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

於久 太祐[†1,a)]　多和田 雅師[†1]　柳澤 政生[†1]　戸川 望[†1]

概要：テスト容易化技術としてスキャンパステストが存在する．スキャンパステストで用いるスキャンチェインは集積回路内のレジスタを直列に繋いでおり，外部から内部レジスタを観測や制御できる．スキャンチェインを悪用したサイドチャネル攻撃としてスキャンベース攻撃が存在する．これまでブロック暗号，ストリーム暗号，公開鍵暗号を対象としたスキャンベース攻撃が提案されているが，ハッシュ関数が搭載された回路に対するスキャンベース攻撃はまだ報告されていない．メッセージ認証でハッシュ関数を用いる方法に HMAC がある．ハッシュ関数を複数回適用するもので TLS 等で採用されている．HMAC において圧縮関数に SHA-256 を用いるものを HMAC-SHA-256 と言う．本稿では，周辺回路を含む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．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 ステップを実行することでスキャンデータと 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．計算機実験から，HMAC-SHA-256 回路とその他制御回路のレジスタがスキャンチェインに接続されている場合でも，提案手法により HMAC-SHA-256 回路ので用いる秘密鍵を復元できることを確認した．

キーワード：サイドチャネル攻撃，スキャンベース攻撃，ハッシュ関数，HMAC-SHA-256

1. はじめに

近年，集積回路上のトランジスタ数は増大し回路の微細化が進んでいる．回路の故障検出技術や動作テストは困難になっており，テスト容易化技術が重要となっている．テスト容易化技術のとしてスキャンパステストがある．スキャンパステストは回路内にテスト用のスキャンチェインを搭載し，製造した集積回路をテストする．スキャンチェインはレジスタを直列に繋ぎ，外部からレジスタを直接観測や制御できる．スキャンパステストは故障検出や動作テストの効率を高めることができる．

テスト容易化技術の需要が高まる一方で，テスト容易化技術を悪用したサイドチャネル攻撃が注目されている．サイドチャネル攻撃は暗号回路を PC やオシロスコープなどの機器を使い，外部から物理的観測・計測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 [8]，電力差分攻撃 [9] などが報告されており，スキャンパステストを悪用したサイドチャネル攻撃として，スキャンベース攻撃も報告されている．スキャンベース攻撃はスキャンチェインから内部レジスタの値を取得できることを利用し，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暗号回路を攻撃する．

スキャンベース攻撃の既存手法にはブロック暗号 AES に対する手法 [1]，ストリーム暗号 Trivium に対する手法 [2]，公開鍵暗号 RSA に対する手法 [3] などがある．既存手法はスキャンチェインから暗号化中のレジスタの値を取得できることを前提としている．いずれの手法もシミュレーション実験では，暗号回路とその周辺回路のレジスタがスキャンチェインに含まれている場合でも，スキャンベース攻撃できることが示されている．

本稿ではハッシュ回路に対するスキャンベース攻撃を考える．ハッシュはデータからハッシュ関数を用いてハッシュ値と呼ばれる整数列を生成する．ハッシュはデータやファイルが改ざんされていないかを確認する，データから特定のデータを探索する，パスワードの代わりにハッシュ値を用いて照合する，という主に 3 つの用途がある．メッセージを認証するアルゴリズムとして HMAC [7] がある．HMAC は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 に対するサイドチャネル攻撃 [4] が提案されているが，HMAC へのスキャンベース攻撃手法はまだ提案されていない．

[†1] 現在，早稲田大学
Presently with Waseda Uniersity
[a)] daisuke.oku@togawa.cs.waseda.ac.jp

表 1 演算の表記法.

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

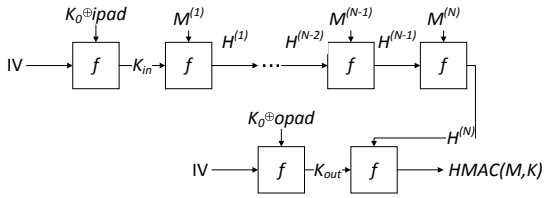

図 1 HMAC の概略.

本稿では 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．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 ステップを実行することでスキャンデータと 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．計算機実験により，HMAC-SHA-256 回路とその他制御回路のレジスタがスキャンチェイン上にある場合でも提案手法を用いることで HMAC-SHA-256 回路の秘密鍵を復元することができた．

2. HMAC-SHA-256

本章では HMAC と SHA-256 のアルゴリズム，HMAC-SHA-256 を紹介する．本章での演算の表記法を表 1 に示す．

2.1 HMAC [7]

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の 1 種である．HMAC でハッシュ関数 H，メッセージ M と鍵 K から認証コード値を式 (1) により生成する．

$$HMAC(M,K) = H(K_{out}, H(K_{in}, M)) \tag{1}$$

HMAC の概略を図 1 に示す．図 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 の生成方法を式 (2) に示す．

$$K_0 = \begin{cases} K & (L(K) = B) \\ K|| \underbrace{0\ldots0}_{B-L(K)} & (L(K) < B) \\ H(K)|| \underbrace{0\ldots0}_{B-L(H(K))} & (B < L(K)) \end{cases} \tag{2}$$

L をビット長を求める関数とし，$L(H(K)) < B$ とする．K_{in}，K_{out} は秘密鍵である．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．図 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．

2.2 SHA-256 [5, 6]

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-256 では入力するメッセージを 512 ビット毎のメッセージブロック $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$ に分割し [*1]，繰り返しハッシュ化操作をする．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) に示す．

$$H^{(i)} = f(M^{(i)}, H^{(i-1)}) \tag{3}$$

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 のアルゴリズムを以下に示す．

(1) $i-1$ 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．初期化は以下の通りである．

$$\begin{aligned} a &\leftarrow H_1^{(i-1)} \\ b &\leftarrow H_2^{(i-1)} \\ &\vdots \\ h &\leftarrow H_8^{(i-1)} \end{aligned}$$

ここで，$H_k^{(i-1)}$ は 8 分割された中間ハッシュ値 $H^{(i-1)}$ の k 番目を表す．

(2) 以下の更新処理を 64 回繰り返す．

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

$$T_1 \leftarrow h + Rot_2(e) + Ch(e,f,g) + K_j + W_j \tag{4}$$
$$T_2 \leftarrow Rot_1(a) + Maj(a,b,c) \tag{5}$$
$$h \leftarrow g \tag{6}$$
$$g \leftarrow f \tag{7}$$
$$f \leftarrow e \tag{8}$$
$$e \leftarrow d + T_1 \tag{9}$$
$$d \leftarrow c \tag{10}$$
$$c \leftarrow b \tag{11}$$
$$b \leftarrow a \tag{12}$$
$$a \leftarrow T_1 + T_2 \tag{13}$$

[*1] メッセージを分割する際，メッセージの終了位置に 1 を，最後にメッセージ長を挿入する．メッセージの終了位置を表す 1 とメッセージ長を表すビットの間を 0 でパディングすることで，全てのメッセージブロックが 512 ビットとなるようにする．詳細は [5,6] を参照されたい．

end for
ここで,

$$Ch(e,f,g) = (e \wedge f) \oplus (\bar{e} \wedge g) \quad (14)$$
$$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \quad (15)$$
$$Rot_1(a) = S^2(a) \oplus S^{13}(a) \oplus S^{22}(a) \quad (16)$$
$$Rot_2(e) = S^6(e) \oplus S^{11}(e) \oplus S^{25}(e) \quad (17)$$

である．$K_j(j = 0 \ldots 63)$ 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．

$$W_j = \begin{cases} M_j{}^{(i)} & (j = 0, 1, \ldots, 15) \\ X_j & (j = 16, 17, \ldots, 63) \end{cases} \quad (18)$$

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

$$X_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$$
$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$$
$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$$

SHA-256 の圧縮関数の概略図を図 2 に示し，W_j を求めるメッセージスケジュールの概略図を図 3 に示す．図中の四角形はレジスタを表し，上述のステップ (1) あるいは式 (18) の $M_j^{(i)}$ によって，初期化される．また，ADD は 2^{32} を法とする加算を意味する．

(3) i 番目の中間ハッシュ値 $H^{(i)}$ を以下の式で計算する．

$$H_1^{(i)} \leftarrow a + H_1^{(i-1)}$$
$$H_2^{(i)} \leftarrow b + H_2^{(i-1)}$$
$$\vdots$$
$$H_8^{(i)} \leftarrow h + H_8^{(i-1)}$$

2.3 HMAC-SHA-256

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 [10, 11]. 本稿では，図 2 および図 3 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．2.2 節の (1) と (3) の処理を 1 クロックとし，HMAC-SHA-256 回路は全体 $(N + 3) \times 66$ クロックで処理を完了する．

3. 前提条件

HMAC-SHA-256 ハッシュ回路を対象としたサイドチャネル攻撃 [4] は，図 1 中の K_{in}，K_{out} を秘密鍵とみなし，

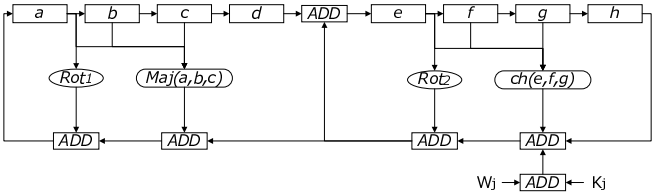

図 2 SHA-256 の圧縮関数.

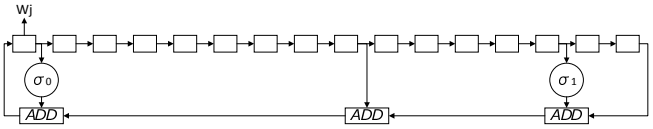

図 3 SHA-256 のメッセージスケジュール.

復元対象としている．本稿の目的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} を復元することとする．

本稿の前提条件を以下に示す．

HMAC-SHA-256 ハッシュ回路への攻撃で攻撃者は以下のことがわかるとする．

- ハッシュ値を生成するタイミング.
- HMAC-SHA-256 の 8 つの 32 ビットレジスタ $a \sim h$ がスキャンチェインに接続されている.

HMAC-SHA-256 ハッシュ回路への攻撃で攻撃者は以下のことができるとする．

- HMAC-SHA-256 回路に任意のメッセージを入力できる.
- 任意のタイミングで HMAC-SHA-256 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できる.

HMAC-SHA-256 ハッシュ回路への攻撃で攻撃者は以下のことがわからないとする．

- HMAC-SHA-256 回路のスキャンチェインのレジスタ接続順.
- HMAC-SHA-256 回路のスキャンチェインに含まれるレジスタの数・種類.

4. スキャンベース攻撃提案手法

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．2.2 節の式 (4)–(13) ならびに図 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．遷移グループの特定 (4.1 節)，ビット位置の特定 (4.2 節)，前半レジスタと後半レジスタの特定 (4.3 節) の 3 ステップから構成される手順によって，これを実現する．

4.1 遷移グループの特定

2.2 節の式 (4)–(13) で示した圧縮関数の動作の通り，HMAC-SHA-256 では，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．つまり，レジスタ a の i 番目のビットを a_i，レジスタ b の i 番目のビットを b_i，レジスタ c の i 番目のビットを c_i，レジスタ d の i 番目のビットを d_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，レジスタ e の i 番目のビットを e_i，レジスタ f の i 番目のビットを f_i，レジスタ g の i 番目のビットを g_i，レジスタ h の i 番目のビットを h_i としたとき，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ことを考える．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 4 に示す．図 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．2.2 節 (2) のように，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 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ニチャを考えたとき，その長さを $(n+1)$ と定義する．なお，n を大きくするのと同時に，入力するメッセージ数を大きくすることで，遷移グループを特定することもできる．

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．

4.2 ビット位置の特定

4.1 節より，遷移グループが 64 個特定できた．本節では，64 個の遷移グループから 2 グループずつペア化し，32 個

図 4 レジスタ a の i ビット目の遷移の探索．

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，式 (4)–(13) 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 X, Y 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 は以下の式で表せる．

$$a^1 = X + W_i^1 \tag{19}$$

$$e^1 = Y + W_i^1 \tag{20}$$

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．

$$a^2 = X + W_i^2 \tag{21}$$

$$e^2 = Y + W_i^2 \tag{22}$$

W_j^1，W_j^2 はメッセージに依存した変数である．適当な変数 α を用いると $W_i^2 = W_i^1 + \alpha$ と表せる．ここでもし，α の MSB が 1 でその他のビットは 0 であるとき，式 (19)–(22) 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 もしくは 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，判定するビット位置を求める．順々に LSB まで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 2 に示す．表 2 で w_j^1, w_j^2 $(0 \le j \le 3)$ は式 (20)–(23) の W_i^1, W_i^2 を 8 ビットずつに分割した時の一要素を表し，β はどのビットが反転しているかを示している．w_j^1, w_j^2 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．

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，表 2 からメッセージの上

表 2 ハミング距離が 1 である例.

w_j^1	w_j^2	β
無い	無い	10000000
q	1	01000000
q	Q	00100000
q	a	00010000
q	y	00001000
q	u	00000100
q	s	00000010
q	p	00000001

位 32 ビットの例として “qqqq” と “1qqq” を入力する．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 ビット反転)．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 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，w_j^1，w_j^2 を組み合わせることで，遷移グループから求められるペアのビット位置を求めることが可能となる．つまり，32 組のペアの各ペア中の 2 つの遷移グループは前半レジスタ ($a \sim d$) か後半レジスタ ($e \sim h$) の i 番目のビットを表す．

4.3 レジスタ a, e の判定

4.1 節と 4.2 節から 64 個の遷移グループを特定し，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，遷移グループがレジスタ a，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式 (5)，(9)，(13) を計算する．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．

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

必要となるレジスタは 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前半レジスタか後半レジスタかを特定できる．

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，$Rot_1(a^T)$ では 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 パターン分試行する必要がある．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 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．

5. 評価実験

本章では，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，HMAC-SHA-256 回路に使われる秘密鍵を復元する実験の結果を示す．実験では，提案手法を python を用いて実装し，HMAC-SHA-256 シミュレータとして SHA-256 を python を用いて実装した．ホスト PC の OS は OS X El Capitan，CPU は Intel Core i5(2.6GHz)，メモリは 8GB のものを使用した．

5.1 実験方法

提案手法では 3 つのステップがあり，解析に必要となるものは複数サイクル分のスキャンデータと複数の入力メッセージである．入力メッセージ数とスキャンデータのサイクル数を変化させたとき，提案手法の各ステップにどのような影響があるかを調べる．実験の条件を以下に示す．

(1) 入力メッセージ数とスキャンチェイン長を変化させたとき，4.1 節の遷移グループの特定に必要なスキャンシグネチャの長さを求める実験.
(2) 遷移グループを特定している前提で 4.2 節のビット位置の特定に必要な入力数を求める実験.
(3) K_{in}, K_{out} を求める実験.

各実験では HMAC-SHA-256 回路にメッセージを入力したときの各サイクルごとのスキャンデータを得る．

5.2 実験結果

5.2.1 遷移グループを特定する実験

本節では，32 ビットの入力メッセージをランダムで生成し，入力メッセージとスキャンチェイン長を変化させたとき，4.1 節の遷移グループの特定に必要な最小スキャンシグネチャ長を求める実験の結果を示す．

表 3 遷移グループの特定に必要な最小スキャンシグネチャ長.

入力	スキャンチェイン長 [ビット]			
メッセージ数	256	512	1024	2048
1	14	17	22	23
2	9	10	10	11
3	6	7	8	8
4	7	7	7	6
5	6	6	6	5
10	5	5	5	5
15	5	5	5	5
20	5	5	5	5

入力するメッセージを 1 から 5，10，15，20 と変化させ，スキャンチェイン長は 256，512，1024，2048 ビットと変化させた．スキャンチェイン長が 256 ビットのとき，スキャンチェインは図 2 のレジスタ $a \sim h$ の合計 256 ビットが接続される．スキャンチェイン長が 512，1024，2048 ビットのときはスキャンチェイン長が 256 ビットのときのスキャンデータにランダムで 256 ビット，768 ビット，1792 ビットのデータを挿入した．HMAC-SHA-256 にメッセはージを入力し，得られたスキャンデータから 4.1 節の手法を使って遷移グループを特定した結果を表 3 に示す．

表 3 から，メッセージ数が 1 であってもスキャンシグネチャ長を長く取ることで，遷移グループの特定が可能である．メッセージ数が 1 としたときの特定に必要な時間はおおよそ数十分であった．また，入力メッセージを多くした場合，スキャンシグネチャ長は 5 サイクル分で特定が可能であった．しかし，特定に必要な時間は数時間程度かかった．

5.2.2 ビット位置を特定する実験

本節では，遷移グループの特定ができた後，32 ビットから 512 ビットの入力メッセージを複数生成し，4.2 節のビット位置の特定に必要な入力メッセージ数を求める実験の結果を示す．

入力メッセージを表 2 を参考に 56 個生成し，HMAC-SHA-256 に入力した結果，64 個の遷移グループから 32 組のペアができ，ビットの位置を特定できた．特定に必要な時間はおおよそ数十分であった．

5.2.3 K_{in}, K_{out} を求める実験

本節では，5.2.1 項と 5.2.2 項の結果を用いて，4.3 節の方法でレジスタ a, e の判定に成功した．その後，K_{in}，K_{out} が使われるタイミングを特定したところ，K_{in}，K_{out} の復元に成功した．復元に要する時間はスキャンチェイン長に依存して数十分から数時間程度であった．

6. おわりに

本稿ではスキャンシグネチャを用いた 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した．提案手法は取得したスキャンデータに対してレジスタの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 ステップがある．3 つのステップを実行することで，スキャンデータから内部レジスタの状態を復元することができ，HMAC のハッシュ関数で用いる秘密鍵を求めることができる．コンピュータ上でのシミュレーション実験より，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ことを示した．

提案手法でハッシュ値を生成するタイミングが不明であった場合，生成するタイミングを予測し，ある程度の範囲からスキャンデータを取得して攻撃することで，秘密鍵の復元が可能であると考えられる．

今後の課題としてはスキャンチェインにスキャンベース攻撃に対する防御手法の提案がある．今後考察する予定である．

謝辞 本研究開発は一部, 総務省 SCOPE(受付番号 141303001) の委託を受けた．

参考文献

[1] R. Nara, N. Togawa, M. Yanagisawa, and 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.

[2] 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.

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Proc. CRYPTO ‘99*, Springer-Verlag, pp. 388–397, 1999.

[10] 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.

[11] 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.