Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9

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

於久 太祐[†1,a)]　柳澤 政生[†1]　戸川 望[†1]

概要：LSI のテスト容易化設計技術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 LSI のレジスタを直列に繋いだスキャンチェイン利用し, 外部から回路内部のレジスタを観測や制御する手法である．スキャンチェインを用いるサイドチャネル攻撃としてスキャンース攻撃がある．これまでにブロック暗号, 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．ハッシュ関数を用いてメッセージ認証する仕組みとして 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 でハッシュ関数に SHA-256 を用いるものを HMAC-SHA-256 と言う．我々は HMAC-SHA-256 回路に対するスキャンベース攻撃を提案したが, HMAC 内の SHA-256 回路が連続してハッシュ値を出力する場合を仮定している．本稿では, SHA-256 回路が連続してハッシュ値を出力しない HMAC-SHA-256 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, 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 回路とその他回路のレジスタがスキャンチェインに接続され, HMAC 内の SHA-256 回路が連続してハッシュ値を出力しない場合でも, 提案手法により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, SHA-256

1. はじめに

LSI の故障検出や動作確認は重要な技術であるが, 回路規模の拡大・微細化が進んでいるため困難になっている．そのため, 故障検出技術やテスト容易化技術が益々重要になっている．LSI のテスト容易化設計技術の 1 つ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, LSI のレジスタを直列に繋いだテスト用のスキャンチェインを利用し, 外部から回路内部のレジスタを観測や制御する手法であり, 故障検出や動作テストの効率を高めることができる．

テスト容易化設計技術が重要となる一方で, テスト容易化設計技術を悪用したサイドチャネル攻撃が報告されている．サイドチャネル攻撃は外部から PC やオシロスコープなどを使い, 動作中の回路から副次的情報を物理的観測, 計測することで秘密情報を得る攻撃である．サイドチャネル攻撃として電力差分攻撃 [1], タイミング攻撃 [2] などが報告されている．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も報告されている．スキャンベース攻撃では, スキャンチェインから得られる内部レジスタ値であるスキャンデータを悪用する．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 [3], ストリーム暗号 Trivium に対する手法 [4], 公開鍵暗号 RSA に対する手法 [5] などがある．

メッセージを認証する仕組みとして HMAC [6] 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値を認証に使用する．HMAC に対するサイドチャネル攻撃 [10] も報告されている．特に, 我々は HMAC-SHA-256 回路に対するスキャンベース攻撃 [9] を提案したが, HMAC 内の SHA-256 回路が連続してハッシュ値を出力することを仮定している．

HMAC では秘密鍵をある長さに設定し, メッセージを指定長に分割する必要がある．メッセージを分割した個数だけ SHA-256 回路が動作する．秘密鍵を設定する動作とメッセージを分割する動作は回路構造や秘密鍵長, メッセージ長に強く依存するため, SHA-256 回路は必ずしも連続して動作しない．このような場合, 文献 [9] で提案したスキャンベース攻撃手法は秘密鍵の復元できない可能性がある．

本稿では, SHA-256 回路が連続してハッシュ値を出力し

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

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9

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
$\boxplus$	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

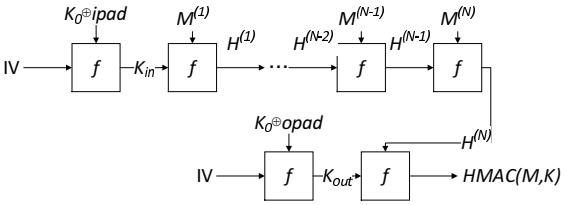


図 1 HMAC の概略.

ない HMAC-SHA-256 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 ステップにより, 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 回路とその他回路のレジスタがスキャンチェインに接続され, HMAC 内の SHA-256 回路が連続して動作しない場合でも, 提案手法により秘密鍵を復元できることを確認した.

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. 本章の演算の表記法を表 1 に示す.

2.1 HMAC [6]

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. 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.

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

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, $L(H(K)) < B$ 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. 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.

Algorithm 1 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
$H^{(N)} = (H_1^{(N)} || H_2^{(N)} || \cdots || H_8^{(N)})$;

2.2 SHA-256 [7, 8]

SHA-256 は NIST によって標準化されたハッシュ関数で, 256 ビットのハッシュ値を出力する. SHA-256 ではメッセージ M を 512 ビット毎のブロック $M^{(1)}, M^{(2)}, \cdots, M^{(N)}$ に分割し[*1], 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 512 ビットのメッセージ $M^{(i)}$ 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の式を以下に示す.

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

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 Algorithm1 に示す. Algorithm1 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.

フェーズ 2 ではレジスタの更新を 64 回する. Algorithm1 中のそれぞれ変数を以下に示す.

$$Ch(e,f,g) = (e \wedge f) \oplus (\bar{e} \wedge g) \quad (1)$$

$$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \quad (2)$$

$$Rot_1(a) = S^2(a) \oplus S^{13}(a) \oplus S^{22}(a) \quad (3)$$

$$Rot_2(e) = S^6(e) \oplus S^{11}(e) \oplus S^{25}(e) \quad (4)$$

$K_j (j = 0 \ldots 63)$ は SHA-256 特有の 32 ビットの定数であ

[*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.

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9

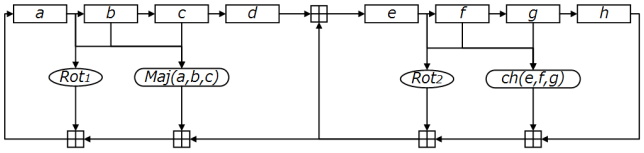


図 2 SHA-256 の圧縮関数.

る. W_j は以下の式で求められる.

$$W_j = \begin{cases} {M_j}^{(i)} & (j = 0, 1, \ldots, 15) \\ X_j & (j = 16, 17, \ldots, 63) \end{cases} \quad (5)$$

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.

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 [11, 12].

本稿では図 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. Algorithm1 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.

本稿で想定する HMAC-SHA-256 回路は連続してハッシュ値を生成しないため, SHA-256 回路が 1 回の動作を完了し, 次の動作を開始するまでに複数クロックかかるものとする. この間のクロックを合計で X クロックとすると, 全体としては $(N+3) \times 66 + X$ 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 [10] は, 図 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.

本稿では, HMAC-SHA-256 ハッシュ回路への攻撃でスキャンチェインは, HMAC-SHA-256 の 8 つの 32 ビットレジスタ $a \sim h$ が接続されて, 反転・動的に変化しないものとし, スキャンデータの圧縮はされていないものとする. 攻撃者は HMAC-SHA-256 回路に任意のメッセージを入力でき, 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. 攻撃者は HMAC-SHA-256 回路のスキャンチェインのレジスタ接続順, 数, 種類がわからないとする.

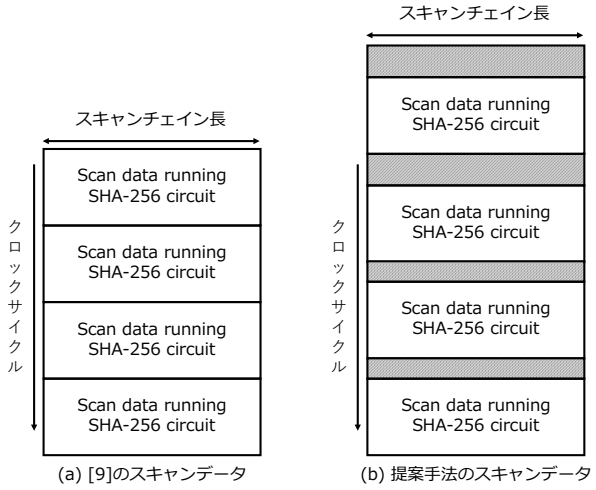


図 3 想定しているスキャンデータ

4. スキャンベース攻撃手法

我々は HMAC-SHA-256 回路へのスキャンベース攻撃 [9] を提案したが, HMAC 内の SHA-256 回路が連続してハッシュ値を出力しない場合, 得られるスキャンデータから秘密鍵を復元できないと考えられる. 本章では, 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を提案する.

文献 [9] で検討したスキャンデータと本章で検討するスキャンデータの概略を図 3 に示す. 図 3(a) では, 遷移グループの特定 (4.1 節), ビット位置の特定 (4.3 節), 前半レジスタと後半レジスタの特定 (4.4 節) の 3 ステップで解析が可能であった. しかし, 図 3(b) 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 ステップだけでは解析は困難であると考えられる. 本章では SHA-256 回路動作の初期位置の特定 (4.2 節) なるステップを加え, 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.

4.1 遷移グループの特定 [9]

2.2 節のフェーズ 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,

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9



図 4 レジスタ a の i ビット目の遷移の探索 [9].

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 ($n < 64$) 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ニチャを考えたとき, 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.

4.2 SHA-256 回路動作の初期位置の特定

4.1 節で示した遷移グループ特定アルゴリズムは, 図 3(a) のように, スキャンデータにランダムデータを含まない場合に対して適用できる. 図 3(b) のように途中にランダムデータ (灰色部分) が含まれるスキャンデータを考えたとき, 図 5 の実線枠のようにランダムデータを含まないようにスキャンシグニチャをとることができれば, 4.1 節で示した遷移グループの特定アルゴリズムを適用することができる. ところが, 図 5 の点線枠のようにスキャンシグニチャをとると, スキャンシグニチャがランダムデータを含むため, 4.1 節で示した遷移グループの特定アルゴリズムを適用することができない. つまり, 遷移グループ特定のために

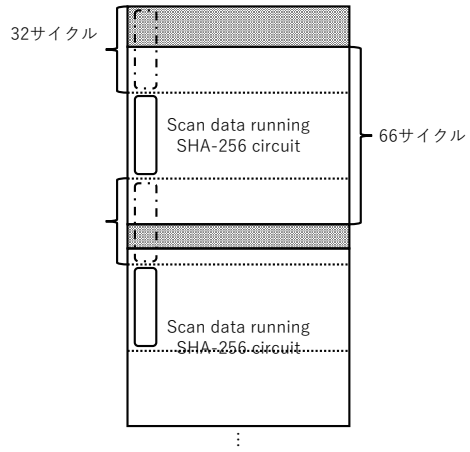


図 5 ランダムデータを含むスキャンデータへの探索.

は, スキャンシグネチャの範囲は, 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.

そこで, ランダムデータを含むスキャンデータ全体を 32 サイクル長ごとに区切り, 複数の 32 サイクル毎の部分スキャンデータを考える. そして部分スキャンデータに対して, 4.1 節で示した遷移グループ特定アルゴリズムを適用するものとする. SHA-256 回路は 1 回の動作に 66 クロック要するため, 部分スキャンデータのうちいくつかは必ずランダムデータを含まないものとなる. そのため部分スキャンデータのいくつかでは, 遷移グループ特定アルゴリズムが成功し遷移グループが特定できる.

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. メッセージを最小にした時, このような SHA-256 回路が動作を開始する初期位置を 4 個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.

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9

4.3 ビット位置の特定 [9]

4.1 節で遷移グループが 64 個特定でき, 4.2 節で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. 本節では, 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 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.

$$a^1 = X \boxplus W_i^1 \tag{6}$$

$$e^1 = Y \boxplus W_i^1 \tag{7}$$

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.

$$a^2 = X \boxplus W_i^2 \tag{8}$$

$$e^2 = Y \boxplus W_i^2 \tag{9}$$

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 (6)–(9) 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 2 に示す. 表 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 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.

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.

表 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

上記の通り, 表 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.

4.4 レジスタ a, e の判定 [9]

4.1 節でスキャンデータから 64 個の遷移グループを特定した. 4.2 節でスキャンデータからレジスタの初期化位置を特定した. 4.3 節では 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.

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

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.22
Vol.2017-SLDM-179 No.22
Vol.2017-EMB-44 No.22
2017/3/9

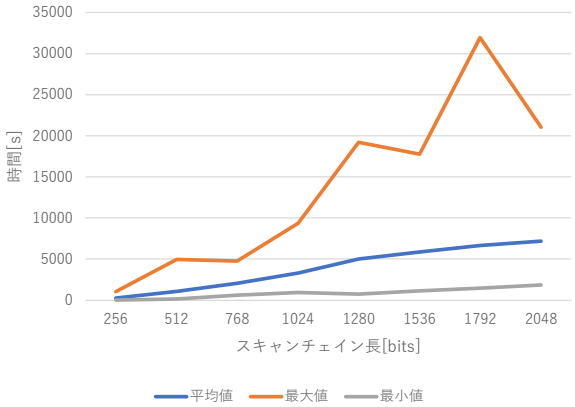


図 6 秘密鍵 K_{in}, K_{out} を復元する時間.

ジスタの対応を求め, 連続してハッシュ値を出力しない 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(2.6GHz), メモリは 8GB のものを使用した.

5.1 実験方法

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. 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 5(b) 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.

5.2 実験結果

各スキャンチェイン長で秘密鍵を復元した. 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 6 に示す. 図 6 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. 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. 入力するメッセージ数は最大で約 150 個程であった.

6. おわりに

本稿では, SHA-256 回路が連続してハッシュ値を出力しない HMAC-SHA-256 回路に対するスキャンベース攻撃手法を提案した. 提案手法は遷移グループの特定, 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 ステップがある. 4 ステップを実行することで, 入力メッセージ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, 秘密鍵を復元できる. 計算機上でのシミュレーション実験から, HMAC-SHA-256 回路とその他回路のレジスタがスキャンチェインに接続され, HMAC 内の SHA-256 回路が連続してハッシュ値を出力しない場合でも, 提案手法により秘密鍵を復元できることを確認した.

今後の課題としては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] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Proc. CRYPTO ‘99*, Springer-Verlag, pp. 388–397, 1999.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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,” DA シンポジウム 2016 論文集, pp. 2–7, 2016.

[10] 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.

[11] 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.

[12] 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.

ⓒ 2017 Information Processing Society of Japan