ビットずつキーストリーム z_i を生成する．平文 N $(\leq 2^{64})$ ビットの数だけ内部状態レジスタを更新し，キーストリームを生成する．Algorithm2 は N クロックサイクルで実行される．1 クロックサイクルで 1 ビットのキーストリーム z_i が生成される．生成したキーストリームと平文の各ビットを排他的論理和することで暗号化する．

図 2.3: 公開されている Trivium コード [8] の結果.

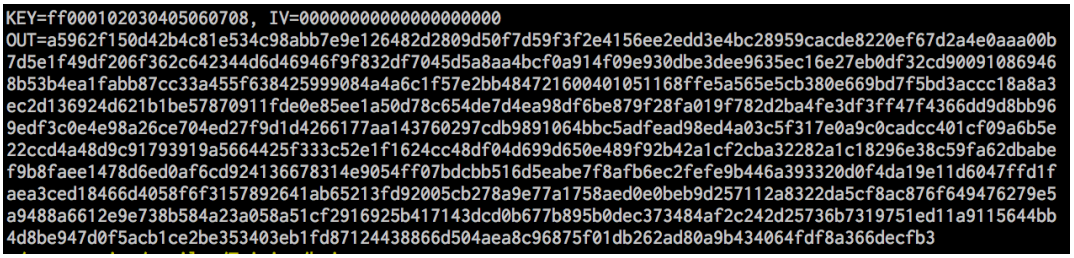


図 2.4: 作成したソースコードのキーストリーム出力結果.

2.2.2 ハードウエアアーキテクチャ

本項では Trivium 回路のハードウェアアーキテクチャを説明する．本実験で使用する Trivium 回路をハードウェア記述言語 Verilog-HDL を用いて実装した．動作の検証シミュレーションは Icarus Verilog [77] と GTKWave [78] を用いた．暗号化した値の検証には公開されている C 言語ソースコード [8] を用いて行った．シミュレーションの出力と公開されてる C 言語のソースコードの出力は 512Byte のキーストリームである．暗号化するための秘密鍵（KEY）と初期値（IV）の値は以下のようにした．

$$\mathrm{KEY} = 0\mathrm{xFF000102030405060708}$$
$$\mathrm{IV} = 0\mathrm{x00000000000000000000}$$

公開されているソースコード [8] の結果を図 2.3 に示す．図 2.3 の出力値 stream[0..511] は正しい順番ではないのでエンディアン変換をする必要がある．作成した Trivium 回路のキーストリーム出力結果を図 2.4 に示す．作成した Trivium 回路のシミュレーション結果を図 2.5，2.6 に示す．図 2.5 は暗号化の開始時のシミュレーションであり，図 2.6 は暗号化の終了時のシミュレーションである．図 2.5，2.6 の各信号の役割を表 2.1 に示す．

エンディアン変換した図 2.3 の出力値 stream[0..511] と図 2.4 の出力値 OUT の値を比較すると同じであることがわかる．図 2.5 では CLK の立ち上がり時に

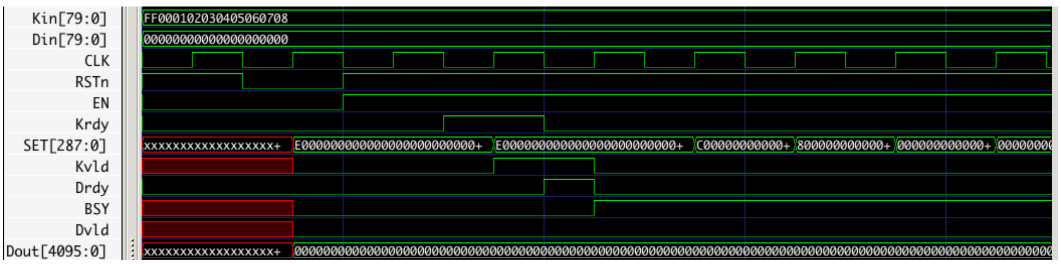


図 2.5: 作成したソースコードのシミュレーション結果（開始時）.

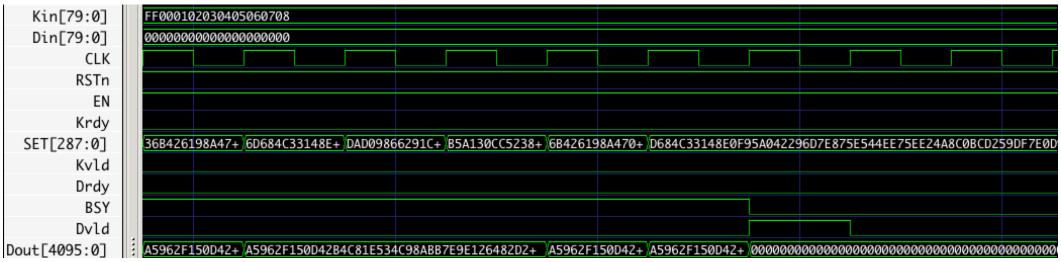


図 2.6: 作成したソースコードのシミュレーション結果（終了時）.

表 2.1: 図 2.5 と図 2.6 の各信号の役割.

信号名	役割	説明
Kin	入力	KEY（秘密鍵）の値
Din	入力	IV の値
CLK	入力	クロック信号
RSTn	入力	リセット信号．レジスタを初期化する.
EN	入力	イネーブル信号．1 の時，回路が動作する.
Krdy	入力	1 の時，KEY 値と IV 値がレジスタにセットする.
SET	レジスタ	288 個のレジスタ
Kvld	出力	1 の時，KEY 値と IV 値がセットされたことを示す.
Drdy	入力	1 の時，暗号化を開始する.
BSY	出力	1 の時，暗号化中を示す.
Dvld	出力	1 の時，暗号化終了を示す.
Dout	出力	キーストリーム値

Algorithm1 の for 文内を 1 回実行している．図 2.6 では CLK の立ち上がり時に Algorithm2 の for 文内を 1 回実行している.

2.3　Trivium 回路に対するスキャンベース攻撃手法

本節では攻撃の前提条件を説明し，実装実験で用いる Trivium 回路に対するスキャンベース攻撃手法である藤代らの提案手法 [36] と Trivium 回路の内部状態を復元する方法を説明する．Trivium に対するスキャンベース攻撃の手法として Agrawal らの手法 [35] も存在する．どちらの手法もシミュレーション上では有効な攻撃手法であるが，実機を使っての有効性は確認されていない．Agrawal らの手法は暗号 LSI 中のスキャンチェイン上に Trivium 回路が使用するレジスタしか無いことを前提としているため，スキャンチェイン上に他のレジスタが存在する場合の暗号 LSI では用いることができないと考えられる．藤代らの手法は，レジスタ値の入力・動作サイクル数による変化がそのレジスタ固有の値になることを利用しており，暗号回路以外のレジスタがスキャンチェイン上に含まれる場合も攻撃が可能である．

2.3.1　攻撃の前提条件

本項では藤代らが提案した攻撃手法の前提条件 [36] を紹介する．藤代らの攻撃の前提条件を以下に示す．

Trivium の攻撃で攻撃者がわかること

- Trivium 暗号回路が出力した暗号文
- Trivium 暗号回路が暗号文を出力した直後のスキャンデータ
- スキャンチェインはフルスキャン設計で，反転・動的に変化しないこと
- スキャンデータを圧縮していないこと

攻撃者ができること

- Trivium 暗号回路に任意の秘密鍵と IV を入力すること
- 任意のタイミングで Trivium 暗号回路のスキャンチェインにアクセスし，スキャンデータを取得すること

攻撃者がわからないこと

- キーストリームの値
- スキャンチェインに接続されているレジスタの接続順と数，種類

一般的な LSI ではスキャンチェイン上に複数の回路のレジスタが含まれている

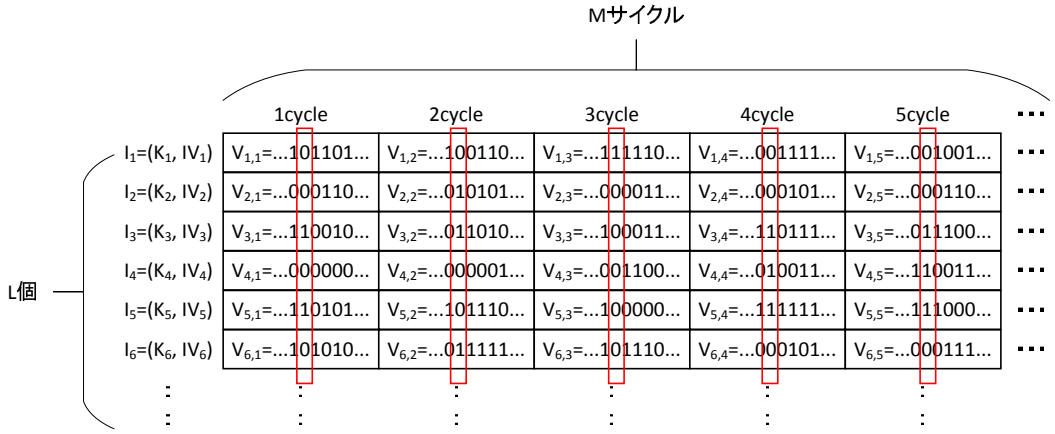


図 2.7: スキャンシグネチャ.

ことがあるので，スキャンチェインに含まれるレジスタの構成・接続順情報を得る必要はない．任意の秘密鍵と IV を用いて，任意のタイミングでスキャンチェインにアクセスすることでスキャンデータを得る．

2.3.2 藤代らの攻撃手法

本項では藤代らの攻撃手法 [36] を説明する．攻撃手法ではレジスタ値の入力・動作サイクル数による変化がそのレジスタ固有の値になることを利用している．攻撃者は Trivium 暗号回路に任意の秘密鍵と IV を入力するので秘密鍵と IV の入力ペアを用意する．各入力ペアに対して Trivium 暗号回路を数サイクルだけ動作させ，クロックサイクル毎に内部レジスタの値を取得し並べる．並べた状態を図 2.7 に示す．図 2.7 では縦に入力ペア，横に動作サイクル数を表している．各クロックサイクルで k ビット目はスキャンチェイン上の内部レジスタのある 1 ビットの状態の変化を表している．図 2.7 で枠で囲った値はあるレジスタの値の変化を表している．入力ペアの数，サイクル数を大きく取った場合，枠で囲った値はある 1 つの内部レジスタの固有の値になる．本論文では枠で囲った値をスキャンシグネチャと呼ぶ．藤代らの手法はスキャンシグネチャを用いてスキャンデータから対応するレジスタを見つけ，内部状態を復元することで秘密鍵を復元する．

藤代らの提案手法のアルゴリズムを以下に示す．

1. 入力ペアのパターンを L 個 $(I_1, \ldots, I_L)$ 用意する
2. 各入力ペア $I_1, \ldots, I_L$ の Trivium の内部状態値をシミュレーションで M サイクル分求める
3. 2 で求めたデータに対して Trivium 暗号回路のある 1 つの内部レジスタ

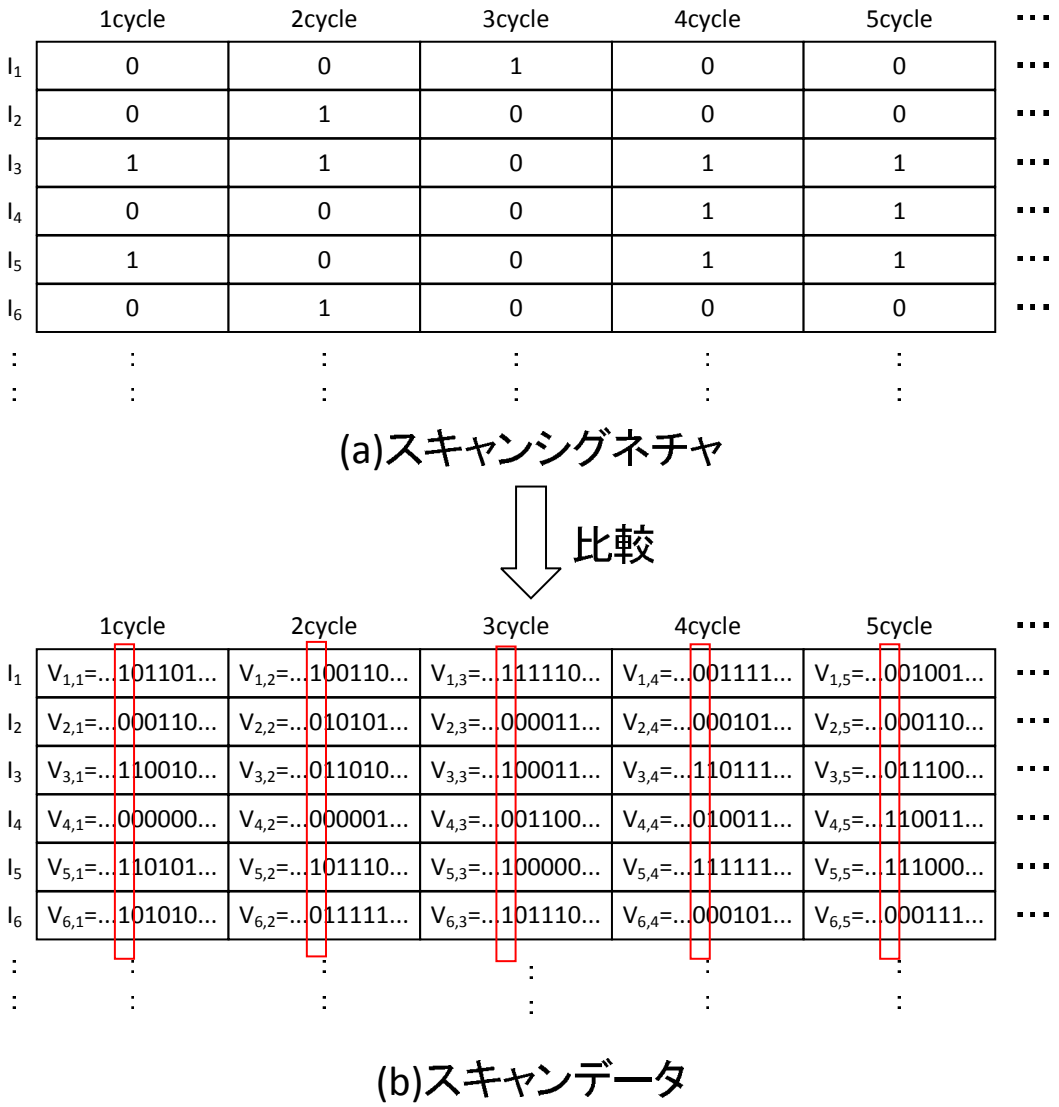


図 2.8: スキャンシグネチャとスキャンデータの比較.

$s_k (1 \leq k \leq 288)$ のスキャンシグネチャを E_{s_k} とする．E_{s_k} のスキャンシグネチャを図 2.8 (a) に示す.

4. 各入力ペア $I_1, \ldots, I_L$ を実際の Trivium 暗号回路に入力し，それぞれ M サイクル分のスキャンデータを取得する．入力ペア I_i，サイクル数 j の時に取得したスキャンデータを $V_{i,j}$ とする.
5. スキャンデータ $V_{1,1}, \ldots, V_{L,1}$ を縦に並べたものを $S_1 = (V_{1,1}, \ldots, V_{L,1})^t$ とする．M サイクル分 $S_1, \ldots, S_M$ を横に並べ，スキャンデータとする．M サイクル分のスキャンデータを図 2.8 (b) に示す．スキャンシグネチャ E_{s_k} をスキャンデータ $S_1, \ldots, S_M$ の何列目に存在するか探索する.
6. 5 で E_{s_k} がスキャンデータ $S_1, \ldots, S_M$ の p 列目のみ存在する場合，内部レ

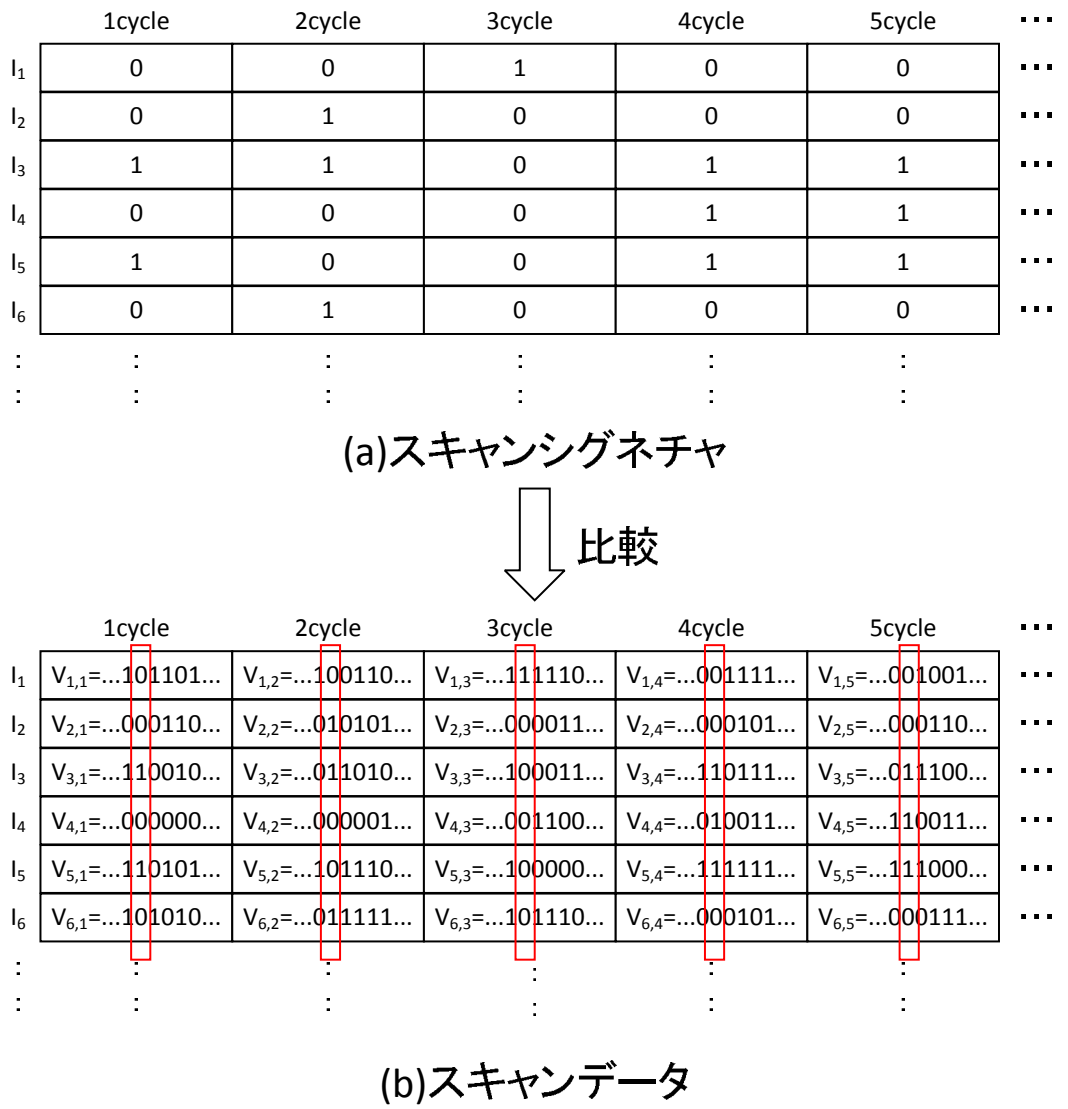


図 2.9: スキャンシグネチャとスキャンデータの比較（発見時）.

ジスタ s_k のビット位置がスキャンデータの p ビット目であるとわかる．スキャンデータ中からビット位置が定まる様子を図 2.9 に示す.

上記アルゴリズムで，L と M を大きく取ると p の値は一意に定まるので，スキャンデータ中のビット位置と内部レジスタの対応が一意に定まる．スキャンデータのビット対応が求まれば，暗号回路が暗号文を出力した直後のスキャンデータから Trivium の内部レジスタの各値（内部状態）を求められる．内部状態を復元できればレジスタを初期化した時の秘密鍵と IV の値が求められる．キーストリームを復元でき，暗号文から平文が復元できる.

2.3.3 内部状態復元

本項では内部状態復元方法を説明する．2.3.1 項で説明した前提条件より，攻撃者は暗号文と暗号回路が暗号文を出力した直後のスキャンデータがわかる．そのため，2.3.2 項で説明した攻撃手法で内部レジスタの対応付けができた後，内部状態が復元できれば暗号文とスキャンデータから平文を求められる．本節では説明のため，暗号文を出力した直後の Trivium の内部レジスタ 288 個の全ての値がわかると仮定する．時刻 T の内部レジスタを s^T とする．Algorithm2 より，時刻 T と時刻 $T-1$ の関係を以下に示す．仮定から $s_1^T, \ldots, s_{288}^T$ は既知である．

$$s_1^{T-1} = s_2^T,\ s_2^{T-1} = s_3^T,\ \ldots, s_{92}^{T-1} = s_{93}^T \tag{2.1}$$
$$s_{94}^{T-1} = s_{95}^T,\ s_{95}^{T-1} = s_{69}^T\ \ \ldots, s_{176}^{T-1} = s_{177}^T \tag{2.2}$$
$$s_{178}^{T-1} = s_{179}^T, s_{179}^{T-1} = s_{180}^T, \ldots, s_{287}^{T-1} = s_{288}^T \tag{2.3}$$

時刻 $T-1$ の内部レジスタの値は式 (2.1)，(2.2)，(2.3) より $s_{93}^{T-1}, s_{177}^{T-1}, s_{288}^{T-1}$ を除き，求めることができる．時刻 $T-1$ は Algorithm2 より

$$t_1 \leftarrow s_{66}^{T-1} \oplus s_{93}^{T-1} \oplus s_{91}^{T-1} \cdot s_{92}^{T-1} \oplus s_{171}^{T-1} \tag{2.4}$$
$$(s_{94}^T, s_{95}^T, \ldots, s_{177}^T) \leftarrow (t_1, s_{94}^{T-1} \ldots, s_{176}^{T-1}) \tag{2.5}$$

である．式 (2.4) は式 (2.5) より，

$$s_{94}^T = s_{67}^T \oplus s_{93}^{T-1} \oplus s_{92}^T \cdot s_{93}^T \oplus s_{172}^T \tag{2.6}$$

と表せる．式 (2.6) を s_{93}^{T-1} について解くと，

$$s_{93}^{T-1} = s_{67}^T \oplus s_{94}^T \oplus s_{92}^T \cdot s_{93}^T \oplus s_{172}^T \tag{2.7}$$

となる．同様に s_{177}^{T-1}，s_{288}^{T-1} も以下のようになる．

$$s_{177}^{T-1} = s_{178}^T \oplus s_{163}^T \oplus s_{176}^T \cdot s_{177}^T \oplus s_{265}^T \tag{2.8}$$
$$s_{288}^{T-1} = s_1^T \oplus s_{244}^T \oplus s_{287}^T \cdot s_{288}^T \oplus s_{70}^T \tag{2.9}$$

前提条件から暗号文が出力された直後の内部状態は取得できるので過去の状態が式 (2.7)，(2.8)，(2.9) から求められる．内部状態がわかれば，Algorithm2 よりキーストリームが復元できる．得られたキーストリームから暗号文と排他的論理和することで平文が復元できる．

2.4 FPGA を使ったスキャンベース攻撃の実装実験

実装した Trivium 回路に対するスキャンベース攻撃した実験の結果を示す.

2.4.1 実験環境

本項では実装実験の環境を示す．実験環境は以下の通りである.

- FPGA 評価ボード:SASEBO-GII [76]
- 暗号・制御回路:SASEBO-GII 用サンプルソースコード [76] + 自作 Trivium ソースコード + 改良した SASEBO Checker AES
- FPGA 開発環境:Xilinx ISE Foundation 14.7+ChipScope Pro 14.7 [79]
- ホスト PC: Panasonic Let's note CF-SX3,
  CPU: Intel Core i5–4200U（1.6 Hz），メモリ: 4GB

SASEBO-GII に Trivium 回路をコンフィギュレーションする．SASEBO-GII は産業技術総合研究所が開発したサイドチャネル攻撃用標準評価ボードである．SASEBO-GII を図 2.10 を示す．SASEBO-GII には暗号回路用 FPGA と制御用 FPGA の 2 つがあり，各 FPGA に JTAG ポートと SPI-ROM が配置されている．FPGA にプログラムを書き込むには JTAG ポート経由でコンフィギュレーションするか，SPI-ROM にプログラムをダウンロードしてコンフィギュレーションする．SASEBO-GII のブロック図を図 2.11 に示す．SASEBO-GII は外部に接続したホスト PC から USB インターフェースを通して制御する.

SASEBO-GII の暗号用 FPGA にコンフィギュレーションする Trivium 回路は 2.2.2 節で説明した自作のソースコードを用いる．暗号回路の制御には “SASEBO-GII クイックスタートガイド バイナリ&ソースコード” [76] で提供されているサンプルソースコードを用いる．サンプルソースコードには制御用 FPGA にコンフィギュレーションする制御回路や，ホスト PC から SASEBO-GII を制御するためのソースコードが含まれている．SASEBO-GII に搭載された暗号回路用 FPGA には FPGA 開発環境 Xilinx ISE Foundation を用いて回路をコンフィギュレーションする.

ホスト PC から暗号用 FPGA を制御するために “SASEBO-GII クイックスタートガイド バイナリ&ソースコード” に含まれている SASEBO Checker AES を用いる．SASEBO Checker AES の GUI は C#で記述されている．SASEBO-GII は本来はホスト PC から AES 暗号回路を制御するために使用するが，ソースコー

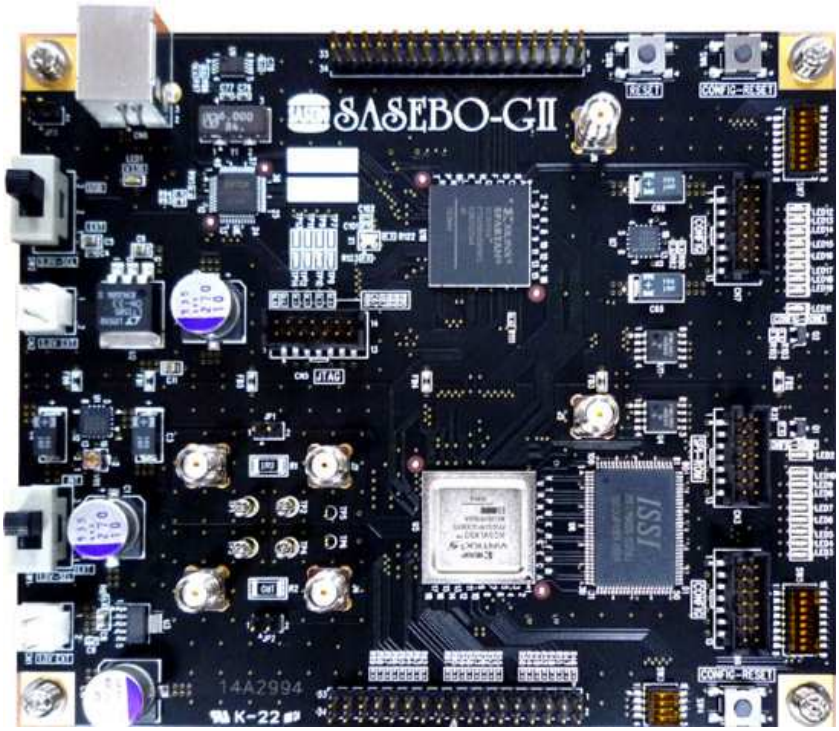


図 2.10: SASEBO-GII [76].

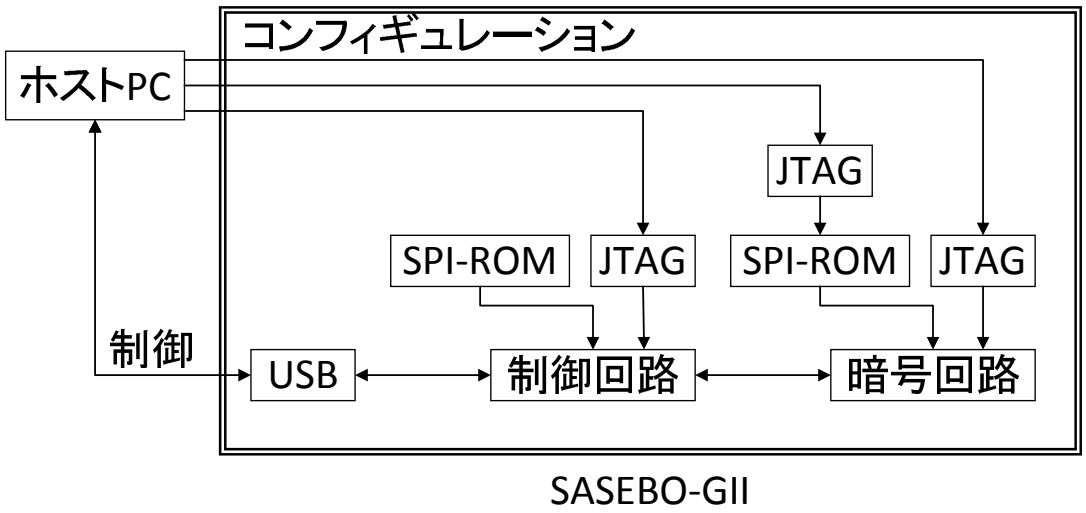


図 2.11: SASEBO-GII のブロック図.

ドを改変することで Trivium 回路用に変更できる．本実験では SASEBO Checker AES のコードを改良し Trivium 回路を制御する．ホスト PC と制御用 FPGA を USB 接続し，制御回路から暗号回路を操作する．改良した SASEBO Checker AES の GUI を図 2.12 に示す．図 2.12 の KEY には任意の秘密鍵を入力し，IV には予め作成したファイルから数値を読み込む．これにより任意の秘密鍵と IV を

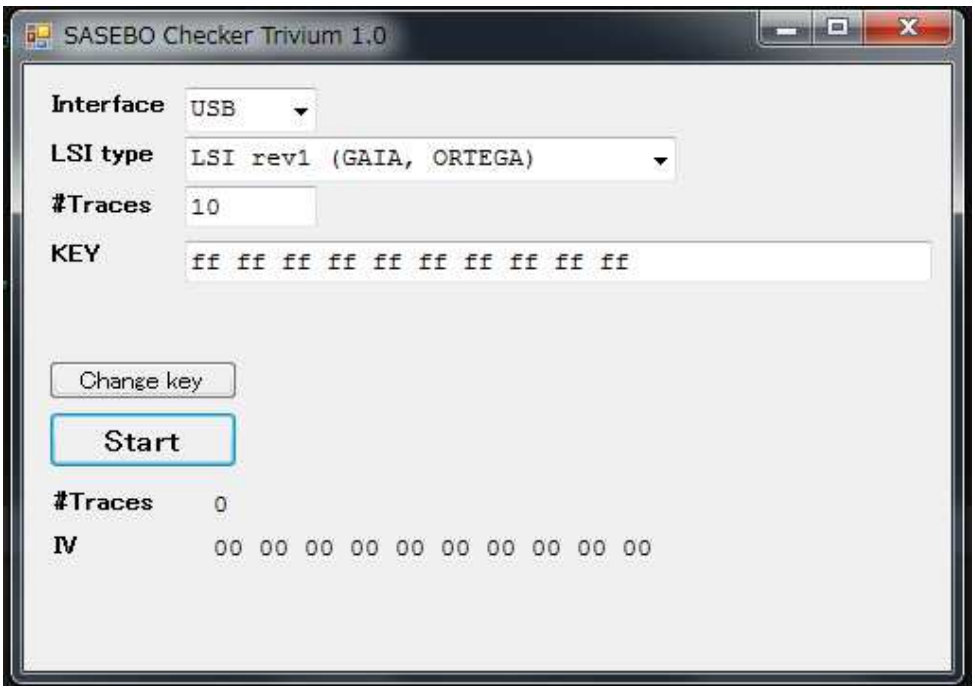


図 2.12: 改良した SASEBO ChipScope AES の GUI.

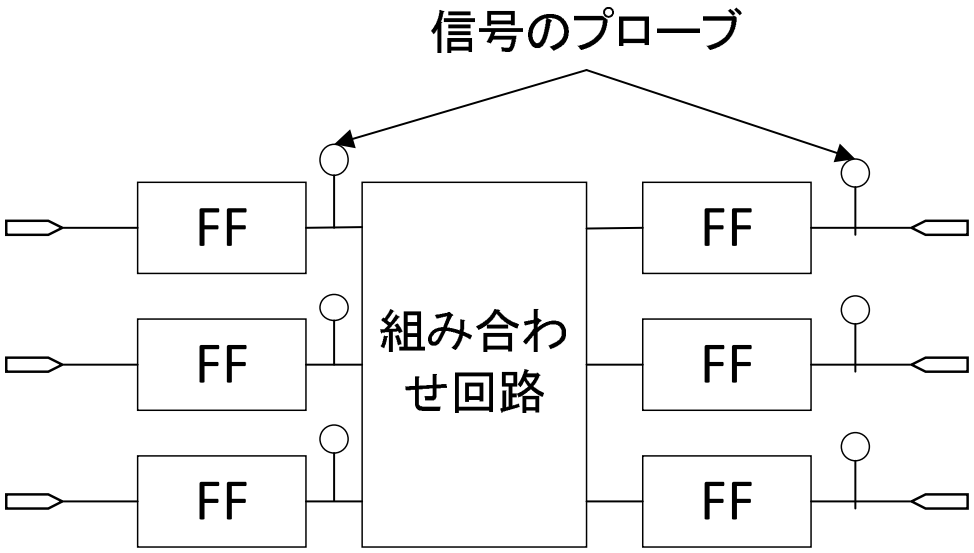


図 2.13: ChipScope Pro による信号観測.

回路へ送信することができる．#Traces の数だけ秘密鍵と IV のペアを回路へ送信する事ができる．

ホスト PC から回路を制御し任意の秘密鍵と IV を送信する．ChipScope Pro で観察回路を挿入した回路から ChipScope Analyzer を利用してスキャンデータを取得し，取得したスキャンデータを藤代らの攻撃 [36] により解析し攻撃する．

暗号処理中のテスト用スキャンデータを取得するためにロジックアナライザ ChipScope Pro [79] を用いる．SASEBO-GII にはスキャンチェインがないためスキャンチェインの代わりとして使用する．ChipScope Pro は FPGA にコンフィギュレーションした回路をデバックする際に使用するツールである．FPGA が動作している間の内部信号を記憶するメモリと制御回路を内部に構築する．JTAG ケーブルを経由してメモリの内容を ChipScope Analyzer で表示し観測できる．ChipScope Pro を用いたスキャンデータ取得の様子を図 2.13 に示す．ChipScope Pro は FPGA 内部に信号を観測するためのプローブを設置し，プローブの出力から内部信号の値を読み取る．コンフィギュレーションした後でもプローブを設置する内部信号を自由に変更できる．一度に観測できる内部信号のデータサイズは内部信号を記録するメモリ容量が上限となる．メモリ容量は観測する内部信号のビット数 × 観測するサイクル数となる．組み合わせ回路の出力値も観測できるため，スキャンパステストより容易に内部状態の読み出しができる．

2.4.2 実験手順

本項では実験手順を示す．前提条件としては 2.3.1 項で紹介した前提条件を用いる．実験手順は以下の通りである．

1. ISE Foundation 上でサンプルソースコードを論理合成する．論理合成後のネットリストに内部信号観察回路を挿入する．内部信号観察回路を挿入後の Trivium 回路のネットリストから bit ファイルを生成し，暗号用 FPGA にコンフィギュレーションする．
2. ホスト PC から改良した SASEBO Checker AES の GUI を操作し，暗号用 FPGA の Trivium 回路に秘密鍵と IV を入力し暗号処理させる．秘密鍵と IV は複数ペア用意しする．同時に，GhipScope Analyzer を用いて Trivium 暗号処理中の内部信号（スキャンデータ）を取得する．今実験では SASEBO-GII 上の内，Trivium 回路の内部レジスタを含む 1024bit を観測する．Trivium のアルゴリズムの初期フェーズの 1 サイクル前に観測を開始する．観測開始から 1024 サイクル分を観測する．
3. 取得したスキャンデータと計算機上のシミュレーションで生成したスキャンシグネチャを用いて藤代らの手法で攻撃し，内部レジスタの特定をする．藤代らの手法は python を用いてプログラムした．

今実験では SASEBO-GII からスキャンデータを取得し，藤代らの攻撃手法を用いて内部レジスタの対応付をすることを目的とする．藤代らの手法を実現するプ

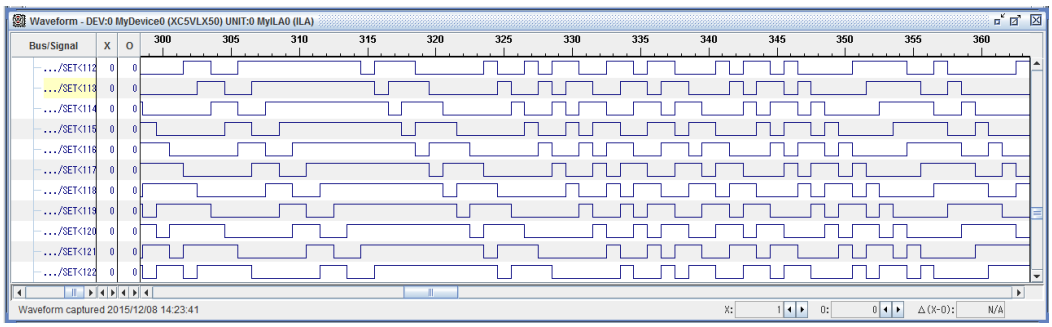


図 2.14: ChipScope Analyzer によるレジスタ信号観測.

ログラムは python を用いて実装した.

2.4.3 実験結果

本項では Trivium 回路を実装した SASEBO-GII にスキャンデータ攻撃した実験結果を示す．図 2.12 の GUI を利用して任意の秘密鍵と IV を SASEBO-GII に入力した．本実験では秘密鍵と IV のペアを 5 組用意し，Trivium 暗号回路内を含む SASEBO-GII 上の内部レジスタ 1024bit を 1024 サイクル分観測した．用意した秘密鍵と IV のペアを表 2.2 に示す．ChipScope Analyzer では観測を開始するトリガーと観測するサイクル数の設定ができ，トリガー発動時から設定するサイクル数分だけ内部信号観測ができる．ChipScope Analyzer による内部信号観測を図 2.14 に示す．図 2.14 では各レジスタの値が変動しているのがわかる．5 組のペアに対して藤代らの攻撃手法を用いた結果，内部レジスタを特定することに成功した．暗号回路内に ChipScope Pro の内部観察回路が備わっている場合，藤代らの手法はシミュレーション上だけではなく実機に対しても攻撃可能な手段であるとわかった．内部レジスタの特定にはペアごとにサイクル数が異なる．攻撃成功の最小サイクル数をまとめた表を表 2.3 に示す．今実験では表 2.1 の Drdy が 1 になった時から観測を開始した．表 2.3 は秘密鍵と IV の 1 組のペアのみで攻撃をした際，内部レジスタの対応付けに必要となる最小サイクル数を表している．1 組のペアのみである場合，スキャンシグネチャは 1bit となる．サイクル数によるスキャンシグネチャの遷移が一致する内部レジスタを探索することで 1bit のスキャンシグネチャでも内部レジスタの対応付けが可能となる．攻撃に複数ペアを使った場合，必要となるサイクル数はペアの中で最小サイクルになる．表 2.3 では上 2 組のペアを攻撃に使った場合，148 サイクルで攻撃が成功する．

解析にかかる時間は内部レジスタの特定に要するサイクル数がペアによって異なるので差がある．しかし，回路の観測開始から内部レジスタの対応付けまで 30

表 2.2: 秘密鍵と IV のペア.

ペア番号	秘密鍵	IV
P. 1	00000000000000000000	00000000000000000000
P. 2	06070809000000000000	4089D544000000000000
P. 3	FFFFFFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFFFFFF
P. 4	0102030405060708090A	00000000000000000000
P. 5	199E1B8344C2AD75C5AF	0783977B8CF6CB7C4CC3

表 2.3: 表 2.3 に示した各ペア番号と最小サイクル数.

ペア番号	最小サイクル数
P. 1	246
P. 2	148
P. 3	110
P. 4	180
P. 5	111

秒から 70 秒の間で行うことができる．実行時間の観点から見ても有効な攻撃手法であることがわかる．

表 2.2 と表 2.3 より鍵と IV の組み合わせによって最小サイクル数が変わる．Trivium は内部レジスタに秘密鍵と IV を入力する際，内部レジスタは以下のようになる．

$$
\begin{aligned}
(s_1, \ldots, s_{93}) &\leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0) \\
(s_{94}, \ldots, s_{177}) &\leftarrow (\mathrm{IV}_1, \ldots, \mathrm{IV}_{80}, 0, \ldots, 0) \\
(s_{178}, \ldots, s_{288}) &\leftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned}
$$

Trivium のアルゴリズムより，内部レジスタを 1 クロック毎に 1bit シフトする．シフトすることで初期レジスタで 0 の部分が変化をする．変化することでスキャンシグネチャが差別化され内部レジスタの対応付けができる．初期レジスタが 0 である部分を変化させるサイクル数に差があるため秘密鍵と IV によってサイクル数の差が生じる．

2.5 本章のまとめ

本章では，ストリーム暗号を実装した回路に対するスキャンベース攻撃の実装実験の結果を示した．本章で攻撃対象とするストリーム暗号は Trivium である．Trivium は暗号評価プロジェクト eSTREAM で推奨アルゴリズムに認定されており，回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．ブロック暗号に対する実装したスキャンベース攻撃実験は示されいるが，実装したストリーム暗号回路に対するスキャンベース攻撃実験は示されていない．幅広い暗号方式の脆弱性を確認し，実装方法の脆弱性を調査するために選定した．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．今実験で使用するハードウェアアーキテクチャについて説明した．さらに，サイドチャネル攻撃評価標準ボードである SASEBO-GII [76] を用いて，実装した Trivium 回路に対してスキャンベース攻撃する実験の結果をした．本実験で使用する Trivium 回路はハードウェア記述言語 Verilog-HDL を用いて実装した．スキャンベース攻撃手法として藤代らの手法 [36] を用いた．提案手法はスキャンチェイン上に Trivium 回路以外のレジスタが存在しても攻撃が可能な手法である．評価ボード上のレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，スキャンデータを取得できることを確認した．また，得られたスキャンデータを提案手法を用いて解析した結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

以下に本章の構成を示す．

2.2 節「ストリーム暗号 Trivium」では，Trivium のアルゴリズムを説明し，今実験で使用するハードウエアアーキテクチャを説明した．Trivium 回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．

2.3 節「Trivium 回路に対するスキャンベース攻撃手法」では，本節では攻撃の前提条件を説明し，実装実験で用いる Trivium 回路に対するスキャンベース攻撃手法である藤代らの提案手法 [36] と Trivium 回路の内部状態を復元する方法

を説明した．Trivium に対するスキャンベース攻撃の手法として Agrawal らの手法 [35] も存在する．どちらの手法もシミュレーション上では有効な攻撃手法であるが，実機を使っての有効性は確認されていない．Agrawal らの手法は暗号 LSI 中のスキャンチェイン上に Trivium 回路が使用するレジスタしか無いことを前提としているため，スキャンチェイン上に他のレジスタが存在する場合の暗号 LSI では用いることができないと考えられる．藤代らの提案した手法を使い，内部レジスタとスキャンデータの対応付けをした後，Trivium 回路の内部状態を復元する．

2.4 節「FPGA を使ったスキャンベース攻撃の実装実験」では，実装した Trivium 回路に対するスキャンベース攻撃した実験の結果をした．シミュレーション上だけでなくハードウェア上に実装し攻撃した．本実験では FPGA 上に Trivium 回路をコンフィギュレーションした．対象アーキテクチャは 2.3 節で説明した Trivium 回路である．スキャンベース攻撃として藤代ら [36] の手法を用いた．SASEBO-GII 上の 1024 ビットのレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，暗号用 FPGA からスキャンデータを取得することができた．また，得られたスキャンデータをスキャンベース攻撃手法を用いて解析した結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

第 3 章

連続動作するハッシュ回路に対するスキャンベース攻撃

3.1 本章の概要

本章[*1]では，連続動作するハッシュ回路に対するスキャンベース攻撃手法を提案する．連続動作するハッシュ回路として, メッセージ認証符号である HMAC-SHA-256 を生成する回路を対象とする．HMAC（Hash-based Message Authentication Code）は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．本章で攻撃対象とするハッシュ回路は SHA-256 回路である．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．実アプリケーションを想定して実装した回路も提案されており，実装方法における脆弱性を調査するため選定した．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成する回路である．攻撃の前提条件を示し，連続動作する HMAC-SHA-256 生成回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外の

[*1] 本章は ⟨2⟩, ⟨12⟩ で発表した内容による．

レジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

以下に本章の構成を示す．

3.2 節「メッセージ認証符号 HMAC-SHA-256」では，メッセージ認証符号の 1 つである HMAC-SHA-256 を説明し，攻撃対象となる HMAC-SHA-256 生成回路を説明する．HMAC は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．ハッシュ関数に SHA-256 [65, 66] を使うものを HMAC-SHA-256 と呼ぶ．

3.3 節「連続動作する HMAC-SHA-256 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃を提案する．攻撃対象となる HMAC-SHA-256 回路は連続してハッシュ値を生成する回路である．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．

3.4 節「HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験」では，連続動作する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示す．対象アーキテクチャは 3.2 節で説明した連続動作する HMAC-SHA-256 回路である．提案手法を用いてスキャンデータから内部レジスタの対応を求め，連続してハッシュ値を生成する HMAC-SHA-256 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 ハッシュ回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．スキャンチェイン長が 2048 ビットの時，入力する平

分を 425 個使い，最大で 7.5 時間程度で攻撃が可能であることを示した．

3.5 節「提案したスキャンベース攻撃手法の応用」では，3.3 節で提案した SHA-256 に対するスキャンベース攻撃手法を SHA-2 へのスキャンベース攻撃手法へ拡大し，提案されているアーキテクチャへの適応を議論する．

3.6 節「本章のまとめ」では，本章の内容をまとめる．

3.2 メッセージ認証符号 HMAC-SHA-256

本節ではメッセージ認証符号の 1 つである HMAC-SHA-256 を説明し，攻撃対象となる HMAC-SHA-256 生成回路を説明する．HMAC-SHA-256 はハッシュ関数 SHA-256 を用いたメッセージ認証符号である．本章での演算の表記法を表 3.1 に示す.

3.2.1 HMAC

本項ではハッシュ関数を用いたメッセージ認証符号である HMAC [64] を説明する．HMAC (Hash-based Message Authentication Code) は反復暗号ハッシュ関数を用いたメッセージ認証符号の 1 種である．ハッシュ関数 H，メッセージ M と鍵 K から認証コード値を以下の式により生成する．

$$HMAC(M, K) = H(K_0 \oplus \mathrm{opad} || H(K_0 \oplus \mathrm{ipad} || M))$$

HMAC の概略を図 3.1 に示す．図 3.1 で f は圧縮関数，ipad と opad は定数，IV は初期値を表す．K_0 は B ビット長であり，K から生成される．K_0 の生成式を式を以下に示す.

$$K_0 = \begin{cases} K & (L(K) = B) \\ K || \underbrace{0 \ldots 0}_{B - L(K)} & (L(K) < B) \\ H(K) || \underbrace{0 \ldots 0}_{B - L(H(K))} & (B < L(K)) \end{cases}$$

L をビット長を求める関数とし，$L(H(K)) \leq B$ とする．HMAC ではメッセージ M を B ビットのブロックに分割し $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ とする．B ビット長は使われるハッシュ関数によって異なる．K_{in} と K_{out} は鍵と見なせる H ビット長の値であり，$H^{(i)}$ は中間ハッシュ値と呼ばれる H ビット長の値である．H ビット長もハッシュ関数によって異なる．図 3.1 の通り HMAC では M を N 分割した時，圧縮関数を $N + 3$ 回適用する.

3.2.2 SHA-256

本項ではハッシュ関数 SHA-256 [65, 66] を説明する．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．SHA-

表 3.1: 本章での演算の表記法.

記号	意味
$\oplus$	排他的論理和
$\wedge$	論理積
$\bar{x}$	x の否定
$\leftarrow$	代入
$\boxplus$	2^{32} を法とする加算
$\|\|$	連結
S^n	n ビットの右シフト
R^n	n ビットの右ローテーション

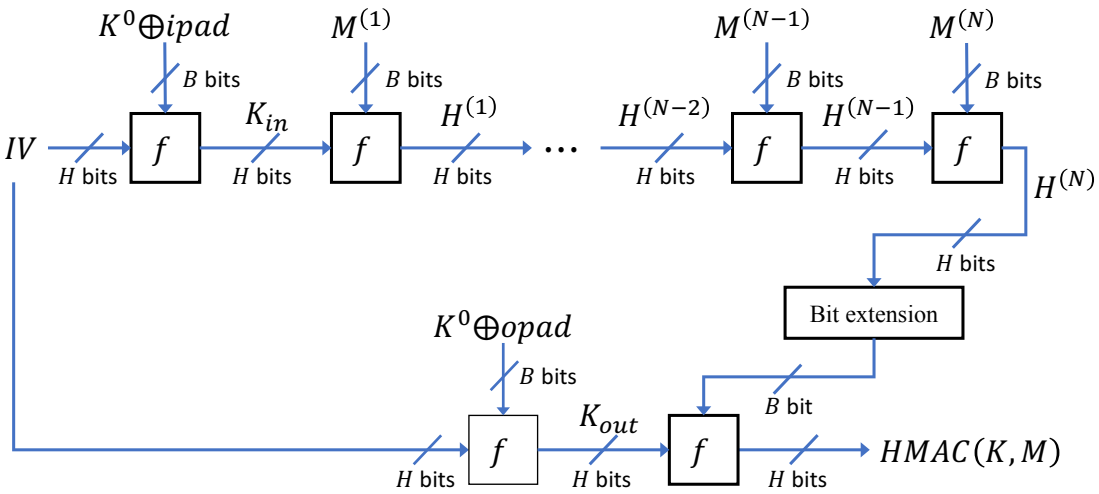


図 3.1: HMAC の概略.

256 ではメッセージ M をブロック長である 512 ビットに分割し，ブロック毎に $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ とする．例として，“abc” というメッセージを 8 ビットのアスキーコードを用いて変換すると，

$$\underbrace{01100001}_{\text{a}}\ \underbrace{01100010}_{\text{b}}\ \underbrace{01100011}_{\text{c}}$$

となる．全部で 24 ビットとなり 512 ビットに足りないためパディング操作する．パディング操作はメッセージ終了の直後に終了ビットとして 1 を挿入し，下位 64 ビットはメッセージ長を示す．終了ビットとメッセージ長の間に 0 を詰める．以

Algorithm 3 SHA-256

for $i = 1$ to N **do**

{**フェーズ 1:** 内部レジスタを初期化する}

$a \leftarrow H_1^{(i-1)}$; $b \leftarrow H_2^{(i-1)}$; $c \leftarrow H_3^{(i-1)}$; $d \leftarrow H_4^{(i-1)}$;

$e \leftarrow H_5^{(i-1)}$; $f \leftarrow H_6^{(i-1)}$; $g \leftarrow H_7^{(i-1)}$; $h \leftarrow H_8^{(i-1)}$;

{**フェーズ 2:** 圧縮関数を適用する}

for $j = 0$ to 63 **do**

$T_1 \leftarrow h \boxplus Rot_2(e) \boxplus Ch(e, f, g) \boxplus K_j \boxplus W_j$;

$T_2 \leftarrow Rot_1(a) \boxplus Maj(a, b, c)$;

$h \leftarrow g$; $g \leftarrow f$; $f \leftarrow e$;

$e \leftarrow d \boxplus T_1$;

$d \leftarrow c$; $c \leftarrow b$; $b \leftarrow a$;

$a \leftarrow T_1 \boxplus T_2$;

end for

{**フェーズ 3:** i 番目の中間ハッシュ値 $H^{(i)}$ を計算する}

$H_1^{(i)} \leftarrow a \boxplus H_1^{(i-1)}$; $H_2^{(i)} \leftarrow b \boxplus H_2^{(i-1)}$; $H_3^{(i)} \leftarrow c \boxplus H_3^{(i-1)}$;

$H_4^{(i)} \leftarrow d \boxplus H_4^{(i-1)}$; $H_5^{(i)} \leftarrow e \boxplus H_5^{(i-1)}$; $H_6^{(i)} \leftarrow f \boxplus H_6^{(i-1)}$;

$H_7^{(i)} \leftarrow g \boxplus H_7^{(i-1)}$; $H_8^{(i)} \leftarrow h \boxplus H_8^{(i-1)}$;

end for

$H^{(N)} = (H_1^{(N)} || H_2^{(N)} || \cdots || H_8^{(N)})$;

下にパディング後のメッセージを示す.

$$M^{(1)} = \underbrace{01100001}_{8}\ \underbrace{01100010}_{8}\ \underbrace{01100011}_{8}\ 1\ \underbrace{0\cdots0}_{423}\ \underbrace{0\cdots011000}_{64}$$

分割した 512 ビットのメッセージに対して繰り返し圧縮関数で処理する．i 番目の 256 ビットの中間ハッシュ値 $H^{(i)}$ は i 番目の 256 ビットのブロック $M^{(i)}$ と $i-1$ 番目の 256 ビットの中間ハッシュ値 $H^{(i-1)}$ から生成する．中間ハッシュ値の導出を式 (3.1) に示す.

$$H^{(i)} = f(M^{(i)}, H^{(i-1)}) \tag{3.1}$$

f は圧縮関数を示す．$H^{(0)}$ は定められている初期値を用いる．SHA-256 の圧縮関数 f のアルゴリズムを Algorithm3 に示す．Algorithm3 は 3 フェーズから成る．

フェーズ 1 では $i-1$ 番目の中間ハッシュ値 $H^{(i-1)}$ を 8 分割し，8 つの 32 ビットレジスタ a, b, c, d, e, f, g, h の初期値とする．$H_k^{(i-1)}$ は 8 分割された中間ハッシュ値 $H^{(i-1)}$ の k 番目を表す.

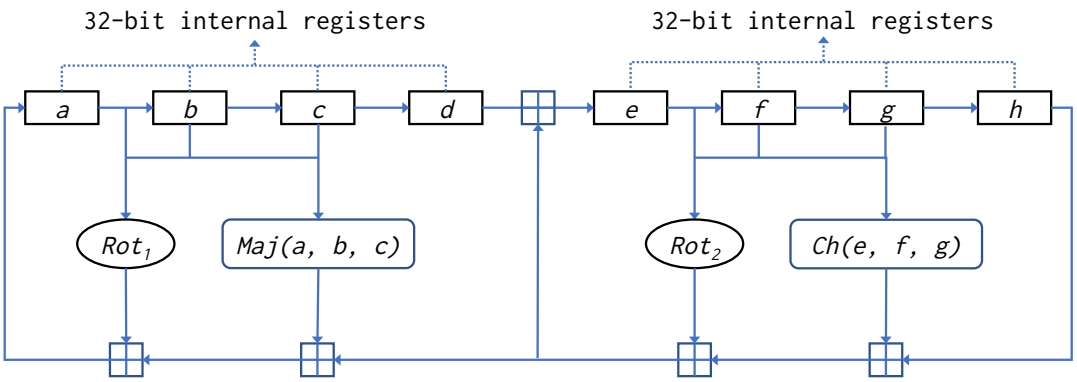


図 3.2: SHA-256 の圧縮関数.

フェーズ 2 ではレジスタの更新を 64 回する．Algorithm3 中のそれぞれ変数を以下に示す.

$$
\begin{aligned}
Ch(e,f,g) &= (e \wedge f) \oplus (\bar{e} \wedge g) \\
Maj(a,b,c) &= (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \\
Rot_1(a) &= R^2(a) \oplus R^{13}(a) \oplus R^{22}(a) \\
Rot_2(e) &= R^6(e) \oplus R^{11}(e) \oplus R^{25}(e)
\end{aligned}
$$

$K_j (j = 0 \ldots 63)$ は SHA-256 特有の 32 ビットの定数である．W_j は以下の式で求められる.

$$
W_j = \begin{cases} {M_j}^{(i)} & (j = 0, 1, \ldots, 15) \\ \sigma_1(W_{j-2}) \boxplus W_{j-7} \boxplus \sigma_0(W_{j-15}) \boxplus W_{j-16} & (j = 16, 17, \ldots, 63) \end{cases}
$$

ここで，${M_j}^{(i)}$ は $M^{(i)}$ を 16 分割した 32 ビットの値である．$\sigma_0(x)$ と $\sigma_1(x)$ は以下のように定義される.

$$
\begin{aligned}
\sigma_0(x) &= R^7(x) \oplus R^{18}(x) \oplus S^3(x) \\
\sigma_1(x) &= R^{17}(x) \oplus R^{19}(x) \oplus S^{10}(x)
\end{aligned}
$$

SHA-256 の圧縮関数の概略図を図 3.2 に示す．図中の四角形は 32 ビットのレジスタを表す.

フェーズ 3 では i 番目の中間ハッシュ値 $H^{(i)}$ をレジスタを初期化した値と更新後のレジスタの値から計算する.

3.2.3　HMAC-SHA-256 生成回路のアーキテクチャ

本項では攻撃対象となる HMAC-SHA-256 回路を説明する．HMAC の各圧縮関数について，SHA-256 の圧縮関数を用いたものを HMAC-SHA-256 という.

HMAC-SHA-256 は IPsec や SSL/TLS などの通信で利用されている [69, 70].

本章で想定する HMAC-SHA-256 回路は連続してハッシュ値を生成する回路である．図 3.2 で表される SHA-256 回路を 1 つ持ち，これを繰り返し用いることで HMAC-SHA-256 を実現する．SHA-256 回路の圧縮関数回路は 64 クロックで処理を完了する．Algorithm3 のフェーズ 1 とフェーズ 3 の処理を 1 クロックで完了する．つまり，SHA-256 回路は 66 クロックで処理を完了する．連続してハッシュ値を生成する回路は連続して SHA-256 回路が動作するため，全体としては $(N + 3) \times 66$ クロックで処理を完了する．

3.3 連続動作する HMAC-SHA-256 回路に対するスキャンベース攻撃手法

本節では，攻撃の前提条件を説明し，連続してハッシュ値を生成する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を提案する．

3.3.1 攻撃の前提条件

本項では攻撃の前提条件を説明する．HMAC-SHA-256 ハッシュ回路を対象としたサイドチャネル攻撃 [21–23] は，図 3.1 中の K_{in}，K_{out} を秘密鍵とみなし，復元対象としている．本章で提案する手法も既存研究と同様にスキャンベース攻撃で HMAC-SHA-256 ハッシュ回路の内部状態を復元し，秘密鍵 K_{in}，K_{out} を復元する．連続してハッシュ値を生成する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃の前提条件を以下に示す．

攻撃者がわかること

- SHA-256 回路が動作するタイミング（ハッシュ値を生成するタイミング）
- スキャンチェインはフルスキャン設計で，反転・動的に変化しないこと
- スキャンデータを圧縮していないこと

攻撃者ができること

- HMAC-SHA-256 ハッシュ回路に任意のメッセージを入力すること
- 任意のタイミングで HMAC-SHA-256 ハッシュ回路のスキャンチェインにアクセスし，スキャンデータを取得すること

攻撃者がわからないこと

- スキャンチェインに接続されているレジスタの接続順と数，種類

一般的な LSI ではスキャンチェイン上に複数の回路のレジスタが含まれていることがあるので，スキャンチェインに含まれるレジスタの構成・接続順情報を得る必要はない．任意のメッセージを用いて，任意のタイミングでスキャンチェインにアクセスすることでスキャンデータを得る．

対象となる HMAC-SHA-256 ハッシュ関数は連続して動作するため，連続してハッシュ値を生成する．想定される，スキャンデータを図 3.3 に示す．図 3.3 では，1 サイクル毎に得られるスキャンデータを縦に並べたものである．横はスキャ

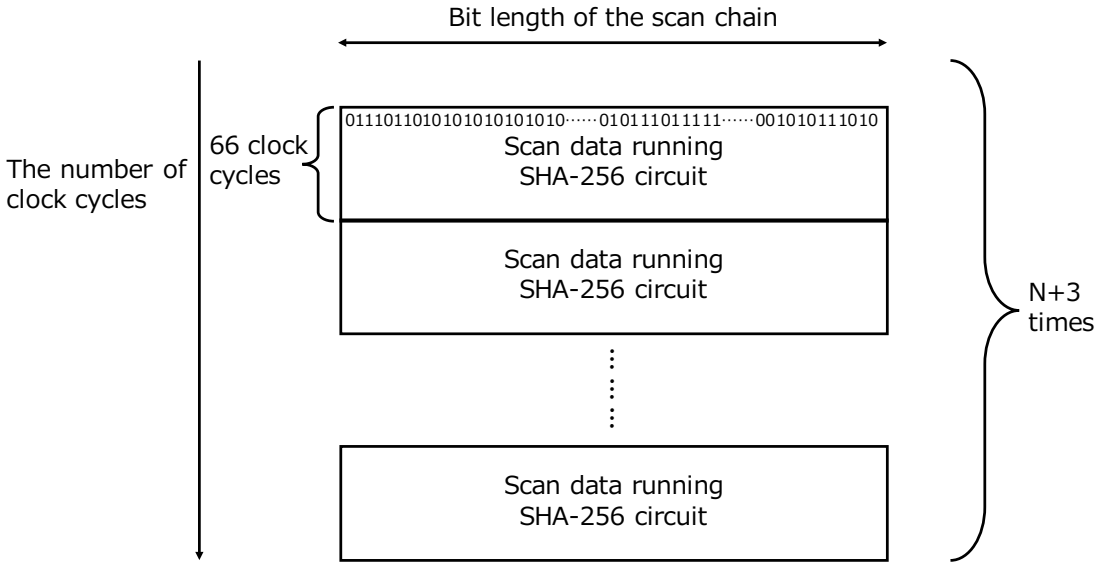


図 3.3: 想定している連続動作するハッシュ回路から得られるスキャンデータ.

ンチェイン長を表している．SHA-256 の動作は 3.2.2 項より，合計で 66 サイクルかかる．

3.3.2 スキャンベース攻撃提案手法

本項では，連続してハッシュ値を生成する HMAC-SHA-256 ハッシュ回路へのスキャンベース攻撃手法を提案する．スキャンチェインから得られるスキャンデータを解析することで秘密鍵を復元することを考える．アルゴリズム 3 のフェーズ 2 と図 3.2 を見ると，レジスタ a の値がレジスタ b，c，d に順に移動していることに気づく．同様に，レジスタ e の値がレジスタ f，g，h に順に移動していることに気づく．つまり，これらのレジスタに対応したスキャンデータのビット値も，サイクル毎に規則的に移動することになる. この性質を利用して，レジスタ $a \sim h$ の各ビットに対応するスキャンデータ中の各ビット位置を特定する．提案手法は遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップにより，内部レジスタとの対応関係を求めることで秘密鍵を復元することを考える．

遷移グループの特定

3.2.2 項のフェーズ 2 で示した圧縮関数より，レジスタ a の値はレジスタ b，c，d の順で遷移し，レジスタ e の値はレジスタ f，g，h の順で遷移する．レジスタ x の i 番目のビットを x_i としたとき，a_i は b_i，c_i，d_i の順で遷移する．同様に，e_i

は f_i，g_i，h_i の順で遷移する．スキャンデータ中で遷移する a_i，b_i，c_i，d_i のビット位置，もしくは e_i，f_i，g_i，h_i のビット位置を遷移グループと呼ぶ．

スキャンデータ中の遷移グループを特定するためにビット列の遷移をスキャンデータ内から探索する．異なるサイクルで得たスキャンデータを縦に並べたとき，スキャンデータの k 番目のレジスタ値の変化が読み取れる．この k 番目のビット値の変化列をスキャンシグネチャと呼ぶ．スキャンシグネチャを用いてスキャンデータ内を探索することで遷移グループが特定できると考える．

レジスタ a の i ビット目の遷移を探索する様子を図 3.4 に示す．図 3.4 ではスキャンデータを縦に並べ，あるレジスタのビット値のスキャンシグネチャに着目する．a_i は次の時刻 $T+1$ に b_i に遷移するため，スキャンデータ中の時刻 T のあるビットが a_i であるとすれば，時刻 T のそのビット値は時刻 $T+1$ のどこかに存在する．3.2.2 項のフェーズ 2 で示した圧縮関数より，この遷移は 64 回繰り返されるため，時刻 $T \sim T+n$ のスキャンシグネチャは時刻 $T+1 \sim T+n+1$ のいずれかのスキャンシグネチャと一致する．図 3.4 のレジスタ a の i ビット目を示す枠で囲われた列の内，$T \sim T+3$ のスキャンシグネチャはレジスタ b の i ビット目を示す $T+1 \sim T+4$ のスキャンシグネチャと一致する．同様にして，レジスタ c の i ビット目とレジスタ d の i ビット目も探索できる．このようにして，レジスタ a の i ビット目の遷移グループの特定ができる．n を適当に大きく取れば $(n < 64)$ こうした遷移が特定できる．時刻 $T \sim T+n$ のスキャンデータ中の特定の 1 ビットから構成されるスキャンシグネチャを考えたとき，その長さを $(n+1)$ と定義する．

スキャンシグネチャを用いた遷移グループの特定の結果，レジスタ a もしくは e どちらに属しているかという点，何番目のビットであるかという点が不明である遷移グループが合わせて 64 個特定できる．

ビット位置の特定

遷移グループの特定で 64 個の遷移グループが特定できる．ビット位置の特定では，64 個の遷移グループと初期位置の特定ができたスキャンデータから 2 グループずつペア化し，32 個のペアを作ることを考える．各ペアはレジスタ a あるいは e の i 番目のビットを表す．

遷移グループのビット位置を特定するためにメッセージ m^1, m^2 の 2 種類を用いる．時刻 T から時刻 $T+1$ でレジスタ a, e が更新される際，フェーズ 2 よりレジスタ $b \sim d$，$f \sim h$ の値を使う．$b \sim d$，$f \sim h$ を定数とみなすと，定数 A, E を用いて，メッセージ m^1 を入力したときの時刻 $T+1$ のレジスタ a^1，e^1 は以下の式

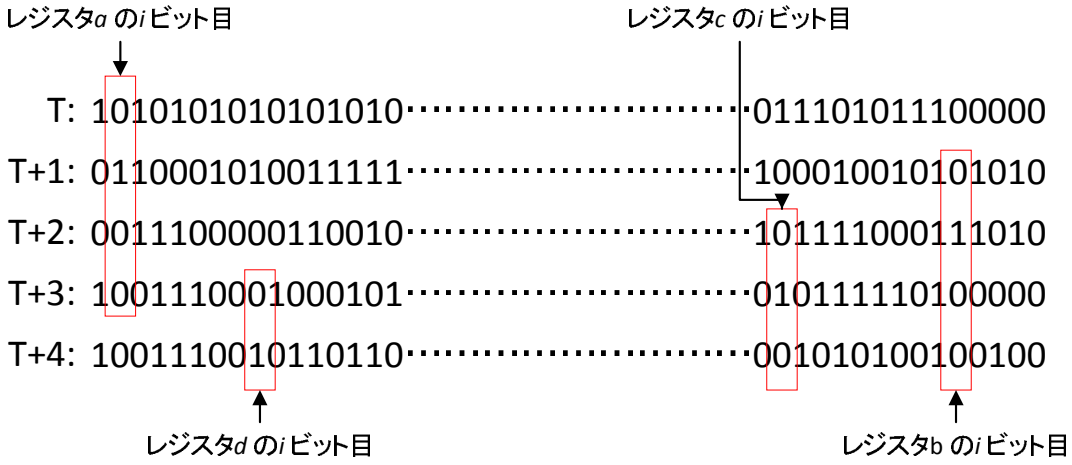


図 3.4: レジスタ a の i ビット目の遷移の探索.

で表せる.

$$a^1 = A \boxplus W_0^1 \tag{3.2}$$
$$e^1 = E \boxplus W_0^1 \tag{3.3}$$

メッセージ m^2 を入力したときの時刻 $T+1$ のレジスタ a^2, e^2 は以下の式で表せる.

$$a^2 = A \boxplus W_0^2 \tag{3.4}$$
$$e^2 = E \boxplus W_0^2 \tag{3.5}$$

W_j^1, W_j^2 はメッセージに依存した 32 ビットの変数である. 適当な変数 α を用いると $W_i^2 = W_i^1 + \alpha$ と表せる. α の MSB が 1 でその他のビットは 0 であるとき, 式 (3.2), (3.3), (3.4), (3.5) より, a^1, e^1 の MSB は a^2, e^2 の MSB の反転した値になる. このような 2 組のメッセージを入力した結果, 反転したビットがレジスタ a か e の MSB であるとわかる. a, e の MSB が定まった次は α を MSB から 1 つ下の位のビットのみが 1 であるとし, 反転するビット位置を求める. 順々に求めることで遷移グループのビット順が特定できる.

HMAC-SHA-256 では, 入力するメッセージはアスキーコードを用いて数値化する. アスキーコードは 8 ビットの値であり, 0x00 から 0x7F までの値をとる. 8 ビットのアスキーコードでハミング距離が 1 である例を表 3.2 に示す. 表 3.2 では W_i^1, W_i^2 を 8 ビットずつに分割した時の一要素の内, どのビット位置が反転しているかを示している. 表 3.2 中の文字を組み合わせることで, ビット順を求めることが可能であると考えられる. しかし, MSB のみが 1 である文字は存在せず, レジスタ a と e の MSB を直接求めることはできない. そこで, 以下のようにこれを解決する.

表 3.2: ハミング距離が 1 である例.

2 組の文字	異なるビット位置
N/A	10000000
(q, 1)	01000000
(q, Q)	00100000
(q, a)	00010000
(q, y)	00001000
(q, u)	00000100
(q, s)	00000010
(q, p)	00000001

MSB の 1 つ下のビット位置を 1 番目のビットと呼ぶ．1 番目のビットを求めるためには，2 つのメッセージを $W_i^2 = W_i^1 + \alpha$ と表した時，α の 1 番目のビットだけが異なるメッセージを入力することを考える．定数 A を以下のように定義する．

$$A = 10001110010101000011001000010000$$

1 番目のビット位置が異なる 2 つのメッセージの例として，表 3.2 から 32 ビットのメッセージ “qqqq” と “1qqq” を入力する．W_0^1 と W_0^2 はアスキーコードとして以下のように表せる．

$$\begin{aligned} W_0^1 &= \text{“qqqq”} \\ &= 0\underline{1}11000101110001011100010111000 1 \\ W_0^2 &= W_0^1 \oplus 01000000000000000000000000000000 \\ &= 0\underline{0}11000101110001011100010111000 1 \\ &= \text{“1qqq”} \end{aligned}$$

式 (3.2) と (3.4) より a^1 と a^2 は以下のようになる．

$$\begin{aligned} a^1 &= A \boxplus W_0^1 \\ &= 1\underline{1}11111111000101101000111000000 1 \\ a^2 &= A \boxplus W_0^2 \\ &= 1\underline{0}11111111000101101000111000000 1 \end{aligned}$$

この場合，1 番目のビットのみが異なる値が得られるため，1 番目のビット位置が判明する．

次に 1 番目のビット位置が異なるとして，2 つのメッセージ “rrrr” と “2rrr” を入力する．

$$
\begin{aligned}
A &= 10001110010101000011001000010000 \\
W_0^{1\prime} &= \text{“rrrr”} \\
&= 0\underline{1}1100110111001101110011011110011 \\
W_0^{2\prime} &= W_0^{1\prime} \oplus 01000000000000000000000000000000 \\
&= 0\underline{0}1100110111001101110011011110011 \\
&= \text{“2rrr”}
\end{aligned}
$$

式 (3.2) と (3.4) より a^1 と a^2 は以下のようになる．

$$
\begin{aligned}
a_0^{1\prime} &= A \boxplus W_0^{1\prime} \\
&= \underline{00}00000011000110101001001000010 \\
a_0^{2\prime} &= A \boxplus W_0^{2\prime} \\
&= \underline{11}00000011000110101001001000010
\end{aligned}
$$

この場合，MSB と 1 番目のビットが異なる．

結果として，1 番目のビットは必ず反転し，繰り上がりがあった場合，MSB も反転する可能性がある．つまり，最大でレジスタ a と e の各々 MSB と 1 ビット目がすべて反転し（4 ビット分が反転し），最小でもレジスタ a と e の各々 1 ビット目が反転する（2 ビット分が反転する）．こうしたメッセージの組を多数入力し反転したビットを数えた時，MSB が反転する回数は 1 番目のビットが反転した値が出る回数より少なくなると予測できる．必ず反転するビット位置を求めれば 1 番目のビットを求めることができる．1 番目のビットが求まると，反転している他のビットは MSB となる．2 から 7 ビット目は表 3.2 の例にならい入力することで求めることができる．8，16，24 番目のビットを定めるためにも 1 番目のビットを求める手法と同様なことをする．上記の通り，表 3.2 中の文字を組み合わせることで，スキャンデータからレジスタのビット位置を求めることが可能となる．つまり，32 組の各ペア中の 2 つの遷移グループは前半レジスタか後半レジスタの i 番目のビットを表す．

前半レジスタと後半レジスタの判定

遷移グループの特定でスキャンデータから 64 個の遷移グループを特定した．ビット位置の特定では 64 個の遷移グループを 32 組のペアに分類したとき，各ペア中の 2 つの遷移グループが内部レジスタ中のどのビット位置であるか判明している．そのため，遷移グループが前半のレジスタであるレジスタ a か後半レジス

タであるレジスタ e どちらから始まるのかを求める必要がある．前半レジスタと後半レジスタを判別するためにフェーズ 2 中の式を利用する．時刻 T のレジスタと時刻 $T+1$ のレジスタの関係は以下のように表せる．

$$a^{T+1} + d^T = e^{T+1} + Rot_1(a^T) + Maj(a^T, b^T, c^T)$$

必要なレジスタは a，b，c，d，e である．等式を満たすように遷移グループを選べばレジスタ a，e かを特定できる．

それぞれのレジスタの上位ビットを求める場合，下位ビットから繰り上がりを考慮する必要があるため，LSB から順にビットを求める．レジスタ a の LSB を a_{31} とする．LSB を求めるとき，式 3 より a_{29}，a_{18}，a_9 が必要となる．31 ビット目の遷移グループ，29 ビット目の遷移グループ，18 ビット目の遷移グループ，9 ビット目の遷移グループが必要であり，2^4 パターン分を試行する．a_{31} が定まると同時に a_{29}，a_{18}，a_9 も定まる．前半レジスタが定まると後半レジスタも定まるので e_{31} も定まる．

レジスタ位置を特定した後，K_{in} と K_{out} が使われているタイミングを特定する必要がある．最小なメッセージ “q” を入力した時に K_{in} と K_{out} を求めることを考える．K_{out} はハッシュ回路が最後に動いたタイミングで使われるため，ハッシュ値を生成する最後のタイミングで取得したスキャンデータから求めることができる．K_{in} を求める際はハッシュ値を生成するタイミングの各スキャンデータを比較する．最小のメッセージをハッシュ化する際，メッセージを分割しない．そのため，図 3.1 より，ハッシュ値を生成するタイミングは全部で 4 回ある．4 回のうち，1 回は K_{out} を初期値とし，2 回はもともとの初期値 IV を使ってハッシュ値を生成する．残りは K_{in} を初期値として使用する．そのため，レジスタが初期化された時のスキャンデータから内部レジスタを復元して K_{in} を得ることができる．

3.4 HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験

本節では 3.3 提案した連続動作する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示す.

3.4.1 実験環境

本項では実装実験の環境を示す．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 シミュレータを python を用いて実装した．ホスト PC の OS は OS X El Sierra，CPU は Intel Core i5（2.6 GHz），メモリは 8GB のものを使用した.

3.4.2 実験手順

本項では実験手順を示す．前提条件としては 3.3.1 項で紹介した前提条件を用いる．HMAC-SHA-256 回路はランダムな秘密鍵を持つものとし，動作中の各サイクルごとの SHA-256 のレジスタ値（$a \sim h$）を全て観測し，スキャンデータとした．つまり，HMAC-SHA-256 回路のスキャンチェイン長は 256 ビットとなる．さらにランダムに 0 ビットから 1792 ビットのビット列を生成しスキャンデータ中に加え，スキャンデータ長は 256 ビットから 2048 ビットとした．これらランダムデータは，HMAC-SHA-256 回路の外部に設置された周辺回路のスキャンデータに相当する.

本実験では，1 つのスキャンデータ長に対して，50 種類の秘密鍵を HMAC-SHA-256 回路にランダムに設定し，提案するスキャンベース攻撃手法で，秘密鍵を復元した．図 3.3 のようなスキャンデータの構成とした.

3.4.3 実験結果

各スキャンチェイン長で秘密鍵を復元した．提案するスキャンベース攻撃手法で，秘密鍵の復元に成功した．つまり，スキャンチェイン上に複数の回路のレジスタが存在していても攻撃が可能である．スキャンチェイン長を変化させた時，秘密鍵を復元させるのにかかった時間を図 3.5 に示す．図 3.5 ではそれぞれのスキャンチェイン長で最大時間，最小時間，平均時間を示している．秘密鍵を復元させるのに必要なメッセージ数を図 3.6 に示す．スキャンチェイン長が 2048 ビットの

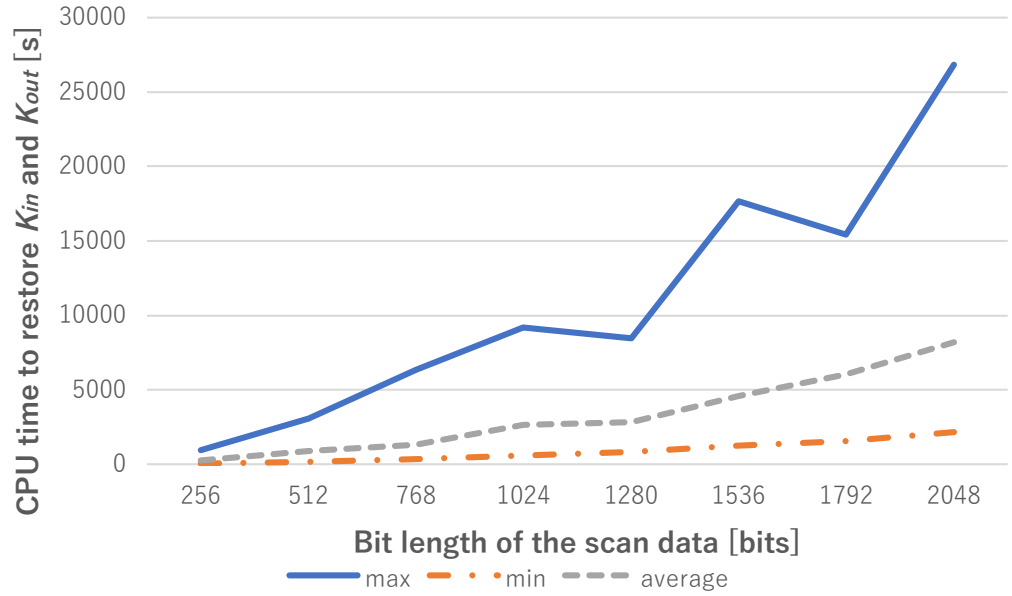


図 3.5: 連続してハッシュ値を生成する回路の秘密鍵 K_{in}，K_{out} を復元する時間.

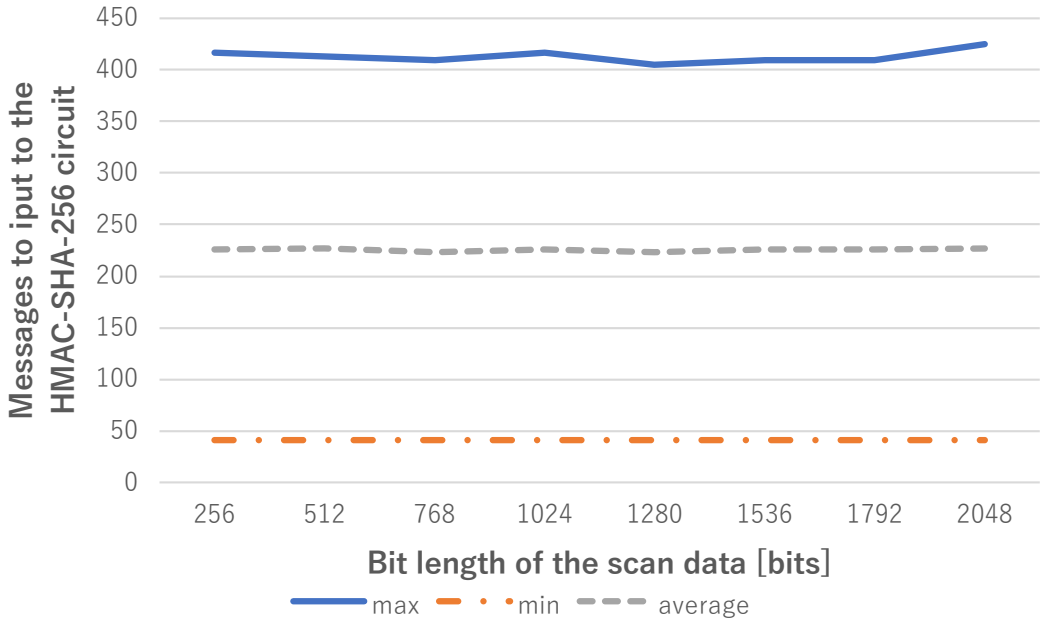


図 3.6: 連続してハッシュ値を生成する回路の秘密鍵 $K_{\mathrm{in}}, K_{\mathrm{out}}$ を復元するために必要な時間メッセージ数.

時，入力する平分を 425 個使い，最大で 7.5 時間程度で攻撃が可能であることを示した.

3.5　提案したスキャンベース攻撃手法の応用

3.5.1　SHA-2 に対するスキャンベース攻撃

SHA-256 は SHA-2 ファミリーの種類の 1 種である．SHA-256 の他に SHA-224, SHA-384, SHA-512, SHA-512/224 と SHA-512/256 の 6 種類のハッシュ関数が存在する [65,66]．3.3 節では HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃を提案したが，その他ハッシュ関数に適応することを考える．

SHA-256 ではレジスタのワード長を 32 ビットとしていたが，SHA-512 はレジスタのワード長を 64 ビットとする．初期値 IV，K_j，とラウンド数が SHA-256 とは異なるが，アルゴリズムの構造は同じである．そのため，HMAC-SHA-512 ハッシュ回路に対しても同様なスキャンベース攻撃アルゴリズムで攻撃が可能となると考えられる．SHA-224 と SHA-384 は SHA-256 で得られるハッシュ値と SHA-512 で得られるハッシュ値をそれぞれ 224 ビットと 384 ビットに切り詰めたものである．アルゴリズムは同じであるが初期値が異なる．SHA-512/224 と SHA-512/256 は SHA-512 で得られるハッシュ値を 224 ビットと 256 ビットに切り詰めたものである．HMAC-SHA-224，HMAC-SHA-384，HMAC-SHA-512/224 と HMAC-SHA-512/256 に関しても提案したスキャンベース攻撃手法のアルゴリズムで回路の秘密鍵が復元可能であると考えられる．

3.5.2　SHA-256 回路の別実装に対するスキャンベース攻撃

3.3 節で攻撃対象としている HMAC-SHA-256 回路は以下の 2 つの条件を満たしている．

(A) 1 つの SHA-256 回路を持っており，SHA-256 回路を複数回使う．

(B) SHA-256 のアルゴリズムで PHASE 1 に 1 サイクル，PHASE 2 に 64 サイクル，PHASE 3 に 1 サイクル，つまり SHA-256 のアルゴリズムを全体で 66 サイクルかけて実行する．

条件 (A)，(B) を満たす HMAC-SHA-256 回路は文献 [67] よるものである．HMAC-SHA256 回路としては最も単純なもので，小面積で，HMAC-SHA-256 よるハッシュ値生成ができる．この実装例は HMAC-SHA-256 回路に対するサイドチャネル攻撃 [24] でも攻撃対象となっている．本章では，HMAC-SHA256 回路に対するスキャンベース攻撃手法を提案するにあたり，この実装を取り上げた．しかし，実際には HMAC-SHA256 回路の実装例として，別なものが存在する．以下で

は，SHA-256 回路の別実装と HMAC-SHA256 回路の別実装に分けて議論する．

SHA-256 回路の実装方法によるスキャンベース攻撃

Basic iterative SHA-256 回路 [67, 80]

SHA-256 回路の basic iterative 実装は上記条件 (B) を満たす回路である．提案するスキャンベース攻撃はこれを対象としている．

Two-unrolled SHA-256 回路 [81, 82]

SHA-256 回路の 2-unrolled implementation は Algorithm 3 の PHASE 2 を 64 サイクルでなく 32 サイクルかけて処理する回路である．この実装では，図 3.2 で，1 サイクル中にレジスタ a からレジスタ c へのデータ遷移，レジスタ b からレジスタ d へのデータ遷移，レジスタ e からレジスタ g へのデータ遷移，レジスタ f からレジスタ h へのデータ遷移がある．そこで，多くのスキャンデータの中からこのようなビット遷移を探し出すことによって，遷移グループを特定できると思われる．この SHA-256 回路に平文を入力できるとすれば，提案手法によって攻撃が可能と考えられる．

Pipelining SHA-256 回路 [83, 84]

SHA256 回路の pipelining implementation は Algorithm 3 の PHASE 2 をパイプライン処理する回路である．この実装では，図 3.2 の左側（レジスタ a, b, c, d から構成される部分回路）と右側（レジスタ e, f, g, h から構成される部分回路）がパイプライン処理される．この場合，提案手法の遷移グループの特定と同様に 1 サイクルでレジスタ a がレジスタ b に遷移し，レジスタ b がレジスタ c に遷移するなど，遷移グループを見つけ出すことができると思われる．この SHA-256 回路に平文を入力できるとすれば，提案手法によって攻撃が可能と考えられる．

Two-unrolled pipelining SHA-256 回路 [85]

SHA256 回路の two-unrolled pipelining SHA-256 回路は 2-unrolled SHA-256 回路の動作をパイプライン処理する回路である．上記の Two-unrolled SHA-256 回路と Pipelining SHA-256 回路の議論を組み合わせることで，Two-unrolled pipelining SHA256 回路も同様な手法で攻撃できると思われる．

Four-unrolled pipelining SHA-256 回路 [85]

SHA-256 回路の four-unrolled implementation は Algorithm3 の PHASE 2 を 64 サイクルでなく 16 サイクルかけて処理し，更にこれを 2 ステージパイプライン化した回路である．この場合，Basic iterative SHA-256 回路や Two-unrolled SHA-256 回路と異なり，あるレジスタの値

が別なレジスタに直接遷移することがなく，遷移グループを特定することができない．この実装に対しては提案するスキャンベース攻撃を適応することができない．

Compact SHA-256 回路 [86]

SHA-256 回路の compact implementation は Algorithm3 の PHASE 2 の 1 イタレーションを 1 サイクルでなく複数クロックかけて処理する回路である．この場合，1 イタレーションの中でどのタイミングでレジスタ遷移が起きるかわかれば，各レジスタの値がどのように遷移するかわかり，これをもとにスキャンデータの中から遷移グループを特定できると思われる．この SHA-256 回路に平文を入力できるとすれば，提案手法によって攻撃が可能と考えられる．

上記のように，提案手法はこれまで提案された多くの SHA-256 回路の実装方法に対して適応できることが期待できる．しかしながら，実際にどの程度のスキャンデータが必要になるかどの程度の計算時間が必要になるかはシミュレーションや実機実験が必要である．これらは今後の課題とする．

HMAC-SHA-256 回路の実装方法によるスキャンベース攻撃

前述の通り，提案手法がターゲットとしている実装方法 [67] が最も単純かつ一般的と考えるが，他のアーキテクチャの例として，HMAC-SHA-256 を実現するために，2 つの SHA-256 回路を利用するアーキテクチャがある．2 つの SHA-256 回路をそれぞれ SHA_a，SHA_b とする．

HMAC-SHA-256 回路を動作させると回路 SHA_a，SHA_b が動作する．動作中 HMAC-SHA-256 回路から得られるスキャンデータを縦に並べ遷移グループを特定すると，2 つの SHA-256 回路 SHA_a，SHA_b があるため，遷移グループは 64 個より多く特定される可能性がある．提案手法では平文を使い遷移グループからレジスタ位置を特定する．回路 SHA_a，SHA_b へ独立してそれぞれ平文を挿入できれば，回路 SHA_a の遷移グループと回路 SHA_b の遷移グループは独立して考えることができる．そのため，SHA-256 回路に平文を入力できるとすれば，提案手法によって攻撃が可能と考えられる．

しかし，回路 SHA_a と回路 SHA_b に独立して平文をそれぞれ挿入できない場合，例えば，SHA_a には任意のメッセージ M を入力できるが，SHA_b には，M を加工した別なメッセージ M' のみ入力される場合には，提案手法により遷移グループは求めることができるが，レジスタ位置を特定できない．そのため，提案するスキャンベース攻撃を適応することができない．

3.6 本章のまとめ

本章では，連続動作するハッシュ回路に対するスキャンベース攻撃手法を提案した．連続動作するハッシュ回路として，メッセージ認証符号である HMAC-SHA-256 を生成する回路を対象とする．HMAC（Hash-based Message Authentication Code）は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．本章で攻撃対象とするハッシュ回路は SHA-256 回路である．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成する回路である．攻撃の前提条件を示し，連続動作する HMAC-SHA-256 生成回路へのスキャンベース攻撃手法を提案した．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

以下に本章の構成を示す．

3.2 節「メッセージ認証符号 HMAC-SHA-256」では，メッセージ認証符号の 1 つである HMAC-SHA-256 を説明し，攻撃対象となる HMAC-SHA-256 生成回路を説明した．HMAC（Hash-based Message Authentication Code）は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．ハッシュ関数に SHA-256 [65, 66] を使うものを HMAC-SHA-256 と呼ぶ．

3.3 節「連続動作する HMAC-SHA-256 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃を提案した．攻撃対象となる HMAC-SHA-256 回路は連続してハッシュ値を生成する回路である．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．

3.4 節「HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験」では，計算機上で HMAC-SHA-256 ハッシュ回路に対してスキャンベース攻撃した実験結果を示した．対象アーキテクチャは 3.2 節で説明した連続動作する HMAC-SHA-256 回路である．提案手法を用いてスキャンデータから内部レジスタの対応を求め，連続してハッシュ値を生成する HMAC-SHA-256 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 ハッシュ回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．スキャンチェイン長が 2048 ビットの時，入力する平分を 425 個使い，最大で 7.5 時間程度で攻撃が可能であることを示した．

3.5 節「SHA-2 へのスキャンベース攻撃手法の適応」では，3.3 節で提案した SHA-256 に対するスキャンベース攻撃手法を SHA-2 へのスキャンベース攻撃手法へ拡大し，提案されているアーキテクチャへの適応を議論した．

第 4 章

連続動作しないハッシュ回路に対するスキャンベース攻撃

4.1 本章の概要

本章[*1]では，連続動作しないハッシュ回路に対するスキャンベース攻撃手法を提案する．連続動作しないハッシュ回路として，メッセージ認証符号である HMAC を対象とする．本章で攻撃対象とするハッシュ回路は SHA-256 である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成しない回路である．攻撃の前提条件を示し，連続動作しない HMAC-SHA-256 回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

以下に本章の構成を示す．

4.2 節「連続動作しない HMAC-SHA-256 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，HMAC-SHA-256 回路に対するスキャンベース攻撃を提案する．攻撃対象となる HMAC-SHA-256 回路は連続してハッ

[*1] 本章は ⟨9⟩, ⟨16⟩ で発表した内容による．

シュ値を生成する回路である．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．

4.3 節「HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験」では，連続動作しない HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示す．対象アーキテクチャは 4.2 節で説明した連続動作しない HMAC-SHA-256 回路である．提案手法を用いてスキャンデータから内部レジスタの対応を求め，連続してハッシュ値を生成する HMAC-SHA-256 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 ハッシュ回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．スキャンチェイン長が 4096 ビットであっても秘密鍵を復元できることを確認した．スキャンチェイン長が 3840 ビットの時，入力する平分を 81 個使い，最大で 80 時間程度で攻撃が可能であることを示す．

4.4 節「本章のまとめ」では，本章の内容をまとめる．

4.2 連続動作しない HMAC-SHA-256 回路に対するスキャンベース攻撃手法

本節では，攻撃の前提条件を説明し，連続してハッシュ値を生成しない HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を提案する．

4.2.1 攻撃の前提条件

本項では攻撃の前提条件を説明する．本章も 3 章と同様に図 3.1 中の K_{in}，K_{out} を秘密鍵とみなし，復元対象としている．本章で提案する手法も既存研究と同様にスキャンベース攻撃で HMAC-SHA-256 ハッシュ回路の内部状態を復元し，秘密鍵 K_{in}，K_{out} を復元する．秘密鍵 K_{in}，K_{out} を復元することとする．連続してハッシュ値を生成しない HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃の前提条件を以下に示す．

攻撃者がわかること

- スキャンチェインはフルスキャン設計で，反転・動的に変化しないこと
- スキャンデータを圧縮していないこと

攻撃者ができること

- HMAC-SHA-256 ハッシュ回路に任意のメッセージを入力すること
- 任意のタイミングで HMAC-SHA-256 ハッシュ回路のスキャンチェインにアクセスし，スキャンデータを取得すること

攻撃者がわからないこと

- スキャンチェインに接続されているレジスタの接続順と数，種類

一般的な LSI ではスキャンチェイン上に複数の回路のレジスタが含まれていることがあるので，スキャンチェインに含まれるレジスタの構成・接続順情報を得る必要はない．任意のメッセージを用いて，任意のタイミングでスキャンチェインにアクセスすることでスキャンデータを得る．

対象となる HMAC-SHA-256 ハッシュ関数は連続して動作しない．つまり，SHA-256 の圧縮関数により中間ハッシュ値が生成してから再び SHA-256 の圧縮関数が動作するまでに数サイクルかかる．想定されるスキャンデータを図 4.1 に示す．図 4.1 では，1 サイクル毎に得られるスキャンデータを縦に並べたものである．

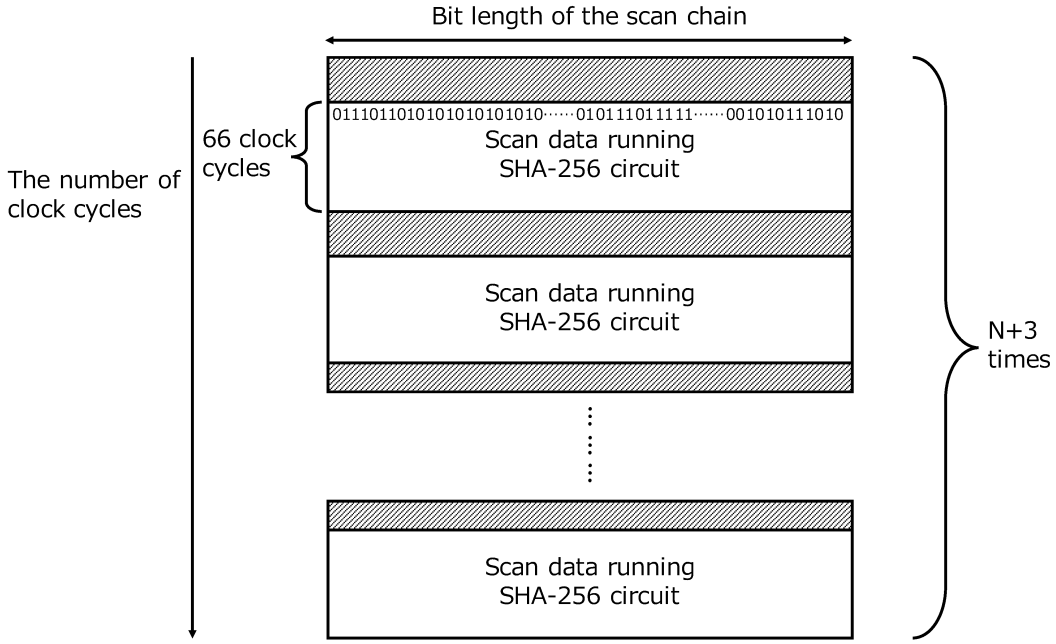


図 4.1: 想定している連続してハッシュ値を生成しない回路から得られるスキャンデータ.

横はスキャンチェイン長を表している．SHA-256 の動作は 3.2.2 項より，合計で 66 サイクルかかる．動作終了後は再び動作するまでに数サイクル必要となる．図 4.1 の灰色部分はランダムデータになると考えられる.

4.2.2 連続してハッシュ値を生成しない HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法

本項では，連続してハッシュ値を生成しない HMAC-SHA-256 ハッシュ回路へのスキャンベース攻撃手法を提案する．3 章では連続してハッシュ値を生成する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を提案した．3 章で想定しているスキャンデータは図 3.3 である．図 3.3 では，遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップで秘密鍵の復元が可能であった．しかし，本項で想定しているスキャンデータは図 4.1 である．図 4.1 では SHA-256 回路動作とは関係のないスキャンデータがあるため，上記の 3 ステップだけでは秘密鍵を復元できないと考えられる．そこで，SHA-256 回路動作の初期位置を特定するステップを加え，遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の順の 4 ステップにより，内部レジスタとの対応関係を求めることで秘密鍵を復元することを考える.

遷移グループの特定

ランダムデータを含むスキャンデータへ 3 章で提案した遷移グループの特定をそのまま用いるとランダムデータがあるため，遷移グループが特定できないと考えられる．そのため，スキャンデータの先頭から順に遷移グループの特定をし，特定できる部分を探す．

ランダムデータを含むスキャンデータへの遷移グループの特定を図 4.2 に示す．図 4.2 で実線枠，破線枠はあるレジスタのスキャンシグネチャを示している．破線枠のスキャンシグネチャはランダムデータも含んでいるため遷移グループの特定はできない．実線枠のスキャンシグネチャはランダムデータを含んでいないため遷移グループの特定が可能である．つまり，遷移グループの特定のためには，スキャンシグネチャの範囲が SHA-256 回路が動いている間のスキャンデータである必要がある．ただし，スキャンデータの先頭から順に探索すると膨大に時間がかかる．そのため，スキャンシグネチャ長を 32 サイクル分取り，32 サイクル毎に遷移グループの特定をする．

HMAC-SHA-256 ではメッセージを最小にした時，SHA-256 回路は 4 回動く．遷移グループの特定によって，特定できるタイミングが 4 回ある．遷移グループの特定はできるが，レジスタの初期位置が特定できていないため，3 章で提案したビット位置の特定ができない．ビット位置の特定をするために，レジスタの初期位置を特定する．

SHA-256 回路動作の初期位置の特定

次に，SHA-256 回路が動作を開始する初期位置を探索する．HMAC-SHA-256 ではメッセージを最小にした時，SHA-256 回路は 4 回動作する．つまり，メッセージを最小にした時，遷移グループの特定が可能となる 32 サイクル長の部分スキャンデータが 4 個あることになる．特定できたサイクルからから前のサイクルに遡ることで，SHA-256 のレジスタが初期化された位置を特定し，動作を開始する初期位置を特定する．SHA-256 のレジスタが初期化された位置から 64 サイクルは SHA-256 が動作していることになり，動作中のスキャンデータを特定することができる．

遷移グループの特定から遷移グループが 64 個特定できた．遷移グループの特定では，スキャンシグネチャ長を 32 サイクル分とし，32 サイクルごとに探索するため，レジスタの初期位置はわからない．SHA-256 回路が動いた回数だけ遷移グループの特定の可能タイミングがある．特定できたタイミングから前のサイクルにさかのぼることで，レジスタが初期化された位置が特定できる．初期化されたサ

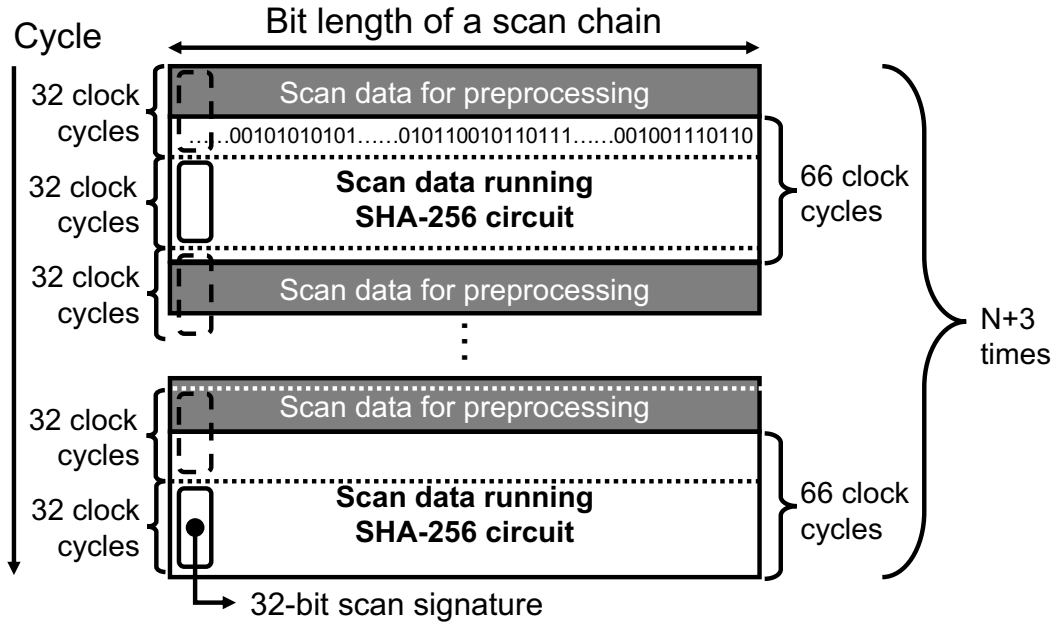


図 4.2: ランダムデータを含むスキャンデータへの探索.

イクルから 64 サイクルは SHA-256 回路が動いているスキャンデータである．初期位置の特定によって，初期位置から 64 サイクル分のスキャンデータをハッシュ値が出力された回数 L だけ求められる．L 個のスキャンデータからビット位置の特定をする．

ビット位置の特定

遷移グループが 64 個特定でき，レジスタが初期化されてから 64 サイクル分のスキャンデータが得られた．本目では，64 個の遷移グループと初期位置の特定ができたスキャンデータから 2 グループずつペア化し，32 個のペアを作ることを考える．各ペアはレジスタ a あるいは e の i 番目のビットを表す．3 章で提案したビット位置の特定と同じアルゴリズムを使う．

前半レジスタと後半レジスタの判定

スキャンデータから 64 個の遷移グループを特定し，スキャンデータからレジスタの初期化位置を特定し，64 個の遷移グループを 32 組のペアに分類した．つまり，各ペア中の 2 つの遷移グループが内部レジスタ中のどのビット位置であるか判明している．次に遷移グループがレジスタ a，e どちらから始まるのかを求める．3 章で提案した前半レジスタと後半レジスタの特定と同じアルゴリズムを使う．

4.3 HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験

本節では 4.2 提案した連続動作する HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示す.

4.3.1 実験環境

本項では実装実験の環境を示す．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 シミュレータを python を用いて実装した．計算機は OS は Cent OS，CPU は Intel Xeon CPU E7-8855 v4（2.9 GHz），メモリは 1TB のものを使用した.

4.3.2 実験手順

本項では実験手順を示す．前提条件としては 4.2.1 項で紹介した前提条件を用いる．HMAC-SHA-256 回路はランダムな秘密鍵を持つものとし，動作中の各サイクルごとの SHA-256 のレジスタ値 ($a \sim h$) を全て観測し，スキャンデータとした．つまり，HMAC-SHA-256 回路のスキャンチェイン長は 256 ビットとなる．ランダムに 0 ビットから 3840 ビットのビット列を生成しスキャンデータ中に加え，スキャンデータ長は 256 ビットから 4096 ビットとした．これらランダムデータは，HMAC-SHA-256 回路の外部に設置された周辺回路のスキャンデータに相当する．また，SHA-256 回路が動作する間に，2 から 100 サイクル分のランダムデータを挿入し，図 4.1 のようなスキャンデータの構成とした.

本実験では，1 つのスキャンデータ長に対して，50 種類の秘密鍵を HMAC-SHA-256 回路にランダムに設定し，提案するスキャンベース攻撃手法で，秘密鍵を復元した.

4.3.3 実験結果

各スキャンチェイン長で秘密鍵を復元した．提案するスキャンベース攻撃手法で，秘密鍵の復元に成功した．つまり，スキャンチェイン上に複数の回路のレジスタが存在していても攻撃が可能である．スキャンチェイン長を変化させた時，秘密鍵を復元させるのにかかった時間を図 4.3 に示す．図 4.3 ではそれぞれのスキャンチェイン長で秘密鍵を復元するために必要な時間の最大時間，平均時間，中央値，

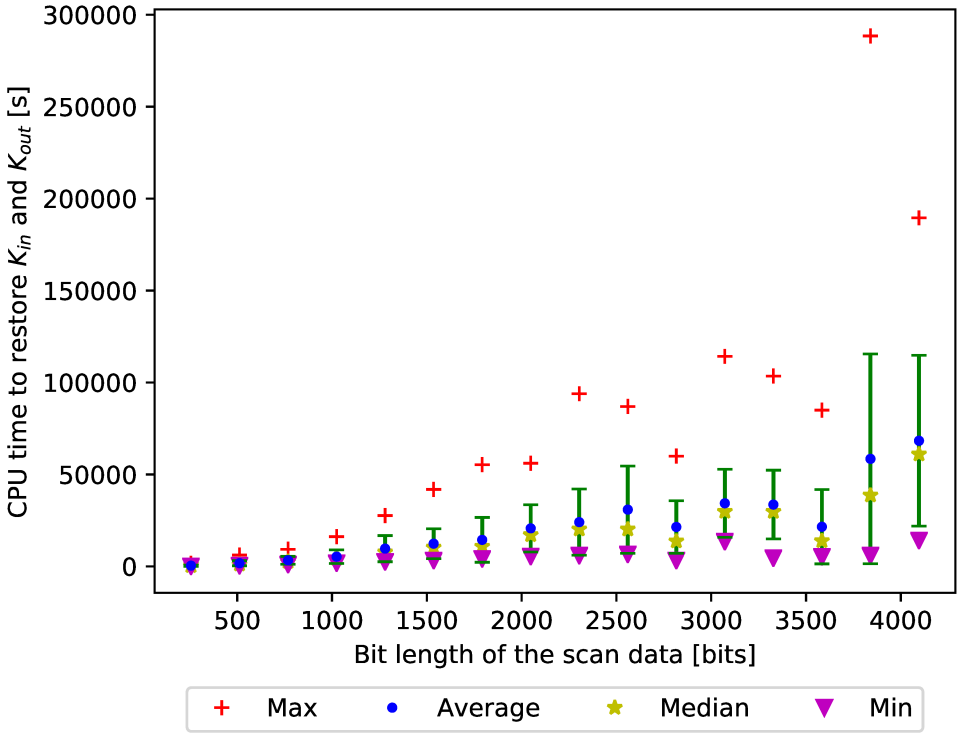


図 4.3: 連続してハッシュ値を生成しない回路の秘密鍵 K_{in}, K_{out} を復元する時間.

最小値を示している．スキャンチェイン長が 4096 ビットであっても秘密鍵を復元できることを確認した．

秘密鍵を復元させるのに必要なメッセージ数を図 4.4 に示す．図 4.4 ではそれぞれのスキャンチェイン長で秘密鍵を復元するために必要なメッセージ数の最大時間，平均時間，中央値，最小値を示している．スキャンチェイン長が 3840 ビットの時，入力する平分を 81 個使い，最大で 80 時間程度で攻撃が可能であることを示した．

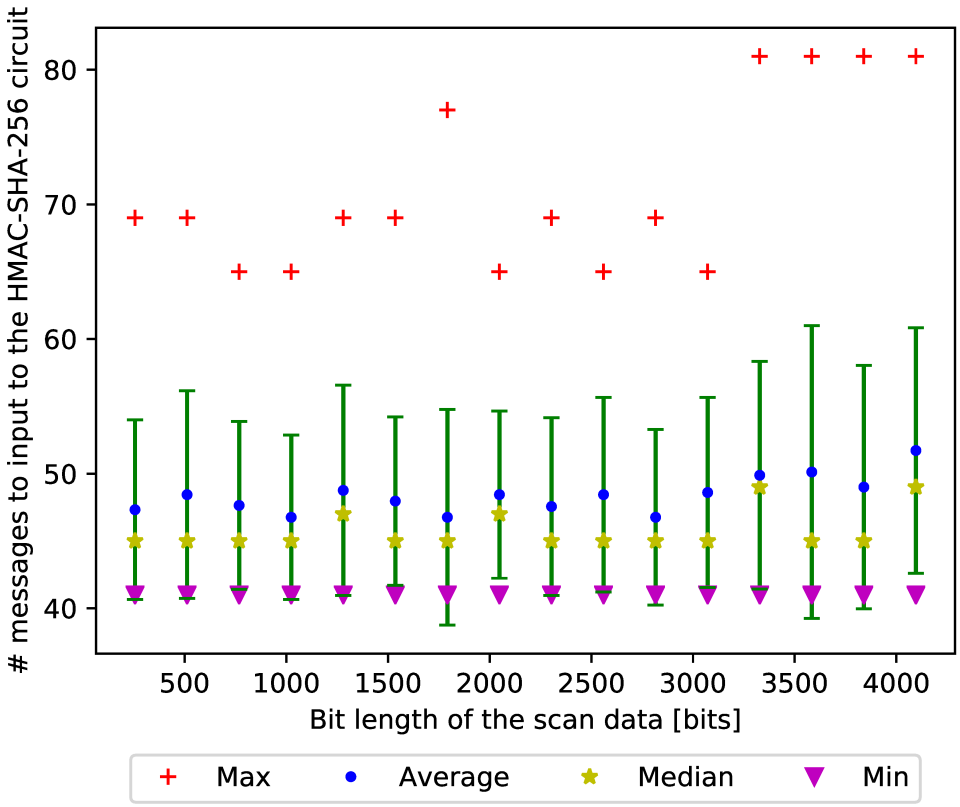


図 4.4: 連続してハッシュ値を生成しない回路の秘密鍵 $K_{\mathrm{in}}, K_{\mathrm{out}}$ を復元するために必要な時間メッセージ数.

4.4 本章のまとめ

本章では，連続動作しないハッシュ回路に対するスキャンベース攻撃手法を提案した．連続動作しないハッシュ回路として，メッセージ認証符号である HMAC を対象とする．本章で攻撃対象とするハッシュ回路は SHA-256 である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成しない回路である．攻撃の前提条件を示し，連続動作しない HMAC-SHA-256 回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定, SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

以下に本章の構成を示す．

4.2 節「連続動作しない HMAC-SHA-256 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，HMAC-SHA-256 回路に対するスキャンベース攻撃を提案した．攻撃対象となる HMAC-SHA-256 回路は連続してハッシュ値を生成する回路である．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．

4.3 節「HMAC-SHA-256 回路に対するスキャンベース攻撃の評価実験」では，連続動作しない HMAC-SHA-256 ハッシュ回路に対するスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示した．対象アーキテクチャは 4.2 節で説明した連続動作しない HMAC-SHA-256 回路である．提案手法を用いてスキャンデータから内部レジスタの対応を求め，連続してハッシュ値を生成する HMAC-SHA-256 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 ハッシュ回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結

果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．スキャンチェイン長が 4096 ビットであっても秘密鍵を復元できることを確認した．スキャンチェイン長が 3840 ビットの時，入力する平分を 81 個使い，最大で 80 時間程度で攻撃が可能であることを示した．

第 5 章

ブロック暗号に対するスキャンベース攻撃

5.1 本章の概要

本章[*1]では，ブロック暗号に対するスキャンベース攻撃を提案する．本章で攻撃対象とするブロック暗号は CLEFIA である．CLEFIA は軽量暗号の国際標準規格 ISO/IEC 29192 に採択されているアルゴリズムであり，AES と互換性がある鍵長，ブロック長を持つ．アルゴリズムをそのまま回路へ実装するだけではなく，より軽量に回路へ実装する手法も提案されており，実装方法における脆弱性を調査するため選定した．平文を暗号化するブロック長は 128 ビットであり，鍵長は 128 ビット，192 ビット，256 ビットの 3 種類ある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．攻撃の前提条件を示し，それぞれの CLEFIA 回路アーキテクチャに対するスキャンベース攻撃手法を提案する．提案手法は多数の入力メッセージを使った内部レジスタの特定，暗号化に使うラウンド鍵の特定の 2 ステップから構成されている．回路から得られるスキャンデータと CLEFIA 回路内のレジスタの対応関係を求め，秘密鍵を復元する．提案手法を使った計算機評価実験の結果，スキャンチェイン上に

[*1] 本章は ⟨11⟩, ⟨15⟩ で発表した内容による．

CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

以下に本章の構成を示す．

5.2 節「軽量ブロック暗号 CLEFIA」では，軽量暗号 CLEFIA のアルゴリズムを説明する．CLEFIA は ISO/IEC 29192 軽量暗号の国際標準規格に採択されているブロック暗号アルゴリズムである．暗号化ブロック長は 128 ビット，秘密鍵の鍵長は 128 ビット，192 ビット，256 ビットがある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．

5.3 節「CLEFIA 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，CLEFIA の 3 種類のアーキテクチャに対するスキャンベース攻撃手法を提案する．鍵スケジュール部とデータ処理部を共有している回路に対するスキャンベース攻撃では，CLEFIA 回路の内部状態を復元することで秘密鍵を復元する．鍵スケジュール部とデータ処理部を共有していない回路に対するスキャンベース攻撃では，CLEFIA 回路の内部状態を復元し，ラウンド鍵を特定することで秘密鍵を復元する．スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していても攻撃が可能な手法である．

5.4 節「CLEFIA 回路に対するスキャンベース攻撃の評価実験」では，CLEFIA 回路に対してスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示す．対象アーキテクチャは 5.3 節で説明した 3 種類のアーキテクチャである．提案手法を用いてスキャンデータから内部レジスタの対応を求め，CLEFIA 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，CLEFIA 回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結果，スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

5.5 節「本章のまとめ」では，本章の内容をまとめる．

5.2 軽量ブロック暗号 CLEFIA

本節では軽量暗号 CLEFIA のアルゴリズムと攻撃の対象とするアーキテクチャを説明する．

5.2.1 CLEFIA のアルゴリズム

本章では軽量暗号 CLEFIA のアルゴリズム [6] を説明する．CLEFIA は ISO/IEC 29192 軽量暗号の国際標準規格に採択されているブロック暗号アルゴリズムである．暗号化ブロック長は 128 ビット，秘密鍵の鍵長は 128 ビット，192 ビット，256 ビットがある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部で構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部ではホワイトニング鍵とラウンド鍵を使い，平文を暗号化する．ラウンド鍵は中間鍵から生成される．CLEFIA では鍵長によって鍵スケジュール部のアルゴリズムが異なり，データ処理部のアルゴリズムではラウンド数が異なる．

データ処理部

データ処理部ではホワイトニング鍵とラウンド鍵を使って，128 ビットの平文 PT から 128 ビットの暗号文 CT を出力する．CLEFIA の暗号アルゴリズムは 4 系列一般化 Feistel 構造をしており，1 ラウンドで 2 つの F 関数 F_0，F_1 を用いる．CLEFIA のデータ処理部の構造を図 5.1 に示す．P_i $(0 \leq i \leq 3)$ は 128 ビットの平文 PT を 4 分割した 32 ビット部分平文である．C_i $(0 \leq i \leq 3)$ は 128 ビットの暗号文 CT を 4 分割した 32 ビット部分暗号文である．32 ビットのホワイトニング鍵 WK_i $(0 \leq i \leq 3)$ と 32 ビットのラウンド鍵 RK_j を使い平文を暗号化する．ラウンド鍵の個数 j は鍵長 128 ビットでは $j = 36$，鍵長 192 ビットでは $j = 44$，鍵長 256 ビットでは $j = 52$ である．

F 関数 F_0，F_1 の構造を図 5.2 に示す．F 関数は 2 つの 32 ビットの入力 P_i，RK_j から 1 つの 32 ビットの出力を得る関数である．rk_i $(0 \leq i \leq 3)$ はそれぞれ 8 ビットであり，32 ビットのラウンド鍵 RK_j を 4 分割したものである．p_i $(0 \leq i \leq 3)$ はそれぞれ 8 ビットであり，演算途中の 32 ビットの部分平文を 4 分割したものである．F 関数の演算内容としては 2 種類の入力 P_i，RK_j の排他的論理和，2 種類の S-box S_0，S_1 による変換，1 つの拡散行列 M_0 もしくは M_1 との乗算の 3 つで構成されている．2 種類の S-box S_0，S_1 はそれぞれ 8 ビットの入

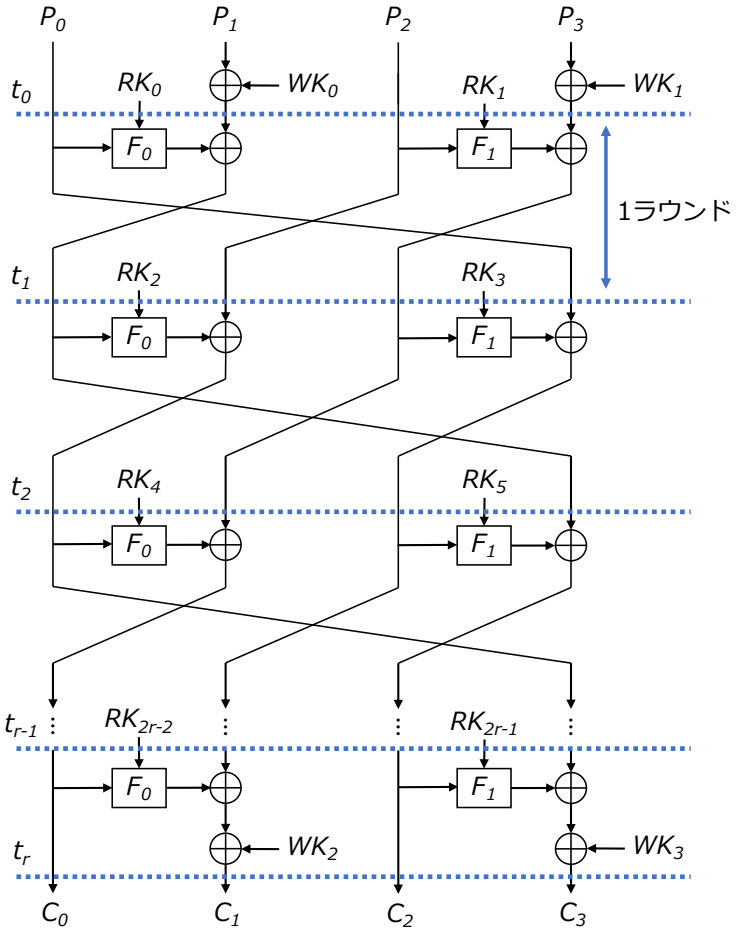
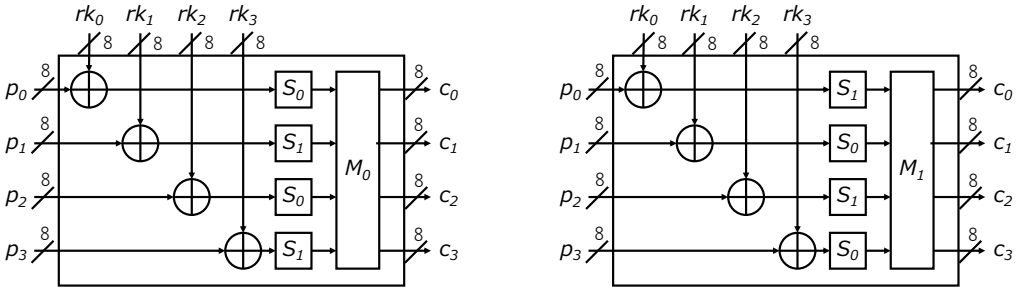


図 5.1: CLEFIA の 4 系列一般化 Feistel 構造 [6].



(a) F_0 の概略.　　(b) F_1 の概略.

図 5.2: F 関数の概略 [6].

力 w に対し，対応した 8 ビットの出力 $S_0(w)$，$S_1(w)$ を返す．2 つの行列 M_0 と

Algorithm 4 ラウンド鍵 RK_j の生成（128 ビット）

for $j = 0$ to 8 **do**
  $CON \leftarrow B^{128}_{24+4j}|B^{128}_{24+4j+1}|B^{128}_{24+4j+2}|B^{128}_{24+4j+3}$;
  $T \leftarrow L \oplus CON$;
  $L \leftarrow \Sigma(L)$;
  if $j \bmod 2 = 1$ **then**
    $T \leftarrow T \oplus K$;
  end if
  $RK_{4j}|RK_{4j+1}|RK_{4j+2}|RK_{4j+3} \leftarrow T$;
end for

Algorithm 5 ラウンド鍵 RK_j の生成（192，256 ビット）

for $j = 0$ to 10 ($k = 192$) or 12 ($k = 256$) **do**
  $CON \leftarrow B^{k}_{40+4j}|B^{k}_{40+4j+1}|B^{k}_{40+4j+2}|B^{k}_{40+4j+3}$;
  if $j \bmod 4 = 0$ or 1 **then**
    $T \leftarrow L_L \oplus CON$;
    $L_L \leftarrow \Sigma(L_L)$;
    if $j \bmod 2 = 1$ **then**
      $T \leftarrow T \oplus K_R$;
    end if
  else
    $T \leftarrow L_R \oplus CON$;
    $L_R \leftarrow \Sigma(L_R)$;
    if $j \bmod 2 = 1$ **then**
      $T \leftarrow T \oplus K_L$;
    end if
  end if
  $RK_{4j}|RK_{4j+1}|RK_{4j+2}|RK_{4j+3} \leftarrow T$;
end for

M_1 を以下に示す．行列の各要素は 16 進数表現である．

$$M_0 = \begin{pmatrix} 01 & 02 & 04 & 06 \\ 02 & 01 & 06 & 04 \\ 04 & 06 & 01 & 02 \\ 06 & 04 & 02 & 01 \end{pmatrix}, \; M_1 = \begin{pmatrix} 01 & 08 & 02 & 0A \\ 08 & 01 & 0A & 02 \\ 02 & 0A & 01 & 08 \\ 0A & 02 & 08 & 01 \end{pmatrix}$$

加算は排他的論理和として計算され，乗算は辞書的順序で最初となる原始多項式 $z^8+z^4+z^3+z^2+1=0$ で定義されている GF (2^8) 上の演算として計算される．

ラウンド鍵は中間鍵（L もしくは L_L, L_R）から生成される．鍵長 128 ビットのラウンド鍵 RK_j の生成アルゴリズムを Algorithm 4 に示す．鍵長 192 ビット，256 ビットのラウンド鍵 RK_j の生成アルゴリズムを Algorithm 5 に示す．B^k は鍵長（$k=128,192,256$）ごとに決められた定数である．Algorithm 4，Algorithm 5 中の関数 Σ を式 5.1 に示す．

$$\Sigma(x)=x_{7:63}|x_{0:6}|x_{121:127}|x_{64:120} \tag{5.1}$$

$x_{b:c}$ は変数 x の b ビット目から c ビット目を表す．関数 Σ を使い中間鍵 (L もしくは L_L，L_R) を更新しながらラウンド鍵を生成する．

鍵スケジュール部

鍵スケジュール部では秘密鍵を入力とし，暗号化で使用する 32 ビットのホワイトニング鍵と中間鍵を生成する．鍵スケジュール部はホワイトニング鍵生成フェーズと中間鍵生成フェーズの 2 つのフェーズがある．

ホワイトニング鍵生成フェーズでは鍵長 128 ビットの場合，秘密鍵 K^{128} を 32 ビット毎に 4 分割する．

$$WK_1|WK_2|WK_3|WK_4=K_0|K_1|K_2|K_3=K^{128}$$

鍵長 192 ビットの場合，秘密鍵 K^{192} を 32 ビット毎に 6 分割し，以下の式で生成する．

$$\begin{aligned}&K_0|K_1|K_2|K_3|K_4|K_5=K^{192}\\&K_L\leftarrow K_0|K_1|K_2|K_3,\ K_R\leftarrow K_4|K_5|\overline{K_0}|\overline{K_1}\\&WK_1|WK_2|WK_3|WK_4=K_L\oplus K_R\end{aligned}$$

鍵長 256 ビットの場合，秘密鍵 K^{256} を 32 ビット毎に 8 分割し，以下の式で生成する．

$$\begin{aligned}&K_0|K_1|K_2|K_3|K_4|K_5|K_6|K_7=K^{256}\\&K_L\leftarrow K_0|K_1|K_2|K_3,\ K_R\leftarrow K_4|K_5|K_6|K_7\\&WK_1|WK_2|WK_3|WK_4=K_L\oplus K_R\end{aligned}$$

中間鍵生成フェーズでは鍵長 128 ビットの場合，図 5.1 のような 4 系列一般化 Feistel 構造を使い 128 ビットの中間鍵 L を生成する．ラウンド数は $r=12$ である．ラウンド鍵として 32 ビットの定数 B_a^{128} $(0\le a\le 23)$ を入力する．鍵長 192

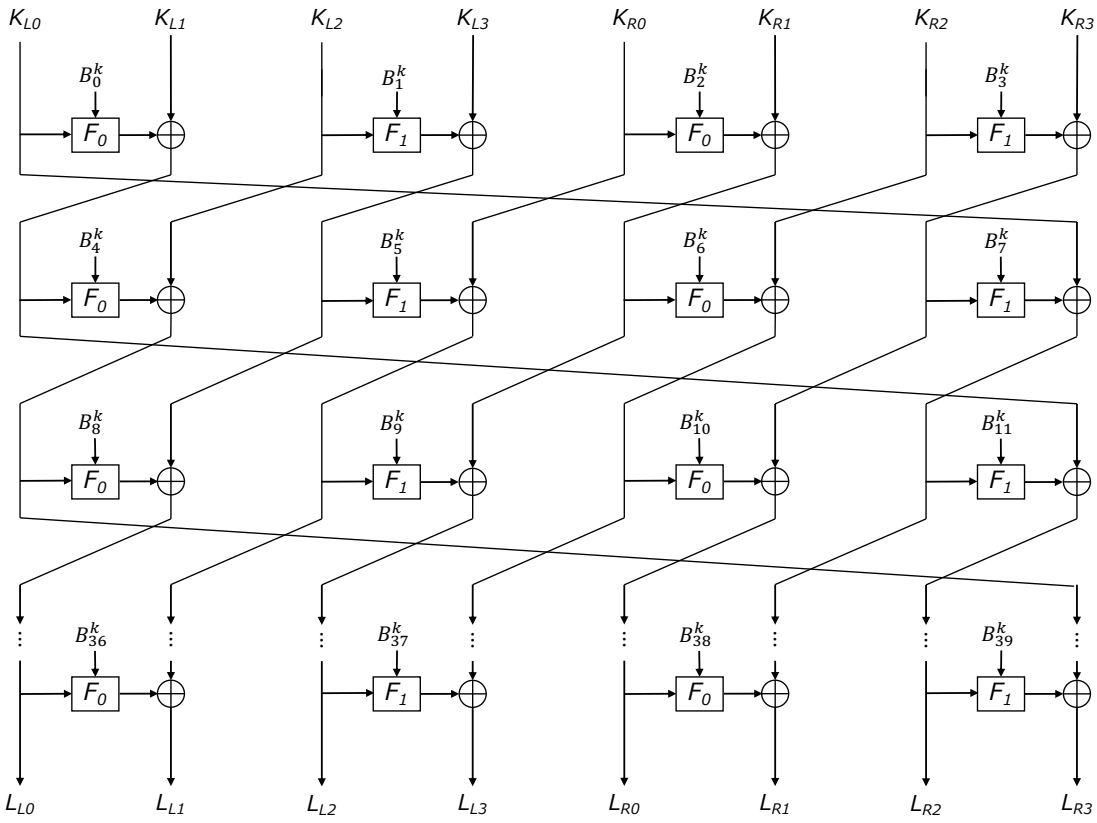


図 5.3: CLEFIA の 8 系列一般化 Feistel 構造 [6].

ビットと 256 ビットの場合，8 系列一般化 Feistel 構造を使い 128 ビットの 2 つの中間鍵 L_L，L_R を生成する．ラウンド数は $r = 10$ である．8 系列一般化 Feistel 構造を図 5.3 に示す．ラウンド鍵として 32 ビットの定数 B_a^{192}，B_a^{256} $(0 \leq a \leq 39)$ を入力する．

5.2.2 攻撃の対象となるアーキテクチャ

攻撃の対象となるアーキテクチャは 3 種類ある．

アーキテクチャ 1

アーキテクチャ 1 は図 5.1 のデータパスを実現する回路である．回路として，データ処理部と鍵スケジュール部を共有している．鍵長は 128 ビットで固定である．図 5.1 の破線のタイミングで値がレジスタに保存される．データパスとスキャンデータの関係を図 5.4 に示す．図 5.4 ではデータ処理部のスキャンデータを示している．データ処理部の最初の破線で保存されるレジスタ値をスキャンチェインから得ることができる．図 5.4 では複数のスキャンデータを縦に並べている．

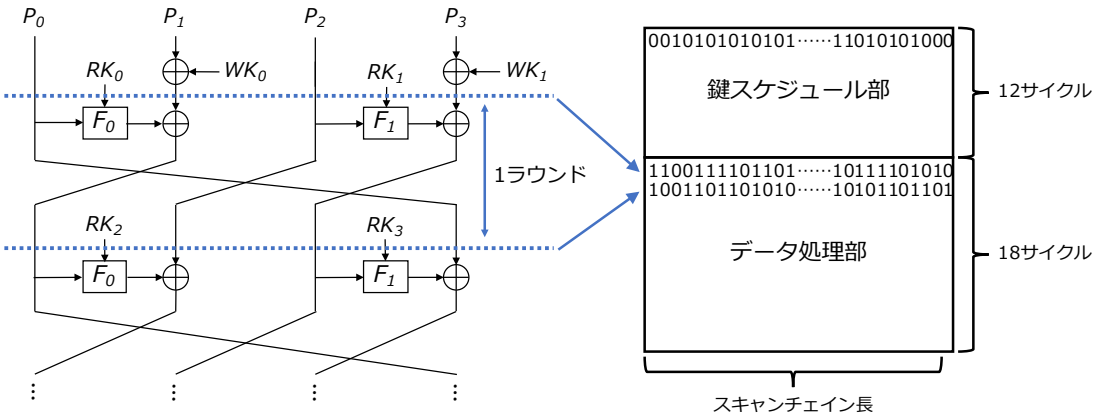


図 5.4: データパスとスキャンデータの関係.

	Reg_0	Reg_1	Reg_2	Reg_3
ラウンド0	P_0	$P_1 \oplus WK_0$	P_2	$P_3 \oplus WK_1$
ラウンド1	$P_1 \oplus WK_0 \oplus F_0$	P_2	$P_3 \oplus WK_1 \oplus F_1$	P_0
ラウンド2	$P_2 \oplus F_0$	$P_3 \oplus WK_0 \oplus F_1$	$P_0 \oplus F_1$	$P_1 \oplus WK_0 \oplus F_0$
	⋮	⋮	⋮	⋮

図 5.5: 図 5.1 の Feistel 構造を使った暗号化部でレジスタに保存されている値.

アーキテクチャ 1 では最初に鍵スケジュール部で秘密鍵 K から中間鍵 L を生成する．秘密鍵を 4 つに分割し入力する．ラウンド鍵を定数とし 11 サイクル実行した後（12 ラウンド後），出力される中間鍵 L が生成される．中間鍵 L が生成し終えると，データ処理部で平文を暗号化する．データ処理部では，平文 P を 4 分割し P_i $(0 \leq i \leq 3)$ とする．平文はホワイトニング鍵 WK_0, WK_1 と排他的論理和され，$(P_1 \oplus WK_0, P_3 \oplus WK_1)$ がレジスタに保存される．暗号化中は中間鍵 L を更新し，ラウンド鍵 RK_j $(0 \leq j \leq 35)$ を生成しながら暗号化する．17 サイクル後（18 ラウンド後）に暗号文を出力する．

各ラウンドにおいて，暗号化部で使われるレジスタに保存されている値を図 5.5 に示す．図 5.5 の Reg_i $(0 \leq i \leq 3)$ はそれぞれ 32 ビットのレジスタを表している．t_0 では部分平文 P_1, P_3 はホワイトニング鍵の上位部 $(WK_0|WK_1)$ と排他的論理和される．暗号化中はラウンド鍵生成で中間鍵 L を更新しながら暗号化する．鍵長に応じたラウンド数後に暗号文を出力する．

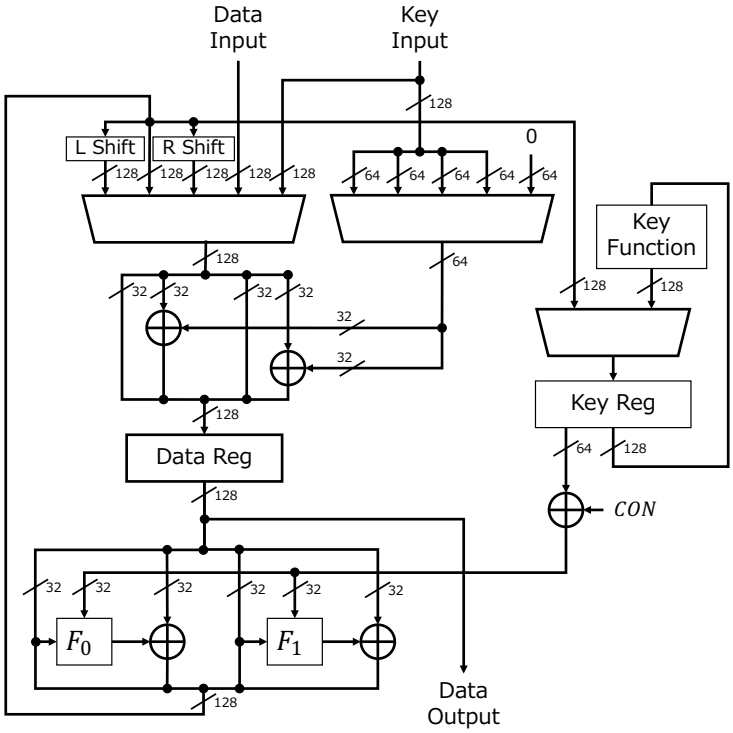


図 5.6: アーキテクチャ 2 のブロック図

アーキテクチャ 2

アーキテクチャ 2 は提案されている実装回路 [87] である．提案されている実装回路を図 5.6 に示す．回路として，データ処理部と鍵スケジュール部を共有している．鍵長は 128 ビットで固定である．実装回路は図 5.1 とは異なるデータパスで動作する．図 5.6 を実現する回路のデータパスを図 5.7 に示す．図 5.6 の回路は 1 サイクルで図 5.7 の 1 ラウンドの動作をする．K_i は 128 ビットの秘密鍵 K を 4 分割した 32 ビットの値である．RK_j^* $(0 \leq j \leq 35)$ は Algorithm4 中の $T \leftarrow T \oplus K$ を除いたアルゴリズムで生成されるラウンド鍵である．

アーキテクチャ 2 では最初に鍵スケジュール部で秘密鍵 K から中間鍵 L を生成する．図 5.6 中の Data Reg には秘密鍵を格納し，Key Reg を 0 で初期化する．定数をラウンド鍵とし，11 サイクル実行した後（12 ラウンド後），出力される中間鍵 L を Key Reg に格納する．中間鍵 L を生成し終えると，平文 P を暗号化する．平文 P は秘密鍵 K の上位部 $(K_0|K_1)$ と排他的論理和され $(P_1 \oplus K_0, P_3 \oplus K_1)$，Data Reg に格納される．暗号化中は Key Reg の中間鍵 L を更新し，ラウンド鍵 RK_j^* $(0 \leq j \leq 35)$ を生成しながら暗号化する．17 サイクル後（18 ラウンド後）に暗号文を出力する．Data Reg には図 5.7 の破線のタイミングで値が保存され

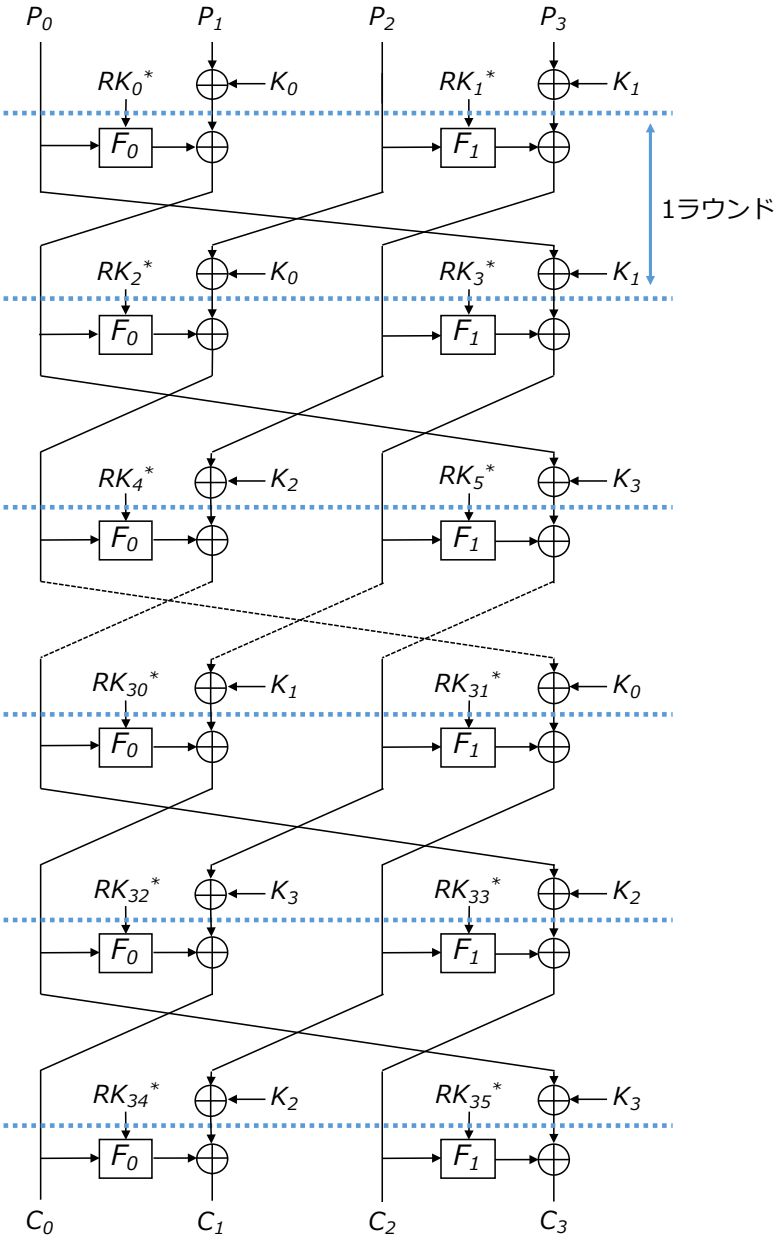


図 5.7: アーキテクチャ 2 のデータ処理部の概略

る．保存されるレジスタ値は図 5.4 と同様である．

各ラウンドにおいて，暗号化部で使われるレジスタに保存されている値を図 5.8 に示す．図 5.8 の Reg_i $(0 \leq i \leq 3)$ はそれぞれ 32 ビットのレジスタを表している．t_0 では部分平文 P_1，P_3 は秘密鍵 K_0，K_1 と排他的論理和される．暗号化中はラウンド鍵生成で中間鍵 L を更新しながら暗号化する．鍵長に応じたラウンド数後に暗号文を出力する．

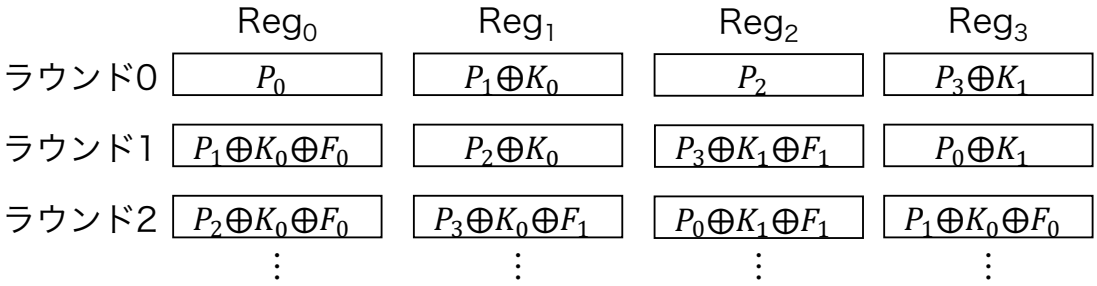


図 5.8: アーキテクチャ 2 でレジスタに保存されている値.

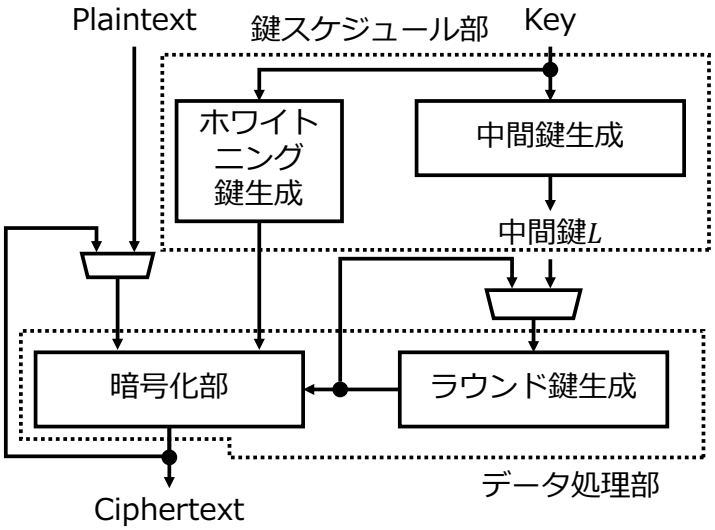


図 5.9: アーキテクチャ 3 の概略図.

アーキテクチャ 3

アーキテクチャ 3 は図 5.9 の構造を実現する回路である．図 5.9 はデータ処理部と鍵スケジュール部で構成されている．回路として，データ処理部と鍵スケジュール部を共有していない．鍵長は 128 ビット，196 ビット，256 ビットである．データ処理部と鍵スケジュール部はレジスタを含んでおり，スキャンチェインが接続されている．データ処理部は暗号化部とラウンド鍵生成で構成されており，暗号化部は図 5.1 を実現する回路である．暗号化部は図 5.1 の破線のタイミング t_n $(0 \leq n \leq r)$ で値がレジスタに保存される．

各ラウンドにおいて，暗号化部で使われるレジスタに保存されている値を図 5.5 に示す．図 5.5 の Reg_i $(0 \leq i \leq 3)$ はそれぞれ 32 ビットのレジスタを表している．t_0 では部分平文 P_1，P_3 はホワイトニング鍵の上位部 $(WK_0|WK_1)$ と排他的論理和される．暗号化中はラウンド鍵生成で中間鍵 L を更新しながら暗号化する．鍵長に応じたラウンド数後に暗号文を出力する．

鍵スケジュール部はホワイトニング鍵生成と中間鍵生成から構成されており，中間鍵生成は図 5.1 もしくは図 5.3 を実現する回路である．鍵長 128 ビットの場合，中間鍵生成の回路と暗号化部の回路は共有できる．しかし，鍵長 192 ビットと 256 ビットの場合，中間鍵生成の回路と暗号化部の回路は共有できない．

5.3 CLEFIA 回路に対するスキャンベース攻撃手法

本節では，攻撃の前提条件を説明し，CLEFIA の 3 種類のアーキテクチャに対するスキャンベース攻撃手法を提案する．

5.3.1 前提条件

本項では攻撃の前提条件を説明する．攻撃の目的は暗号文から平文を復元することである．そのために，回路の内部状態を特定し，秘密鍵を復元する．CLEFIA 回路に対するスキャンベース攻撃の前提条件を以下に示す．

攻撃者がわかること

- 暗号化のタイミング
- スキャンチェインはフルスキャン設計で，反転・動的に変化しないこと
- スキャンデータを圧縮していないこと

攻撃者ができること

- LSI に任意のメッセージを入力できる．
- 任意のタイミングで LSI のスキャンチェインにアクセスし，スキャンデータを取得できる．

攻撃者がわからないこと

- スキャンチェインに接続されているレジスタの接続順と数，種類

スキャンチェインから得られるスキャンデータはレジスタの接続順がわからないため，ランダムな値に見える．提案手法では多数の平文を入力し，得られるスキャンデータを解析する．

5.3.2 アーキテクチャ 1 に対するスキャンベース攻撃手法

5.3.1 節より，スキャンチェインに含まれるレジスタの接続順がわからないので，スキャンチェインから得られるスキャンデータ中のある 1 ビットと回路内部のある 1 ビットレジスタとの対応関係は不明である．攻撃者から見るとスキャンデータは意味のないデータ列に見える．そのため，ラウンド鍵を求める前にスキャンデータから CLEFIA 回路で使われている内部レジスタを特定する．本節では多数の平文を入力し，得られるスキャンデータを使い，スキャンデータと暗号化部の内

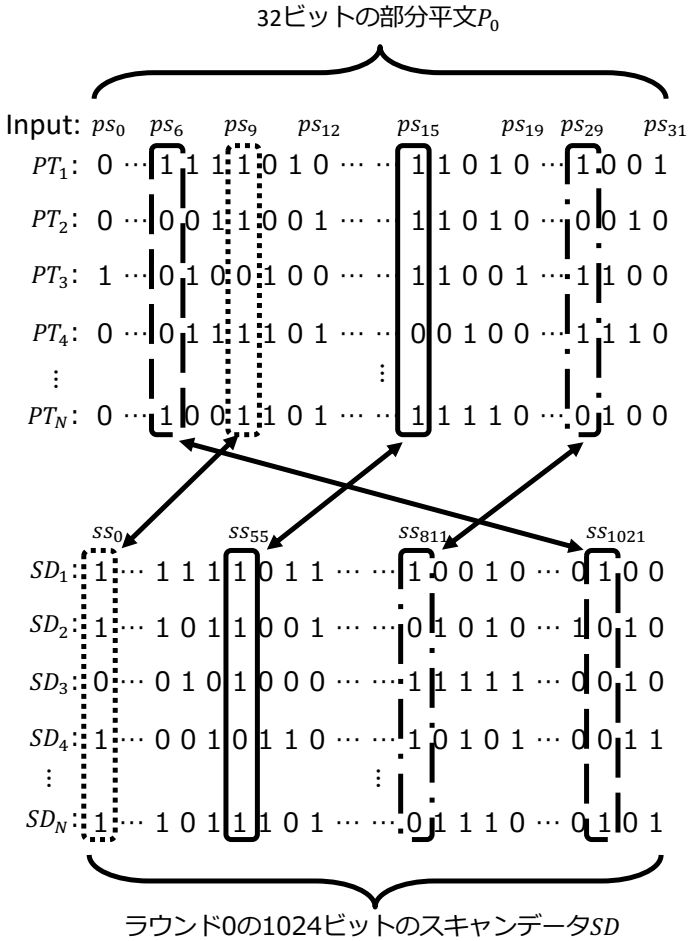


図 5.10: スキャンシグネチャを用いたビット位置の特定.

部レジスタを対応付けを説明する.

暗号化部は図 5.1 の破線のタイミング t_n $(0 \leq n \leq r)$ で値がレジスタに保存される．図 5.5 の t_0 では部分平文 P_0，P_2 が挿入されるレジスタ Reg_0，Reg_2 があり，t_1 では部分平文 P_0，P_2 が挿入されるレジスタ Reg_1，Reg_3 がある．5.3.1 節より，攻撃者は平文を自由に回路に入力できるので，t_0 で Reg_0 と Reg_2 には自由に入力できる．同様に，t_1 で Reg_1 と Reg_3 にも自由に入力できる．しかし，周辺回路は制御できない．1 つの平文だけでは周辺回路のレジスタと Reg_i $(0 \leq i \leq 3)$ の値は区別できない．そのため，多数の平文を入力し，得られるスキャンデータ中のある 1 ビットに注目する．入力した順に平文を縦に並べ，取得したスキャンデータも縦に並べる．平文とスキャンデータを縦に見ると平文とスキャンデータそれぞれで，ある 1 ビットの変化が読み取れる．ある 1 ビットの変化列をスキャンシグネチャと呼ぶ．スキャンシグネチャを使用して，得られるスキャンデータと内部レジスタを対応付けする.

平文 PT を 4 分割した 1 つの部分平文 P_0 とスキャンデータの対応付けを図 5.10 に示す．図 5.10 では例としてスキャンデータを 1024 ビットとした．図 5.10 の上部では平文 PT を N 個用意し，縦に並べる．図 5.10 の下部では平文に対応した N 個の t_0 時のスキャンデータ SD を縦に並べる．図 5.5 より，t_0 時には Reg_0 に P_0 が保存されている．そのため，平文のスキャンシグネチャ ps_l $(0 \leq l \leq 31)$ はスキャンデータのスキャンシグネチャ ss_m $(0 \leq m \leq 1023)$ 中のいずれかと一致する．図 5.10 では，平文のスキャンシグネチャをスキャンデータのスキャンシグネチャと比較した結果，$ps_6 = ss_{1021}$，$ps_9 = ss_0$，$ps_{15} = ss_{55}$，$ps_{29} = ss_{811}$ となっている．比較を繰り返すことでスキャンデータから部分平文 P_0 が求まると考えられる．つまり，スキャンデータから Reg_0 のビット位置が特定できる．同様に部分平文 P_2 を使うことで，Reg_2 のビット位置を求めることができる．Reg_1，Reg_3 のビット位置を求めるためには，部分平文 P_2 と P_0 を使い，t_1 時のスキャンデータ中を探索する．平文の数 N を多く取ることで特定できる可能性が高まる．

上記より，スキャンデータのあるビットとデータ処理部で使うレジスタのビット位置を対応付けできる．対応付けした後は鍵スケジュール部のスキャンデータを用いることで，秘密鍵を復元する．5.2.2 項より，アーキテクチャ 1 は鍵スケジュール部ではデータ処理部と同じ回路を使う．鍵スケジュール部のラウンド 0 のスキャンデータは秘密鍵の値を含んでいるため，スキャンデータの対応付けから秘密鍵の復元が可能である．

5.3.3 アーキテクチャ 2 に対するスキャンベース攻撃手法

本項では，図 5.7 を実現する回路に対するスキャンベース攻撃を提案する．アーキテクチャ 2 では平文を暗号化するデータ処理部で保存されるレジスタ値を考える．レジスタに計算途中の値が保存されるタイミングは図 5.7 の破線時である．各ラウンドでデータ処理部で使われるレジスタに保存されている値を図 5.8 に示す．図 5.8 の Reg_i $(0 \leq i \leq 3)$ はそれぞれ 32 ビットのレジスタを表し，K_i $(0 \leq i \leq 3)$ は秘密鍵を 4 分割した値を表している．5.3.2 節と同様に図 5.8 のラウンド 0 では，部分平文 P_0, P_2 の値がレジスタにそのままの形で挿入される．そのため，Reg_0, Reg_2 のビット位置は求めることができる．

Reg_1, Reg_3 のビット位置を求めるためにラウンド 0 のスキャンデータとラウンド 1 のスキャンデータの排他的論理和を取る．Reg_1 に注目すると $P_1 \oplus K_0 \oplus P_2 \oplus K_0 = P_1 \oplus P_2$ と表せる．部分秘密鍵 K_0 が消え部分平文のみで表せるため，平文毎の $P_1 \oplus P_2$ を縦に並べ，ラウンド 0 のスキャンデータとラウンド 1 のスキャンデータの排他的論理和を縦に並べる．このとき，$P_1 \oplus P_2$ のスキャンシグネチャは

Reg_1 のあるビットの変化列を表している．スキャンデータの排他的論理和を取ったスキャンシグネチャのどこかに存在するはずである．スキャンデータのスキャンシグネチャから $P_1 \oplus P_2$ のスキャンシグネチャを特定することで，Reg_1, Reg_3 のビット位置を特定できる．

$Reg_i (0 \leq i \leq 3)$ の全てのビット位置がスキャンデータから特定できるので，鍵スケジュール部のラウンド 0 のスキャンデータから秘密鍵の復元が可能である．

5.3.4　アーキテクチャ 3 に対するスキャンベース攻撃手法

本項では図 5.9 の構造を実現する鍵長 128 ビット，192 ビット，256 ビットの CLEFIA 回路に対するスキャンベース攻撃手法を提案する．データ処理部と鍵スケジュール部を共有していないアーキテクチャである．内部レジスタの特定とラウンド鍵の特定の 2 ステップで構成されている．

内部レジスタの特定

内部レジスタの特定は 5.3.2 と同様である．

ラウンド鍵の特定

秘密鍵を復元するために，データ処理部の内部レジスタの特定後にラウンド鍵を特定する．ラウンド鍵 RK_j を特定できれば，Algorithm 4，Algorithm 5 により，2 つの中間鍵と秘密鍵が復元できると考えられる．

図 5.5 中の値を使ってラウンド鍵の特定を説明する．ラウンド 1 の Reg_0 に保存されている値からラウンド鍵 RK_0 を特定する．M_0 の逆行列 M_0^{-1} とラウンド 0 の Reg_1 に保存されている値を用いて RK_0 を求める式を式 (5.2) に示す．

$$\begin{aligned} M_0^{-1}(P_1 \oplus WK_0 \oplus F_0(P_0,\, RK_0) \oplus P_1 \oplus WK_0) &= M_0^{-1}(F_0(P_0,\, RK_0)) \\ &= \begin{pmatrix} S_0(p_0 \oplus rk_0) \\ S_1(p_1 \oplus rk_1) \\ S_0(p_2 \oplus rk_2) \\ S_1(p_3 \oplus rk_3) \end{pmatrix} \end{aligned} \tag{5.2}$$

rk_i $(0 \leq i \leq 3)$ はそれぞれ 8 ビットであり，32 ビットのラウンド鍵 RK_0 を 4 分割したものである．p_i $(0 \leq i \leq 3)$ はそれぞれ 8 ビットであり，32 ビットの部分平文 P_0 を 4 分割したものである．2 種類の S-box S_0，S_1 はそれぞれ 8 ビットの入力 w に対し，対応した 8 ビットの出力 $S_0(w)$，$S_1(w)$ を返す．そのため，2 種類の S-box の逆関数である S_0^{-1}，S_1^{-1} は既知である．内部レジスタの特定よりスキャンデータから Reg_1 の値も特定できているため，$P_1 \oplus WK_0$ の値も求めるこ

とができる．ラウンド鍵 RK_j を特定するためには M_0^{-1} を使えばよい．M_0 の逆行列は M_0 自身であるから，式 5.2 の $M_0^{-1} = M_0$ とすれば，RK_0 を特定できる．また，M_1 の逆行列も M_1 自身となるので，F_1 関数に関しても式 5.2 と同様の計算ができる．

Algorithm 4 より鍵長 $k = 128$ の場合，

$$
\begin{aligned}
RK_0|RK_1|RK_2|RK_3 &\leftarrow L \oplus (B_{24}^k|B_{25}^k|B_{26}^k|B_{27}^k) \\
RK_4|RK_5|RK_6|RK_7 &\leftarrow \Sigma(L) \oplus K^k \oplus (B_{24}^k|B_{25}^k|B_{26}^k|B_{27}^k)
\end{aligned}
$$

となる．RK_0 から RK_7 まで特定すれば，中間鍵と秘密鍵を復元できる．Algorithm 5 より鍵長 $k = 192, 256$ の場合，

$$
\begin{aligned}
RK_0|RK_1|RK_2|RK_3 &\leftarrow L_L \oplus (B_{40}^k|B_{41}^k|B_{42}^k|B_{43}^k) \\
RK_4|RK_5|RK_6|RK_7 &\leftarrow \Sigma(L_L) \oplus K_R^k \\
RK_4|RK_5|RK_6|RK_7 &\leftarrow \Sigma(L_L) \oplus K_R^k \oplus (B_{44}^k|B_{45}^k|B_{46}^k|B_{47}^k) \\
RK_8|RK_9|RK_{10}|RK_{11} &\leftarrow L_R \oplus (B_{48}^k|B_{49}^k|B_{50}^k|B_{51}^k) \\
RK_{12}|RK_{13}|RK_{14}|RK_{15} &\leftarrow \Sigma(L_R) \oplus K_L^k \oplus (B_{52}^k|B_{53}^k|B_{54}^k|B_{55}^k)
\end{aligned}
$$

となる．RK_0 から RK_{15} まで特定すれば，中間鍵と秘密鍵を復元できる．中間鍵からはラウンド鍵を全て生成でき，秘密鍵からはホワイトニング鍵が生成できる．暗号文から平文の復元が可能であると考えられる．

5.4 CLEFIA 回路に対するスキャンベース攻撃の評価実験

本節では，5.3 節で提案したスキャンベース攻撃手法を使い，スキャンデータと CLEFIA 回路の内部レジスタの対応関係を求め，秘密鍵が復元を復元する実験の結果を示す．評価実験では，提案手法と CLEFIA 回路のシミュレータを python を用いて実装した．ホスト PC の OS は OS X Sierra，CPU は Intel Core i5 (2.9 GHz)，メモリは 8GB を用いた．

5.4.1 実験方法

攻撃対象は 5.2.2 項で説明した 3 種類のアーキテクチャとした．

アーキテクチャ 1 に対する実験

アーキテクチャ 1 は図 5.1 のデータパスを実現する回路である．CLEFIA 回路はランダムな秘密鍵を持つものとし，図 5.1 の破線のタイミングでレジスタに値が保存されるとしている．破線のタイミングで保存される値を CLEFIA 回路が動作中の各サイクルで得られるスキャンデータとした．CLEFIA 回路のデータ処理部で使われるレジスタのスキャンチェイン長は 256 ビットである．本実験では回路のスキャンチェイン長は 256 ビットから 2560 ビットまで 256 ビット毎変化させた．回路以外のその他周辺回路を実現するためにランダムデータを各サイクルで得られるスキャンデータに加えた．加えるランダムデータは 0 ビットから 2432 ビットまでのビット列とした．1 つのスキャンデータ長に対して，50 種類の秘密鍵を CLEFIA 回路に設定し，5.3.2 項で提案したスキャンベース攻撃を用いて秘密鍵を復元した．

アーキテクチャ 2 に対する実験

アーキテクチャ 2 は提案されている実装回路 [87] である．CLEFIA 回路はランダムな秘密鍵を持つものとし，図 5.7 の破線のタイミングでレジスタに値が保存されるとしている．破線のタイミングで保存される値を CLEFIA 回路が動作中の各サイクルで得られるスキャンデータとした．CLEFIA 回路のデータ処理部で使われるレジスタのスキャンチェイン長は 256 ビットである．本実験では回路のスキャンチェイン長は 256 ビットから 2560 ビットまで 256 ビット毎変化させた．回路以外のその他周辺回路を実現するためにランダムデータを各サイクルで得ら

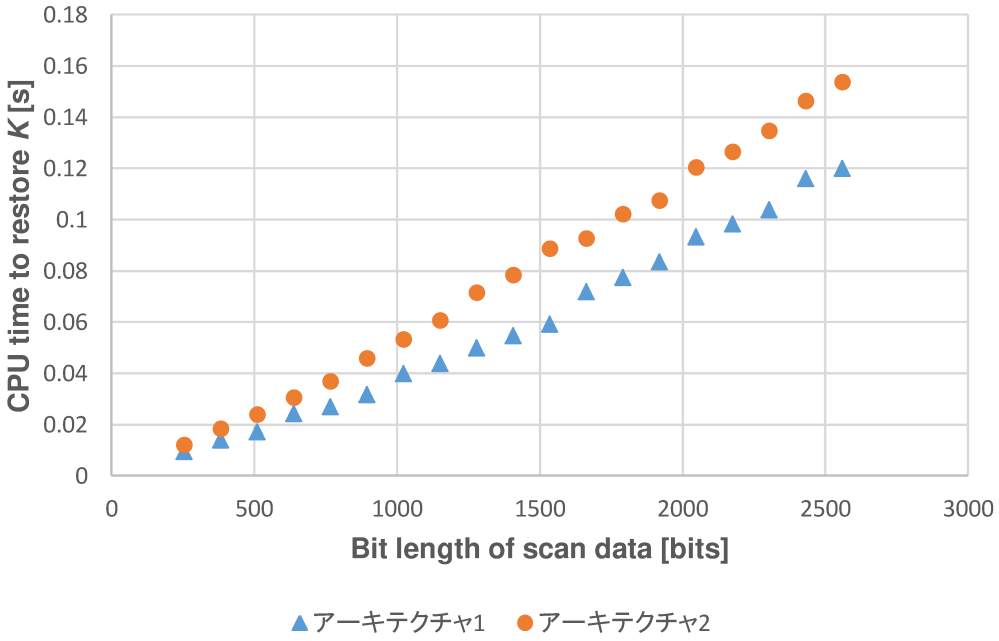


図 5.11: 各アーキテクチャで秘密鍵を復元する時間.

れるスキャンデータに加えた．加えるランダムデータは 0 ビットから 2432 ビットまでのビット列とした．1 つのスキャンデータ長に対して，50 種類の秘密鍵を CLEFIA 回路に設定し，5.3.3 項で提案したスキャンベース攻撃を用いて秘密鍵を復元した．

アーキテクチャ 3 に対する実験

アーキテクチャ 3 は図 5.9 を実現する回路である．CLEFIA 回路はランダムな秘密鍵を持つものとし，図 5.1 の破線のタイミングでレジスタに値が保存されるとしている．5.3.4 節で提案したラウンド鍵の特定する実験の結果を示す．ただし，内部レジスタの特定は終了してるものとする．

5.4.2 実験結果

アーキテクチャ 1 とアーキテクチャ 2 に対する実験結果

128 個の平文を使いアーキテクチャ 1 とアーキテクチャ 2 のスキャンチェインを 256 ビットから 2560 ビットまで 256 ビット毎変化させ，それぞれで秘密鍵を復元した．各スキャンチェイン長で 50 回秘密鍵を復元した．50 回全てで秘密鍵の復元に成功した．スキャンチェイン長を変化させたときに秘密鍵の復元にかかった時間を図 5.11 に示す．図 5.11 では 50 回の平均時間を示している．

表 5.1: 鍵長ごとの特定時間と平均時間.

	特定時間 [s]		
Trials	128 ビット	192 ビット	256 ビット
1	0.002879	0.002302	0.001798
2	0.000792	0.000962	0.001138
3	0.000747	0.001153	0.001041
4	0.000742	0.001003	0.001096
5	0.000751	0.000971	0.001049
6	0.000751	0.000991	0.001038
7	0.000736	0.000962	0.001048
8	0.001064	0.001025	0.001092
9	0.000746	0.000962	0.001047
10	0.000719	0.001482	0.002185
Average	0.000993	0.001181	0.013432

アーキテクチャ 3 に対する実験結果

内部レジスタの特定後のスキャンデータからラウンド鍵の特定に必要な時間を計測した．特定するラウンド鍵は鍵長が 128 ビットの場合，ラウンド鍵 RK_0 から RK_7 とし，鍵長 192 と 256 ビットの場合，ラウンド鍵 RK_0 から RK_{15} とした．それぞれの鍵長で 10 個の秘密鍵を用意し，10 個全てのラウンド鍵の特定に成功した．ラウンド鍵の特定に必要な時間と平均時間を表 5.1 に示す.

5.5 本章のまとめ

本章では，ブロック暗号に対するスキャンベース攻撃を提案した．本章で攻撃対象とするブロック暗号は CLEFIA である．CLEFIA は軽量暗号の国際標準規格に採択されているアルゴリズムである．平文を暗号化するブロック長は 128 ビットであり，鍵長は 128 ビット，192 ビット，256 ビットの 3 種類ある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．攻撃の前提条件を示し，それぞれの CLEFIA 回路アーキテクチャに対するスキャンベース攻撃手法を提案する．提案手法は多数の入力メッセージを使った内部レジスタの特定，暗号化に使うラウンド鍵の特定の 2 ステップから構成されている．回路から得られるスキャンデータと CLEFIA 回路内のレジスタの対応関係を求め，秘密鍵を復元する．提案手法を使った計算機評価実験の結果，スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

以下に本章の構成を示す．

5.2 節「軽量ブロック暗号 CLEFIA」では，軽量暗号 CLEFIA のアルゴリズムを説明した．CLEFIA は ISO/IEC 29192 軽量暗号の国際標準規格に採択されているブロック暗号アルゴリズムである．暗号化ブロック長は 128 ビット，秘密鍵の鍵長は 128 ビット，192 ビット，256 ビットがある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．

5.3 節「CLEFIA 回路に対するスキャンベース攻撃手法」では，攻撃の前提条件を説明し，CLEFIA の 3 種類のアーキテクチャに対するスキャンベース攻撃手法を提案した．鍵スケジュール部とデータ処理部を共有している回路に対するス

キャンベース攻撃では，CLEFIA 回路の内部状態を復元することで秘密鍵を復元する．鍵スケジュール部とデータ処理部を共有していない回路に対するスキャンベース攻撃では，CLEFIA 回路の内部状態を復元し，ラウンド鍵を特定することで秘密鍵を復元する．スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していても攻撃が可能な手法である．

5.4 節「CLEFIA 回路に対するスキャンベース攻撃の評価実験」では，CLEFIA 回路に対してスキャンベース攻撃手法を用いて計算機上で評価実験した結果を示した．対象アーキテクチャは 5.3 節で説明した 3 種類のアーキテクチャである．提案手法を用いてスキャンデータから内部レジスタの対応を求め，CLEFIA 回路の秘密鍵を復元する．実験では，提案手法を python を用いて実装し，CLEFIA 回路シミュレータを python を用いて実装した．ランダムなデータを付与したスキャンデータに対して提案手法を適応した結果，スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

第 6 章

結論

本論文では，実装されているストリーム暗号回路に対するスキャンベース攻撃の実験，ハッシュ回路とブロック暗号回路に対するスキャンベース攻撃手法を提案し，計算機評価実験の結果を示す．

以下に本論文の構成を示す．

第 2 章「ストリーム暗号に対するスキャンベース攻撃の実装実験」では，ストリーム暗号を実装した回路に対するスキャンベース攻撃の実装実験の結果を示した．本章で攻撃対象とするストリーム暗号は Trivium である．Trivium 回路の内部構造は AND 演算，OR 演算，シフト演算で構成されているため高速に動作する．Trivium のアルゴリズムは初期化フェーズとキーストリーム生成フェーズの 2 つのフェーズからなる．初期化フェーズで内部レジスタを初期化し，キーストリーム生成フェーズでキーストリームを生成する．生成したキーストリームと平文で排他的論理和取ることで暗号化する．今実験で使用するハードウェアアーキテクチャについて説明した．さらに，サイドチャネル攻撃評価標準ボードである SASEBO-GII [76] を用いて，実装した Trivium 回路に対してスキャンベース攻撃する実験の結果をした．本実験で使用する Trivium 回路はハードウェア記述言語 Verilog を用いて実装した．スキャンベース攻撃手法として藤代らの手法 [36] を用いた．提案手法はスキャンチェイン上に Trivium 回路以外のレジスタが存在しても攻撃が可能な手法である．評価ボード上のレジスタを観測し，Trivium 回路の内部レジスタを特定できるか実験した結果，スキャンデータを取得できることを確認した．また，得られたスキャンデータを提案手法を用いて解析した結果，内部レジスタの対応付けに成功した．暗号化処理開始からデータ解析終了までの時間は 30 秒から 70 秒の間であった．

第 3 章「連続動作するハッシュ回路に対するスキャンベース攻撃」では，連続動作するハッシュ回路に対するスキャンベース攻撃手法を提案した．連続動作す

るハッシュ回路として，メッセージ認証符号である HMAC-SHA-256 を生成する回路を対象とする．HMAC（Hash-based Message Authentication Code）は反復暗号ハッシュ関数を用いたメッセージ認証コードである．ハッシュ関数を複数回適用するもので，IPsec と SSL/TLS で採用されている．メッセージと秘密鍵をハッシュ関数に与えることで生成されるハッシュ値を認証に使用する．本章で攻撃対象とするハッシュ回路は SHA-256 回路である．HMAC のハッシュ関数に SHA-256 を使ったものを HMAC-SHA-256 という．SHA-256 は NIST によって標準化されたハッシュ関数であり，CRYPTREC で電子政府推奨暗号とされている．256 ビットのハッシュ値を出力するハッシュ関数である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成する回路である．攻撃の前提条件を示し，連続動作する HMAC-SHA-256 生成回路へのスキャンベース攻撃手法を提案した．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 3 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

第 4 章「連続動作しないハッシュ回路に対するスキャンベース攻撃」では，連続動作しないハッシュ回路に対するスキャンベース攻撃手法を提案した．連続動作しないハッシュ回路として，メッセージ認証符号である HMAC を対象とする．本章で攻撃対象とするハッシュ回路は SHA-256 である．攻撃対象とする HMAC-SHA-256 回路のアーキテクチャは連続してハッシュ値を生成しない回路である．攻撃の前提条件を示し，連続動作しない HMAC-SHA-256 回路へのスキャンベース攻撃手法を提案する．提案手法は入力メッセージから得られるスキャンデータから遷移グループの特定，SHA-256 回路動作の初期位置の特定，ビット位置の特定，前半レジスタと後半レジスタの特定の 4 ステップから構成されている．回路から得られるスキャンデータと HMAC-SHA-256 回路内のレジスタの対応関係を求め，秘密鍵を復元する．スキャンチェイン上に HMAC-SHA-256 回路以外のレジスタが存在していても攻撃が可能な手法である．提案手法を使った計算機評価実験の結果，スキャンチェイン上に SHA-256 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，HMAC-SHA-256 回路で用いる秘密鍵を復元できることを確認した．

第 5 章「ブロック暗号に対するスキャンベース攻撃」では，ブロック暗号に対するスキャンベース攻撃を提案した．本章で攻撃対象とするブロック暗号は CLEFIA である．CLEFIA は軽量暗号の国際標準規格に採択されているアルゴリズムである．平文を暗号化するブロック長は 128 ビットであり，鍵長は 128 ビット，192 ビット，256 ビットの 3 種類ある．CLEFIA のアルゴリズムは鍵スケジュール部とデータ処理部から構成されている．鍵スケジュール部では暗号化に使うホワイトニング鍵と中間鍵を生成する．データ処理部では暗号化に使うラウンド鍵を生成しながら平文を暗号化する．攻撃対象とする CLEFIA 回路のアーキテクチャは鍵スケジュール部とデータ処理部を共有している 2 つの回路（鍵長は 128 ビット）と鍵スケジュール部とデータ処理部を共有していない回路（鍵長は 128 ビット，192 ビット，256 ビット）の 3 つのアーキテクチャである．攻撃の前提条件を示し，それぞれの CLEFIA 回路アーキテクチャに対するスキャンベース攻撃手法を提案する．提案手法は多数の入力メッセージを使った内部レジスタの特定，暗号化に使うラウンド鍵の特定の 2 ステップから構成されている．回路から得られるスキャンデータと CLEFIA 回路内のレジスタの対応関係を求め，秘密鍵を復元する．提案手法を使った計算機評価実験の結果，スキャンチェイン上に CLEFIA 回路以外のレジスタが存在していてもスキャンデータと内部レジスタの対応付けに成功し，3 つのアーキテクチャそれぞれで秘密鍵の復元に成功した．

今後の課題として，秘密情報を安全に保護できる集積回路を設計するために対応可能な暗号アルゴリズムの拡張と対応可能な実装モデルの拡張がある．提案してきたスキャンベース攻撃手法を拡張することで，各暗号方式（ブロック暗号，ストリーム暗号，公開鍵暗号，ハッシュ関数）に対するスキャンベース攻撃手法を提案する．それぞれの暗号方式に対して秘密鍵を求め平文の復元することで，暗号方式の脆弱性をさらに解明する．これまでのスキャンチェインの構造に依存しないスキャンベース攻撃手法を拡張することで，異なる実装方法の集積回路に対するスキャンベース攻撃手法を提案する．異なる実装方法の例としては，スキャンチェインの出力データを圧縮，マスク，動的に変化させる等がある．実装の構造，実装方法の数理的特性を調査し，スキャンベース攻撃手法を提案することで，実装方法の脆弱性をさらに解明する．

また，防御手法の脆弱性の評価と実装方法の脆弱性の評価が必要である．各防御手法に対する評価と実装方法に関しても今後の課題とします．

これらの研究では大量なデータから秘密鍵を見つけ出す．そのために，従来の計算機だけでなく新しい計算機の活用も重要な課題である．特に，組合せ最適化問題に特化した計算機として量子アニーリングマシンがある．新たな計算機（量子アニーリングマシン）の活用も今後の課題とする．

謝辞

本研究は，筆者が早稲田大学大学院 基幹理工学研究科情報理工・情報通信専攻 博士後期課程在学中に同大学院 基幹理工学研究科 戸川望教授の指導のもとに行ったものである．

本論文の執筆にあたり，5 年半に亘り，日々の研究活動ならびにゼミ等において，多大なる御指導，御助言を頂きました戸川望教授に深く感謝いたします．日々，多くを学び成長することができました．また，修士課程の早期終了に至り大変感謝いたします．同じく日頃から研究活動ならびにゼミ等において，的確な御意見や御指摘を頂きました柳澤政生教授に深く御礼申し上げます．研究者としての姿勢を学ぶことができ，研究の質を高めることができました．また，多くの御指摘を頂き，本論文をより良い内容にできました．同じくお忙しい中，多岐に亘る御指摘を頂いた森達哉教授に深く御礼申し上げます．御指摘のおかげで，本論文の質を高めることができました．また，学部 4 年生の半年間に亘り，御指導いただき成長できました．同じく戸川望教授と柳澤政生教授と共に，ゼミの場で研究に関する多くの御意見，御指摘を頂きました史又華教授に深く感謝いたします．研究の質を高めることができました．

筆者が研究室に入ってから長きに亘り，御指導を頂きました多和田雅師講師と川村一志特任助教（東京工業大学）に深く感謝いたします．論文の書き方から研究の議論まで多くの御指導を頂き，成長することができました．また，研究活動全般にわたり，御支援して頂いた戸川研究室秘書の渡部周子氏に深く感謝いたします．鮑思雅講師，大屋優講師，寺田晃太朗氏（ヤフー株式会社）には日々の研究生活において多くを学ぶことができました．御礼申し上げます．量子アニーリングの研究において多くの御助言，御指導を頂きました田中宗准教授（慶應義塾大学）と白井達彦講師にも御礼申し上げます．学部時代より研究を御指導頂き，研究に関する鋭い御意見，御助言をくださいました藤代美佳氏（株式会社東芝）と蒋慧倩氏（日本電気株式会社）に御礼申し上げます．お二人がいなければ，ここまで成長できなかったと思います．研究室同期の岩名地良太氏（株式会社デンソー），河野圭亮氏

（パナソニック株式会社），柿沼博幸氏（パナソニック株式会社），長谷川健人氏（株式会社 KDDI 総合研究所），真鍋知樹氏（PwC コンサルティング合同会社），村松和侑氏（株式会社ＪＲ東日本情報システム），矢野椋也氏（日本電気株式会社）には，学部 4 年生のときから大変お世話になり，大いに感謝致します．ご支援のおかげで研究を高めることができました．

最後に研究および生活全般にわたり支援して頂きました戸川研究室および情報システム研究室の皆さまに深く感謝いたします．

参考文献

[1] 特定非営利活動法人 日本ネットワークセキュリティ協会, “２０１３年 情報セキュリティインシデントに関する調査報告書 個人情報漏えい編 ,” http://www.jnsa.org/result/incident/data/2013incident_survey_ver1.2.pdf.

[2] “FIPS 46-3, Data Encryption Standard (DES),” http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[3] “Advanced encryption standard (AES),” Federal Information Processing Standards Publication 197(FIPS 197), National institute of standards and technology (NIST), Tech. Rep., 2001.

[4] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, “Camellia: a 128-bit block cipher suitable for multiple platforms,” Nippon Telegraph and Telephone Corporation, Mitsubishi Electric Corporation, 2001.

[5] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw, “The LED block cipher,” *Lecture Note in Computer Science*, vol. 6917, pp. 326–341, 2011.

[6] Sony Croporation, “The 128-bit blockcipher CLEFIA algorithm specification,” https://www.sony.co.jp/Products/cryptography/clefia/download/data/clefia-spec-1.0.pdf.

[7] 三菱電機株式会社, “暗号技術仕様書 MISTY1,” http://www.cryptrec.go.jp/cryptrec_03_spec_cypherlist_files/PDF/05_02jspec.pdf.

[8] C. D. Cannière and B. Preneel, “Trivium,” http://www.ecrypt.eu.org/stream/triviumpf.html.

[9] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel, “A New Keystream Generator MUGI,” *Lecture Notes in Computer Science*, vol. 2365, pp. 179–194, 2002.

[10] C. J. A. Jansen, “Stream cipher design: Make your LFSRs jump!,” in

Proc. The State of the Art of Stream Ciphers, Workshop Record, ECRYPT Network of Excellence in Cryptology, 2004, pp. 94–108.

[11] S. Babbage and M. Dodd, "MICKEY 2.0," `http://www.ecrypt.eu.org/stream/mickeypf.html`.

[12] M. Hell, T. Johansson, and W. Meier, "Grain v1," `http://www.ecrypt.eu.org/stream/grainpf.html`.

[13] KDDI 株式会社, "ストリーム暗号 KCipher-2," `http://www.cryptrec.go.jp/cryptrec_13_spec_cypherlist_files/PDF/21_00jspec.pdf`.

[14] R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21.2, pp. 120–126, 1978.

[15] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, pp. 203–209, 1987.

[16] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, " *Lecture Notes in Computer Science*, vol. 1109, pp. 104–113, 1996.

[17] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525, 1997.

[18] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," *Lecture Notes in Computer Science*, vol. 1233, pp. 37–51, 1997.

[19] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi, "Cryptanalysis of block ciphers implemented on computers with cache," *Lecture Notes in Computer Science*, vol. 2779, pp. 62–76, 2003.

[20] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. CRYPTO '99*, Springer-Verlag, 1999, pp. 388–397.

[21] K. Okeya, "Side channel attacks against HMACs based on block cipher based hash functions," *Lecture Notes in Computer Science*, vol. 4058, pp. 432–443, 2006.

[22] R. McEvoy, M. Tunstall, C. C. Murphy, and W. P. Marnane, "Differential power analysis of HMAC based on SHA-2, and countermeasures," *Lecture Notes in Computer Science*, vol. 4867, pp. 317–332, 2007.

[23] S. Belaïd, L. Bettale, E. Dottax, L. Genelle, and F.Rondepierre, "Differential power analysis of HMAC SHA-2 in the hamming weight model," in

Proc. International Conference on Security and Cryptography (SECRYPT 2013), 2013, pp. 230–241.

[24] C. H. Gebotys, B. A. White, E. Mateos, "Preaveraging and carry propagate approaches to side-channel analysis of HMAC-SHA256," *ACM Trans. on Embedded Computing Systems*, vol. 15, no. 1, pp. 4:1–4:19 (2016).

[25] B. Yang, K. Wu and R. Karriv, "Scan based side channel attack on dedicated hardware implementations of Data Encryption Standard," in *Proc. of International Test Conference*, 2004, pp. 339–344.

[26] H. Kodera, M. Yanagisawa, and N. Togawa, "Scan-based Attack against DES and Triple DES Cryptosystems Using Scan Signatures," *Journal of Information Processing*, vol. 21, no. 3, pp. 572–579, 2013.

[27] J. D. Rolt, G. D. Natale, M. Flottes, and B. Rouzeyre, "A novel differential scan attack on advanced DFT structures," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 4, pp. 58:1–58:22, 2013.

[28] B. Yang, K. Wu and R. Karri, "Secure scan: a design-for-test architecture for crypto chips," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2287–2293, 2006.

[29] R. Nara, N. Togawa, M. Yanagisawa, T.Ohtsuki, "A scan-based attack based on discriminators for AES cryptosystems," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 12, pp. 3229–3237, 2009.

[30] S. S. Ali, S. Saeed, O. Sinanoglu, and R. Karri, "Novel test-mode-only scan attack and countermeasure for vompression-based scan architebures," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 808–821, 2015.

[31] H. Jiang, M. Fujishiro, H. Kodera, M. Yanagisawa, and N. Togawa, "Scan-based side-channel attack on the camellia block cipher using scan signatures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 12, pp. 2547–2555, 2015.

[32] M. Fujishiro, M. Yanagisawa, N. Togawa, "Scan-based side-channel attack on the LED block cipher using scan signagures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E97-A. no. 12, pp. 2434–2442, 2014.

[33] R. Nara, K. Satoh, M. Yanagisawa, T. Ohtsuki, and N. Togawa, "Scan-

based side-channel attack against RSA cryptosystems using scan signatures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, no. 12, pp. 2481–2489, Dec. 2010.

[34] R. Nara, M. Yanagisawa, T. Ohtsuki and N. Togawa, "Scan vulnerability in elliptic curve cryptosystems," *IPSJ Trans. System LSI Design Methodology*, vol. 4, pp. 47–59, Feb. 2011.

[35] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, "Scan based side channel attacks on stream ciphers and their counter-measures," *Lecture Notes in Computer Science*, vol. 5365, pp. 226–238, 2008.

[36] M. Fujishiro, M. Yanagisawa, and N. Togawa, "Scan-based attack against Trivium stream cipher using scan signatures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E97-A no. 7 pp. 1444–1451, 2014.

[37] D. Mukhopadhyay, S. Banerjee, D. RoyChowdhury, and B. B. Bhattacharya, "CryptoScan: A secured scan chain architecture," in *Proc. 14th Asian Test Symposium (ATS'05)*, 2005, pp. 348–353.

[38] Y. Liu, K. Wu, and R. Karri, "Scan-based attacks on linear feedback shift register based stream ciphers," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 2, pp. 20:1–20:15, 2011.

[39] J. Da Rolt, A. Das, G. Di Natale, M. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test Versus Security: Past and Present," *IEEE Trans. Emerging Topics in Computing*, vol. 2, no. 1, pp. 50–62, 2014.

[40] A. Das, J. Da Rolt, S.Ghosh, S. Seys, S. Dupuis, G. Di Natale, M. Flottes, B. Rouzeyre, and I. Verbauwhede, "Secure JTAG Implementation Using Schnorr Protocol," *Journal of Electronic Testing,* vol. 29, no. 2, pp. 193–209, Apr. 2013.

[41] E. DeBusschere, and M. McCambridge, "Modern Game Console Exploitation," Technical Report, Department of Computer Science, University of Arizona, 2012.

[42] 穂田知治, "DES 暗号 LSI に対するスキャンベース攻撃の実装実験に関する研究," 2011 年度卒業論文, 2011

[43] 小寺博和, 柳澤政生, 戸川望, "スキャンシグネチャを利用した Triple DES に対するスキャンベース攻撃の実装評価," 信学暗号と情報セキュリティシンポジウム (SCIS2012), 3C2-4, Jan. 2012.

[44] 奈良竜太, 小寺博和, 柳澤政生, 大附辰夫, 戸川望, "SASEBO-GII を使用した

AES に対するスキャンベース攻撃の実装実験," 信学暗号と情報セキュリティシンポジウム (SCIS2011), 1D1-2, Jan. 2011.

[45] 蒋慧倩, "スキャンシグネチャーを用いた暗号 LSI に対するスキャンベース攻撃に関する研究," 2015 年度修士論文, 2015.

[46] X. Li, W. Li, J. Ye, H. Li and Y. Hu, "Scan Chain Based Attacks and Countermeasures: A Survey," *IEEE Access*, vol. 7, pp. 85055–85065, 2019.

[47] G. Sengar, D. Mukhopadhyay and D. R. Chowdhury, "Secured flipped scan-chain model for crypto-architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 11, pp. 2080–2084, Nov. 2007.

[48] S. Banik and A. Chowdhury, "Improved scan-chain based attacks and related countermeasures," in *Progress in Cryptology-INDOCRYPT*, 2013, pp. 78–97.

[49] Y. Atobe, Y. Shi, M. Yanagisawa and N. Togawa, "Dynamically changeable secure scan architecture against scan-based side channel attack," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2012, pp. 155–158.

[50] Y. Atobe, Y. Shi, M. Yanagisawa and N. Togawa, "State dependent scan flip-flop with key-based configuration against scan-based side channel attack on RSA circuit," in *Proc. IEEE Asia – Pacific Conf. Circuits Syst.*, Dec. 2012, pp. 607–610.

[51] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 4, no. 4, pp. 325–336, Nov. 2007.

[52] Y. Atobe, Y. Shi, M. Yanagisawa, and N. Togawa, "Secure scan design with dynamically configurable connection," in *Proc. IEEE 19th Pacific Rim Int. Symp. Dependable Comput.*, Dec. 2013, pp. 256–262.

[53] D. Hely, F. Bancel, M. L. Flottes, and B. Rouzeyre, "Test control for secure scan designs," in *Proc. 10th IEEE European Symposium on Test (EST'05)*, 2005, pp. 190–195.

[54] S. Banik, A. Chattopadhyay, and A. Chowdhury, "Cryptanalysis of the double-feedback XOR-chain scheme proposed in indocrypt 2013," in *Progress in Cryptology-INDOCRYPT*, Springer, 2014, pp. 179 – 196.

[55] A. Cui, Y. Luo and C.-H. Chang, "Static and dynamic obfuscations of scan data against scan-based side-channel attacks," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 363–376, Feb. 2017.

[56] F. DaSilva, "IEEE standard testability method for embedded core-based

integrated circuits," *IEEE Std 1500TM-2005*, 2011.

[57] G.-M. Chiu and J. C.-M. Li, "A secure test wrapper design against internal and boundary scan attacks for embedded cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 126–134, Jan. 2012.

[58] M. D. Silva, E. Valea, M.-L. Flottes, S. Dupuis, G. Di Natale and B. Rouzeyre, "A new secure stream cipher for scan chain encryption," in *Proc. IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, Jul. 2018, pp. 68–73.

[59] M. D. Silva, M.-L. Flottes, G. Di Natale and B. Rouzeyre, "Preventing scan attacks on secure circuits through scan chain encryption," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 3, pp. 538–550, May 2019.

[60] W. Li, J. Ye, X. Li, H. Li, and Y. Hu, "Bias PUF based secure scan chain design," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 31–36.

[61] E. Ebrard, B. Allard, P. Candelier, P. Waltz, "Review of fuse and antifuse solutions for advanced standard CMOS technologies," *Microelectronics Journal*, vol. 40, no. 12, pp. 1755–1765, Dec. 2009.

[62] F. Saqib, and P. Jim, "VLSI test and hardware security background for hardware obfuscation," in *Hardware Protection through Obfuscation*, Springer, 2017, pp. 33–68.

[63] O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc. USENIX Workshop Smartcard Technol. USENIX Workshop Smartcard Technol.*, vol. 99, pp. 9–20, 1999.

[64] "FIPS 198-1 The Keyed-Hash Message Authentication Code," `http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf`.

[65] "Descriptions of SHA-256, SHA-384, and SHA-512," `http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf`.

[66] "FIPS 180-4 Secure Hash Standard," `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.

[67] M. Juliato, and C. Gebotys, "FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions," University of Waterloo, Tech. Rep, 2011.

[68] H. Niedermayer, A. Klenk, and G. Carle, "The networking perspective of security performance-a measurement study," in *Proc. 13th GI/ITG Con-*

ference Measuring, Modelling and Evaluation of Computer and Communication Systems, pp. 1–17, 2006.

[69] S. Kelly, and S. Frankel, "Request for Comments 4868: Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec," IETF, `https://tools.ietf.org/html/rfc4868`, May, 2007.

[70] T. Dierks, and E. Rescorla, "Request for Comments 5246: The Transport Layer Security (TLS) protocol version 1.2," IETF, `https://tools.ietf.org/html/rfc5246`, August, 2008.

[71] N. J. Al Fardan, and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *Proc. IEEE Symposium on Security and Privacy*, 2013, pp. 526–540.

[72] A. Uskov, and A. Hayk, "The efficiency of block ciphers in galois/counter mode in IPsec-based virtual private networks," in *Proc. IEEE International Conference on Electro/Information Technology*, 2014, pp. 173–178.

[73] K. Bhargavan, and G. Leurent, "Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH," in *Proc. Network and Distributed System Security Symposium*, 2016.

[74] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Lecture Notes in Computer Science*, vol. 773, pp. 368–378, 1993.

[75] 藤代美佳, "暗号集積回路に対するスキャンベースサイドチャネル攻撃に関する研究," 2016 年度博士論文, 2016.

[76] 独立行政法人産業技術総合研究所, "サイドチャネル攻撃用標準評価ボード SASEBO-GII," `http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-ja.html`.

[77] "Icarus Verilog," `http://iverilog.icarus.com`.

[78] "GTKWave," `http://gtkwave.sourceforge.net`.

[79] Xilinx Inc., "ChipScope Pro およびシリアル I/O ツールキット," `http://japan.xilinx.com/tools/cspro.htm`.

[80] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido, and M. Morales-Sandoval, "Novel Hardware Architecture for implementing the inner loop of the SHA-2 Algorithms," in *Proc. 14th Euromicro Conference on Digital System Design*, 2011, pp. 543-549.

[81] M. Zeghid, B. Bouallegue, M. Machhout, A. Baganne, and R. Tourki, "Architectural design features of a programmable high throughput reconfig-

urable SHA-2 Processor," *Journal of information Assurance and Security 2*, pp. 147–158, 2008.

[82] H. E. Michail, G. S. Athanasiou, V. Kelefouras, G. Theodoridis, and C. E. Goutis, "On the exploitation of a high-throughput SHA-256 FPGA design for HMAC," *ACM Trans. on Reconfigurable Technology and Systems*, vol. 5, no. 1, pp. 2:1–2:28, 2012.

[83] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ACIC unit for the hash function SHA-256 (382, 512)," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, 2004, vol. 3, pp. 70–75.

[84] M. D. Rote, N. Vijendran, and D. Selvakumar, "High performance SHA-2 core using the Round Pipelined Technique," in *Proc. IEEE International Conference on Electronics, Computing and Communication Technologies*, 2015, pp. 1–6.

[85] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Proc. IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, 2006, pp. 317–322.

[86] M. Kim, J. Ryou, and S. Jun, "Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing," *Lecture Notes in Computer Science*, vol. 5487, pp. 240–252, 2008.

[87] 白井太三, 渋谷香士, 秋下徹, 盛合志帆, 岩田哲, "128 ビットブロック暗号 CLEFIA のハードウエア実装評価, " 信学技報, vol. 107, no. 141, ISEC2007-49, pp. 29–36, 2007.

研究業績

論文（学術誌原著論文）

⟨1⟩ **D. Oku**, K. Terada, M. Hayashi, M. Yamaoka, S. Tanaka, and N. Togawa, "A fully-connected Ising model embedding method and its evaluation for CMOS annealing machines," *IEICE Transactions on Information and Systems*, IEICE, vol. E102. D, no. 9, pp. 1696–1706, Feb. 2019, DOI: http://dx.doi.org/10.2197/ipsjtsldm.11.16.

⟨2⟩ ◯ **D. Oku**, M. Yanagisawa, and N. Togawa, "Scan-based side-channel attack against HMAC-SHA-256 circuits based on isolating bit-transition groups using scan signatures," *IPSJ Transactions on System LSI Design Methodology*, IPSJ, vol. 11, pp. 16–28, Sep. 2018, DOI: http://dx.doi.org/10.1587/transinf.2018EDP7411.

国際会議（査読付）

⟨3⟩ K. Takehara, **D. Oku**, Y. Matsuda, S. Tanaka, and N. Togawa, "A multiple coefficients trial method to solve combinatorial optimization problems for simulated-annealing-based Ising machines," in *Proc. IEEE International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 64–69, Berlin, German, Sep. 2019, DOI: http://dx.doi.org/10.1109/ICCE-Berlin47944.2019.8966167.

⟨4⟩ **D. Oku**, S. Tanaka, and N. Togawa, "A concept of Ising machines common platform," in *Adiabatic Quantum Computing (AQC)*, Innsbruck, Austria, Jun. 2019.

⟨5⟩ S. Kanamaru, **D. Oku**, M. Tawada, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "Efficient Ising model mapping to solving slot placement problem," in *Proc. IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, Las Vegas, America,

Jan. 2019, DOI: http://dx.doi.org/10.1109/ICCE.2019.8661947.

⟨6⟩ K. Terada, **D. Oku**, S. Kanamaru, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "An Ising model mapping to solve rectangle packing problem," in *Adiabatic Quantum Computing (AQC)*, Mountain View, America, Jun. 2018.

⟨7⟩ K. Terada, **D. Oku**, S. Kanamaru, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "A fully-connected Ising model embedding method and its evaluation for CMOS annealing machines," in *IEEE/ACM 55th Design Automation Conference (DAC)*, San Francisco, America, Jun. 2018.

⟨8⟩ K. Terada, **D. Oku**, S. Kanamaru, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "An Ising model mapping to solve rectangle packing problem," in *Proc. IEEE International Symposium on VLSI Design, Automation and Test (VLSI–DAT)*, pp. 1–4, Hsinchu, Taiwan, Apr. 2018, DOI: http://dx.doi.org/10.1109/VLSI-DAT.2018.8373233.

⟨9⟩ ○ **D. Oku**, M. Yanagisawa, and N. Togawa, "A robust scan-based side-channel attack method against HMAC-SHA-256 circuits," in *Proc. IEEE International Conference on Consumer Electronics–Berlin (ICCE–Berlin)*, pp. 79–84, Berlin, German, Sep. 2017, DOI: http://dx.doi.org/10.1109/ICCE-Berlin.2017.8210596.

⟨10⟩ ○ **D. Oku**, M. Yanagisawa, and N. Togawa, "Implementation evaluation of scan-based attack against a Trivium cipher circuit," in *Proc. IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 220–223, Jeju, Korea, Oct. 2016, DOI: http://dx.doi.org/10.1109/APCCAS.2016.7803938.

国内学会 (査読付)

⟨11⟩ **於久 太祐**, 多和田 雅師, 柳澤 政生, 戸川 望, "スキャンシグネチャを用いた周辺回路を含む軽量暗号 CLEFIA に対するスキャンベース攻撃," 情報処理学会 DA シンポジウム 2017 論文集, 加賀市, 石川県, pp. 116–121, Aug. 2017.

⟨12⟩ **於久 太祐**, 多和田 雅師, 柳澤 政生, 戸川 望, "スキャンシグネチャを用いたスキャンデータ解析に基づく HMAC-SHA-256 ハッシュ回路のスキャン

ベース攻撃," 情報処理学会 DA シンポジウム 2016 論文集, 加賀市, 石川県, pp. 2–7, Sep. 2016.

国内学会 (査読なし)

⟨13⟩ 竹原 康太, 於久 太祐, 松田 佳希, 田中 宗, 戸川 望, "SA ベースのイジングマシンにより巡回セールスマン 問題を高速解法するための多種軽量係数試行法," 情報処理学会研究報告, 西之表市, 鹿児島県, vol. 2019-SLDM-187, no. 53, pp. 1–6, Mar. 2019.

⟨14⟩ 金丸 翔, 於久 太祐, 多和田 雅師, 田中 宗, 林 真人, 山岡 雅直, 柳澤 政生, 戸川 望, "イジング計算機によるスロット配置問題の解法," 電子情報通信学会技術報告, 札幌市, 北海道, vol. 118, no. 83, VLD2018-34, pp. 161–166, Jun. 2018.

⟨15⟩ **於久 太祐**, 多和田 雅師, 柳澤 政生, 戸川 望, "鍵長 128 ビット, 192 ビット, 256 ビットの軽量暗号 CLEFIA に対するスキャンベース攻撃手法," 情報処理学会研究報告, 隠岐の島町, 島根県, vol. 2018-SLDM-183, no. 43, pp. 1–6, Mar. 2018.

⟨16⟩ **於久 太祐**, 柳澤 政生, 戸川 望, "連続してハッシュ値を出力しない HMAC-SHA-256 回路へのスキャンベース攻撃手法, " 情報処理学会研究報告, 久米島町, 沖縄県, 2017-SLDM-179, no. 22, pp. 1–6, Mar. 2017.

⟨17⟩ **於久 太祐**, 柳澤 政生, 戸川 望, "Trivium 暗号回路に対するスキャンベース攻撃の実装評価, " 電子情報通信学会技術報告, 弘前市, 青森県, vol. 116, no. 94, VLD2016-8, pp. 7–12, Jun. 2016.

業績賞等

⟨18⟩ 情報処理学会 第 179 回システムと LSI の設計技術研究発表会 優秀発表学生賞 (2017 年 8 月)

⟨19⟩ ICCE-Berlin 2017 Best Paper (2017 年 9 月)

研究費・助成金

⟨20⟩ 日本学術振興会特別研究員奨励費, "集積回路に対するスキャンベース攻撃手法と防御手法に関する研究," 2020 年 4 月–2022 年 3 月, 総額 210 万円 (2020 年度: 110 万円, 2021 年度:100 万円).