SHA-1の攻撃専用ハードウェアのアーキテクチャとコストに関する考察

佐藤 証[†]

ハッシュ関数の攻撃に関する研究が急速に進展し，数多くのアルゴリズムが相次いで破られている．Wang らが提案した手法は，最も広く使われているハッシュ関数 SHA-1 の Collision を 2^{69} のオーダ計算量で求めることが可能であると見積もられている．これは総当り攻撃の一種である Birthday Attack の 2^{80} オーダに比べはるかに少ないものの，ソフトウェアだけで計算することは難しい．そこで，本稿では SHA-1 を破るための専用ハードウェア・アーキテクチャを提案し，0.13 μm CMOS スタンダードセル・ライブラリを用いた速度・回路規模・消費電力の評価，さらに攻撃のコストと時間の見積りを行った．1,000 万ドルの予算があれば，32 個の SHA-1 攻撃専用の LSI を搭載した PC ボードを 16 枚つないだ PC システムを 303 セット用意することができる．各 LSI 内では 64 個の SHA-1 コアが並列に動作しており，このシステムにより SHA-1 の Collision を 138 日で生成できることを明らかにした．また Wang らの最新の研究結果として，同様の手法で計算量のオーダを 2^{63} まで下げることができるという報告がなされている．詳細は不明であるが，2^{69} オーダと同様の手法をより効率の良い Collision 生成の検索パスに用いているものと見られる．したがって，この結果をハードウェア攻撃に適用できるならば，100 万ドルの予算で SHA-1 をわずか 22 日で破ることが可能となる．

Study on Architecture and Cost Estimates for SHA-1 Attacking Hardware

Akashi Satoh[†]

The cryptanalysis of hash functions has advanced rapidly, and many hash functions have been broken one after another. The most popular hash function SHA-1 has not been broken yet, but the new collision search techniques proposed by Wang et al. reduced the computational complexity down to 2^{69}, which is greatly lower than the 2^{80} operations needed for a birthday attack, however it is still very large even for today's supercomputers. Therefore, we proposed a hardware architecture to break SHA-1, evaluated speed, area, and power consumption of the architecture by using a 0.13-μm CMOS standard cell library, and estimated cost and time for the attack. A $10 million budget can build a custom hardware system that would consist of 303 personal computers with 16 circuit boards each, in which 32 SHA-1-breaking LSIs are mounted. Each LSI has 64 SHA-1 cores that can run in parallel. This system would find a real collision in 138 days. Wang et al. reported recently that the computational complexity can be reduced down to 2^{63}. Details are not published yet, but almost the same method with the 2^{69} complexity seems to be applied to a more efficient path that produces collisions. Therefore, if the improved method implemented on the SHA-1 attacking hardware, SHA-1 can be broken with a $1 million budget in 22 days.

1. はじめに

ハッシュ関数は任意の長さのメッセージを短い固定長のハッシュ値に圧縮するもので，認証や完全性検証などに用いられる．ハッシュ関数にはメッセージからハッシュ値を求めるのは容易な一方，与えられたハッシュ値を持つメッセージを作ることが計算量的に困難な一方向性（One-wayness）と，同じハッシュ値を持つ複数メッセージの生成が困難な無衝突性（Collision Resistance）が求められる．無衝突性を持つ理想的なハッシュ関数では，同じ n ビットのハッシュ値を持つ 1 組のメッセージを見つけるためには $2^{n/2}$ 程度のメッセージを調べればよく[1)]，これは Birthday Attack と呼ばれる最も基本的な攻撃法である．これに対して，近年ハッシュ関数の攻撃に関する研究が進み，新たな手法が次々と提案されている[11)~24)]．

米国連邦標準技術研究所 NIST（National Institute of Standards and Technology）は 1993 年に 160 ビットのハッシュ関数 SHA（Secure Hash Algorithm）を，FIPS（Federal Information Processing Standard）180[2)] として標準化した．これは Rivest が 1990 年

† 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Japan Ltd.

表 1 ハッシュ関数の概要
Table 1 Summary of Hash Function.

アルゴリズム	ハッシュ長	メッセージブロック長	ラウンド数	提案年	破られた年
MD4	128 bit	32 bit × 16	48	1990	2004
MD5	128	32 bit×16	64	1992	2004
HAVAL-128 /192/224/256	128/192 /224/256 bit	32 bit×32	96-160	1992	2004 (HAVAL-128)
RIPEMD	128 bit	32 bit×16	48 (×2 parallel)	1992	2004
RIPEMD-128	128 bit	32 bit×16	64 (×2 parallel)	1996	
RIPEMD-160	160 bit	32 bit×16	80 (×2 parallel)	1996	
SHA	160 bit	32 bit×16	80	1993	2005
SHA-1	160 bit	32 bit×16	80	1994	
SHA-224	224 bit	32 bit×16	64	2004	
SHA-256	256 bit	32 bit×16	64	2002	
SHA-384/512	384/512 bit	64 bit×16	80	2002	

に提案した 128 ビットのハッシュ関数 MD4[7)~9)] の影響を強く受けている．SHA は 1995 年に，メッセージ拡大関数に 1 ビットの巡回シフトを挿入し安全性を向上させた SHA-1（FIPS 180-1)[3),4)] に修正された．SHA-1 との区別を明確にするために，SHA は SHA-0 と呼ばれることが多い．2002 年に NIST は 256〜512 ビットのハッシュ値を生成する 3 つの新たなハッシュ関数，SHA-256/-384/-512 を開発し，SHA-1 と合わせた FIPS 180-2[5)] を発行した．さらに 2004 年には SHA-256 をベースとした 224 ビットのハッシュ関数 SHA-224[6)] が加えられている．SHA-1 と MD4 の改良版である MD5[10)] は最も広く使用に供されているハッシュ関数であるが，MD5 はすでに Collision を生成することが可能となり[16)~18)]，SHA-0 や SHA-1 に対する強力な攻撃法が近年相次いで発表されている[11)~15),19)~24)]．

Chabaud と Joux は 1998 年に，計算量のオーダ 2^{61} の SHA-0 の差分攻撃法を提案し，80 ラウンドから 35 ラウンドに短縮した SHA-0 に対する Collision を示した[11)]．その後，Joux らは 80 ラウンドの SHA-0 の Collision を 2004〜2005 年に次々と発表している[12)~14),22)]．Wang，Yin，Yu は 2005 年 2 月に SHA-0 と SHA-1 の攻撃に対する短いメモを公開し[19)]，2005 年 8 月にその攻撃法の詳細を発表している[20),21)]．Wang らは，80 ラウンドの SHA-0 と 58 ラウンド短縮版 SHA-1 の Collision をそれぞれ 2^{39} と 2^{33} オーダの計算量で求めることに成功し，80 ラウンドの SHA-1 も Birthday Attack の 2^{80} よりも大幅に少ない 2^{69} のオーダで攻撃であることを明らかにしている．さらに詳細はまだ明らかにされていないが，この攻撃法を改良することでオーダを 2^{63} にまで下げることができるという見積りが示されている[22)]．Wang らは SHA だけでなく，**表 1** にも示したように MD4，MD5，RIPEMD，そして HAVAL-128 といった様々なハッシュ関数の攻撃にも成功している[16)~18)]．

現在のところ，オーダ 2^{63} のハッシュ演算をソフトウェアだけで処理することは困難であるが，専用のハードウェアを用いた場合の検討はなされていない．そこで本稿では，Wang らの手法に基づく SHA-1 の攻撃専用ハードウェア・アーキテクチャを提案し，そのコストと性能評価を行う．

以降では，まず 2 章で SHA-0 と SHA-1 のアルゴリズムを簡単に述べ，3 章で Wang らの手法の概要を説明する．そして 4 章では SHA-1 攻撃専用ハードウェア・アーキテクチャを提案し，さらに 0.13 μm CMOS スタンダードセル・ライブラリによる ASIC チップの性能を示す．次いで，5 章でこの ASIC チップを複数用いたシステムにより，SHA-1 の Collision を求めるためのコストと時間について考察し，最後に 6 章で本稿をまとめる．

2. SHA-0 と SHA-1 のアルゴリズム

SHA-1 の変数のワード長は 32 ビットで，20 ステップごとに切り替わる次の関数 f_t が用いられる．加算はすべて mod 2^{32} 上で行われ，式中の $<<< n$ は左 n ビット巡回シフトを表している．

$$f_i(x,y,z)=\begin{cases}(x\wedge y)\oplus(\neg x\wedge z) & i=0\sim 19\\ x\oplus y\oplus z & i=20\sim 39\\ (x\wedge y)\vee(x\wedge z)\vee(y\wedge z) & i=40\sim 59\\ x\oplus y\oplus z & i=60\sim 79\end{cases} \quad (1)$$

ハッシュ値 $H = H_0||H_1||H_2||H_3||H_4$ を式 (2) の 160 ビット（32 ビット ×5）定数に初期化した後，次の 1）〜4）のステップを，複数の 512 ビットメッセージブロック M に対して繰り返しながらハッシュ値 H を更新する．

$$H = (67452301, efcdab89, 98badcfe, 10325376, c3d2e1f0) \quad (2)$$

1） M を 16 ワード $M_0, M_1, \ldots, M_{15}$ に分割し，80 ワード $W_0, W_1, \ldots, W_{79}$ に拡大する．

(a) SHA-0

$$W_i = \begin{cases} M_i & \\ & i = 0 \sim 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & \\ & i = 16 \sim 79 \end{cases} \quad (3)$$

(b) SHA-1

$$W_i = \begin{cases} M_i & i = 0 \sim 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & \\ <<< 1 & \\ & i = 16 \sim 79 \end{cases} \quad (4)$$

2） 5 ワードの作業変数 a，b，c，d，e を，前回までのハッシュ値 H で更新する．

$$(a_0, b_0, c_0, d_0, e_0) = (H_0, H_1, H_2, H_3, H_4) \quad (5)$$

3） 圧縮関数と呼ばれる次の一連の処理を 80 ラウンド（$i = 0 \sim 79$）繰り返す．なお加算は mod 2^{32} 上で行われる．

$$\begin{aligned}(a_{i+1}, b_{i+1}, c_{i+1}, d_{i+1}, e_{i+1}) \\ = ((a_i <<< 5) + f_i(a_i, b_i, c_i) + e_i \\ + W_i + K_i, a_i, b_i <<< 30, c_i, d_i)\end{aligned} \quad (6)$$

ここで K_i は 20 ステップごとに変わる次の値である．

$$K_i = \begin{cases} 5abe7999 & i = 0 \sim 19 \\ 6ed9eba1 & i = 20 \sim 39 \\ 8f1bbcdc & i = 40 \sim 59 \\ ca62c1d6 & i = 60 \sim 79 \end{cases} \quad (7)$$

4） mod 2^{32} 上の加算によりハッシュ値を更新する．

$$\begin{aligned}(H_0, H_1, H_2, H_3, H_4) = \\ (H_0 + a_{80}, H_1 + b_{80}, H_2 + c_{80}, \\ H_3 + d_{80}, H_4 + e_{80})\end{aligned} \quad (8)$$

SHA-0 と SHA-1 の唯一の違いは式 (3)，(4) に示したように，メッセージ拡大処理において 1 ビットの巡回シフトがあるかないかであり，**図 1** に示したメッセージ圧縮関数は両者で共通である．

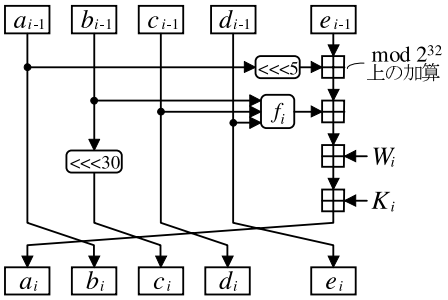

図 1 SHA-0/-1 のメッセージ圧縮関数
Fig. 1 Message compression function of SHA-0/-1.

このように，16 ワードから 80 ワードに拡大したメッセージを，圧縮関数を 80 ラウンド繰り返しながら 5 ワードに圧縮するのが SHA-1/-0 の骨子である．次章では，SHA-1 の Collision を生成するために，この圧縮関数の性質を利用しながら，2 つの異なるメッセージのビットパターンを 80 ラウンド中で巧妙に制御する Wang の攻撃法について解説する．

3. Wang の SHA-1 攻撃法

本章では，Wang らによる SHA-1 の攻撃法[19)〜22)]を簡単に説明する．

Chabaud と Joux は**表 2** に示したように，$W_{i,j}$（i 番目の 32 ビットメッセージワード W_i の j 番目のビット）の反転を，それに続く 5 つの連続するメッセージワードの補完的なビット反転によりキャンセルすることで，SHA-0 に対する 6 ステップの Local Collision を作り出せることを示した[11)]．Wang らの攻撃法はこの Local Collision を利用しており，各ステップにおける差分 $2^{j+30 \bmod 32}$ は主に 2^{31}，つまり MSB となるように $j = 1$ が選ばれる．これは差分が加算によって上位桁に伝播し，多くのビットに影響を与えるのを避けるためである．

この Local Collision はステップ i によって変わる関数 $f_{i+2} \sim f_{i+5}$ に応じて，$2^{-2} \sim 2^{-5}$ の確率で生成することができる．また，そのためにはメッセージワードに次の条件が要求される．

$$W_{i,2} = \neg W_{i+1,7} \quad i = 20 \sim 39, 60 \sim 79 \quad (9)$$

$$W_{i,2} = \neg W_{i+2,2} \quad i = 40 \sim 59 \quad (10)$$

文献 20) に示された SHA-0 の Collision のための差分パスは，上記の 6 ステップの Local Collisions を複数重ね合わせたもので，各 Local Collision の開始位置を示した 80 ビットの系列（$x_0, \ldots x_{79}$）は Distur-

表 2 SHA-0 における 6 ステップの Local Collision
Table 2 A 6-step Local Collision for SHA-0.

Step	ΔW	Δa	Δb	Δc	Δd	Δe	条件
i	2^j	2^j					NC (No Carry)
i+1	$2^{j+5 \bmod 32}$		2^j				
i+2	2^j			$2^{j+30 \bmod 32}$			NC, $\Delta f_{i+2} = 2^j$
i+3	$2^{j+30 \bmod 32}$				$2^{j+30 \bmod 32}$		NC, $\Delta f_{i+3} = 2^{j+30 \bmod 32}$
i+4	$2^{j+30 \bmod 32}$					$2^{j+30 \bmod 32}$	NC, $\Delta f_{i+4} = 2^{j+30 \bmod 32}$
i+5	$2^{j+30 \bmod 32}$						NC, $\Delta f_{i+5} = 2^{j+30 \bmod 32}$

表 3 SHA-1 におけるハミング重み（HW）の小さい Disturbance Vector
Table 3 Disturbance vectors with low Hamming Weights (HW) for SHA-1.

i	x_i	HW	i	x_i	HW	i	x_i	HW	i	x_i	HW	i	x_i	HW
			0	40000001	2	20	3	2	40	0	0	60	0	0
			1	2	1	21	0	0	41	0	0	61	0	0
			2	2	1	22	2	1	42	2	1	62	0	0
			3	80000002	2	23	2	1	43	0	0	63	0	0
			4	1	1	24	1	1	44	2	1	64	4	1
			5	0	0	25	0	0	45	0	0	65	0	0
			6	80000001	2	26	2	1	46	2	1	66	0	0
			7	2	1	27	2	1	47	0	0	67	8	1
			8	2	1	28	1	1	48	2	1	68	0	0
			9	2	1	29	0	0	49	0	0	69	0	0
			10	0	0	30	0	0	50	0	0	70	10	1
			11	0	0	31	2	1	51	0	0	71	0	0
			12	1	1	32	3	2	52	0	0	72	8	1
			13	0	0	33	0	0	53	0	0	73	20	1
			14	80000002	2	34	2	1	54	0	0	74	0	0
-5	80000000	1	15	2	1	35	2	1	55	0	0	75	0	0
-4	2	1	16	80000002	2	36	0	0	56	0	0	76	40	1
-3	0	0	17	0	0	37	0	0	57	0	0	77	0	0
-2	80000001	2	18	2	1	38	2	1	58	0	0	78	28	2
-1	0	0	19	0	0	39	0	0	59	0	0	79	80	1
小計		4	小計		21	小計		14	小計		4	小計		9

bance Vector と呼ばれる．この系列は式 (3) のメッセージ拡大関数に沿って作られるので，連続する 16 ビットを決めると残りのビットは一意に定まることになる．文献 21) の SHA-1 に対する差分パスも Local Collision の重ね合わせによって作られているが，式 (4) のメッセージ拡大関数の巡回シフトにより，1 ビットの差分が 32 ビットワード中に展開されていくため，Disturbance Vector は 80 ビットではなく 80 ワードとなる．

ここで，ハミング重みの小さい，つまり重ね合わせる Local Collision の数が少ない Disturbance Vector を探すことが，攻撃手順の複雑性の低減にきわめて重要である．しかしながら，512（$= 80 \times 32$）ビットの空間をくまなく調べるわけにはいかず，また初期メッセージ M_i に対するいかなる 1 ビットの差分も，拡大されたメッセージ W_i の少なくとも 107 ビットに影響することが知られている[24]．そこで，Wang らはメッセージ M_i に 1 ビットの差分を入れるのではなく，16 ワードのベクタ {2,0,0, . . . ,0} からスタートし，式 (4) を進めたり戻したりしながらハミング重みの小さい系列を検索し，**表 3** に示したような 5 ワード（$i = -5 \sim -1$）に続く 80 ワード（$i = 0 \sim 79$）を Disturbance Vector として選択している[21]．

表 3 においてビット $x_{48,1}$ に挿入された差分（$= 2$）は，SHA-1 のメッセージ拡大時の巡回シフトによって，多くのビット位置に影響を与えることになる．そして，この差分の拡散は Local Collision が満たすべき条件の数を増大させてしまう．それに対処するために，ワード内の差分を 0 と 1 の反転としてとらえた XOR 差分ではなく，整数値の差としてとらえた算術差分を導入する．これは，たとえば連続する 2 ビット $x_{i,j+1}$ と $x_{i,j}$ に差分がある場合，メッセージ差分 $\Delta W_i = W'_i - W_i$ の対応する 2 ビットの符号を $\Delta W_{i,j+1} = 2^{j+1}$ と $\Delta W_{i,j+1} = -2^j$ のように反対にするものである．これにより差分は $2^{j+1} - 2^{j+1} = 2^j$ のように 1 ビットにすることができ，考慮すべき条件の数を減らすことができる．

表 4 は SHA-1 の Near Collision を生成する差分パ

表 4 SHA-1 の Near Collision に対する差分パス
Table 4 Differential path for SHA-1 Near Collision.

i	x_{i-1}	ΔW_{i-1}	Δa_i		Δb_i	Δc_i	Δd_i	Δe_i
			No carry	With carry				
1	40000001	2^{30}, 2^{29}	2^{30}, 2^{29}	2^{30}, 2^{29}				
2	2	2^{31}, -2^{30}, -2^{29} 2^5, -2^3, -2^1	2^{29} 2^5, 2^1	$2^{31}-2^{30}-2^{29}$ $2^7-2^6-2^5$, 2^2-2^1	Δa_1			
3	2	2^{29}, -2^6 2^1, 2^0	2^{10} 2^3, -2^0	$2^{13}-2^{12}-2^{11}-2^{10}$ 2^3, -2^0	Δa_2	$\Delta a_1^{<<<30}$		
4	80000002	-2^{31}, -2^{29} 2^{28} 2^6	-2^{31} 2^{15} 2^8, -2^1	-2^{31}, $2^{18}-2^{17}-2^{16}-2^{15}$ 2^8, -2^1	Δa_3	$\Delta a_2^{<<<30}$	$\Delta a_1^{<<<30}$	
5	1	2^{31}, 2^{30}, 2^{28} 2^6,-2^4,-2^1, 2^0	2^{27}, 2^{20} 2^5, -2^4	2^{27}, $2^{21}-2^{20}$ 2^5, -2^4	Δa_4	$\Delta a_3^{<<<30}$	$\Delta a_2^{<<<30}$	$\Delta a_1^{<<<30}$
6	0	2^{31}, 2^{30}, 2^{28} -2^5, -2^1	2^{25}, 2^{15} 2^{10}	$2^{26}-2^{25}$, $2^{16}-2^{15}$ $2^{12}-2^{11}-2^{10}$	Δa_5	$\Delta a_4^{<<<30}$	$\Delta a_3^{<<<30}$	$\Delta a_2^{<<<30}$
7	80000001	2^{29}	2^{31}, -2^5, -2^3, 2^0	2^{31}, -2^6+2^5 -2^3, 2^0	Δa_6	$\Delta a_5^{<<<30}$	$\Delta a_4^{<<<30}$	$\Delta a_3^{<<<30}$
8	2	2^{30}, 2^{29} -2^5, -2^4, -2^1	-2^{18}	$-2^{25}+2^{24}+\ldots+2^{18}$	Δa_7	$\Delta a_6^{<<<30}$	$\Delta a_5^{<<<30}$	$\Delta a_4^{<<<30}$
9	2	-2^{30}, -2^{29} -2^6, -2^1, 2^0	-2^9 -2^1	$-2^{19}+2^{18}+\ldots+2^9$ -2^1	Δa_8	$\Delta a_7^{<<<30}$	$\Delta a_6^{<<<30}$	$\Delta a_5^{<<<30}$
10	2	-2^{29}, 2^6	2^1	2^1	Δa_9	$\Delta a_8^{<<<30}$	$\Delta a_7^{<<<30}$	$\Delta a_6^{<<<30}$
11	0	-2^{31}, 2^{30}, 2^{29} -2^6, 2^1	2^8	2^9-2^8	Δa_{10}	$\Delta a_9^{<<<30}$	$\Delta a_8^{<<<30}$	$\Delta a_7^{<<<30}$
12	0	-2^{30}, -2^{29},-2^1	-2^3	-2^3	Δa_{11}	$\Delta a_{10}^{<<<30}$	$\Delta a_9^{<<<30}$	$\Delta a_8^{<<<30}$
13	1	-2^{30}, 2^0	2^0	2^0	Δa_{12}	$\Delta a_{11}^{<<<30}$	$\Delta a_{10}^{<<<30}$	$\Delta a_9^{<<<30}$
14	0	-2^5			Δa_{13}	$\Delta a_{12}^{<<<30}$	$\Delta a_{11}^{<<<30}$	$\Delta a_{10}^{<<<30}$
15	80000002	2^1, -2^0	-2^{31}	-2^{31}		$\Delta a_{13}^{<<<30}$	$\Delta a_{12}^{<<<30}$	$\Delta a_{11}^{<<<30}$
16	2	-2^{30},-2^6,2^4,2^1	2^1	2^1	Δa_{15}		$\Delta a_{13}^{<<<30}$	$\Delta a_{12}^{<<<30}$
17	80000002	2^{30}, -2^6	2^{31}, -2^1	2^{31}, -2^1	Δa_{16}	$\Delta a_{15}^{<<<30}$		$\Delta a_{13}^{<<<30}$
18	0	2^{31}, 2^{30}, 2^{29} 2^6, -2^4, -2^1			Δa_{17}	$\Delta a_{16}^{<<<30}$	$\Delta a_{15}^{<<<30}$	
19	2	2^{31}, 2^{29}	2^1	2^1		$\Delta a_{17}^{<<<30}$	$\Delta a_{16}^{<<<30}$	$\Delta a_{15}^{<<<30}$
20	0	2^{31}, -2^6			Δa_{19}		$\Delta a_{17}^{<<<30}$	$\Delta a_{16}^{<<<30}$

スの一部で，また**表 5** は表 4 のメッセージ差分 $\Delta W_0 \sim \Delta W_5$ を例に，それが Disturbance Vector $x_{-5} \sim x_5$ で指定された Local Collision の重ね合わせによってどのように定まるのかを示している．各ベクタ x_i からは，6 もしくは 12 項の差分が生成される．たとえば，$x_{-4} = 2 = 2^1$ からは 6 項の差分 $\{2^1, 2^6, 2^1, 2^{31}, 2^{31}, 2^{31}\}$ が，$x_{-2} = 800000001 = 2^{31} + 2^0$ からは 12 項の差分 $\{2^{31}, 2^4, 2^{31}, 2^{29}, 2^{29}, 2^{29}\} + \{2^0, 2^5, 2^0, 2^{30}, 2^{30}, 2^{30}\}$ が生じている．いくつかの Local Collisions はオーバラップしているので，それらを各ステップ i ごとに縦方向に足し合わせたものが最終的なメッセージ差分 ΔW_i となる．表 5 では簡単のため ΔW_i を XOR 差分で示しているが，本来求めるべき表 4 のような算術差分はメッセージワード W'_i と W_i の対応するビットの 0 と 1 を適切に設定することで簡単に作ることができる．

差分パスに挿入された ΔW_{i-1} は 5 ワードの作業変数 $a \sim e$ に伝播し，そこに差分 Δa_i，Δb_{i+1}，Δc_{i+2}，Δd_{i+3}，Δe_{i+4} が発生する．式 (6) および図 1 から分かるように差分 $\Delta b_{i+1} \sim \Delta e_{i+4}$ は a_i によって決まり，また $\Delta a_i \sim \Delta e_i$ は Δa_{i+1} に影響を与える．したがって，この差分 Δa_{i+1} へのフィードバックの影響をいかに減らすかが Collision の生成において重要となる．差分 Δb_i，Δc_i，Δd_i（$i = 0 \sim 19$）は式 (1) のビット演算 $f_i(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$ の性質を利用することで制御可能なので，差分 Δe_i をキャンセルするために，キャリー伝播を利用して差分 Δa_{i-1} をビット拡張する．たとえば，表 3 において $\Delta a_8 = -2^{18}$ は $-2^{25} + 2^{24} + \ldots + 2^{18}$ とビット拡張され，Δb_9 へと渡される．そのとき差分 Δe_9 は $\Delta a_5 <<< 30 = (2^{27} + 2^{21} - 2^{20} + 2^5 - 2^4) <<< 30 = 2^{25} + 2^{19} - 2^{18} + 2^3 - 2^2$ であり，そこに含まれる 2^{25} と -2^{18} という項は Δb_9 の -2^{25} と 2^{18} によってキャンセルされる．このようなビット拡張によるキャンセルは，XOR ではなく算術演算による差分評価を導入した効果によるものである．

表 5 SHA-1 の Near Collision に対する差分パス
Table 5 Message differentials for Local Collisions traced from disturbance vector.

Disturbance Vector		メッセージ差分 ΔW_i										
		ΔW_{-5}	ΔW_{-4}	ΔW_{-3}	ΔW_{-2}	ΔW_{-1}	ΔW_0	ΔW_1	ΔW_2	ΔW_3	ΔW_4	ΔW_5
x_{-5}	80000000	2^{31}	2^4	2^{31}	2^{29}	2^{29}	2^{29}					
x_{-4}	2		2^1	2^6	2^1	2^{31}	2^{31}	2^{31}				
x_{-3}	0			0	0	0	0	0	0			
x_{-2}	80000001				$2^{31},2^0$	$2^4,2^5$	$2^{31},2^0$	$2^{29},2^{30}$	$2^{29},2^{30}$	$2^{29},2^{30}$		
x_{-1}	0					0	0	0	0	0	0	
x_0	40000001						$2^{30},2^0$	$2^3,2^5$	$2^{30},2^0$	$2^{28},2^{30}$	$2^{28},2^{30}$	$2^{28},2^{30}$
x_1	2							2^1	2^6	2^1	2^{31}	2^{31}
x_2	2								2^1	2^6	2^1	2^{31}
x_3	80000002									$2^{31},2^1$	$2^4,2^6$	$2^{31},2^1$
x_4	1										2^0	2^5
x_5	0											0
Exclusive-ORed summation for ΔW_i		2^{31}	$2^4,2^1$	$2^{31},2^6$	$2^{31},2^{29}$ $2^1,2^0$	$2^{31},2^{29}$ $2^5,2^4$	$2^{30},2^{29}$	$2^{31},2^{30}$ $2^{29},2^5$ $2^3,2^1$	$2^{29},2^6$ $2^1,2^0$	$2^{31},2^{29}$ $2^{28},2^6$	$2^{31},2^{30}$ $2^{28},2^6$ $2^4,2^1,2^0$	$2^{31},2^{30}$ $2^{28},2^5$ 2^1

表 6 SHA-1 の差分パスにおける作業変数 a_i が満たすべき条件
Table 6 Conditions on chaining variable a_i for differential path of SHA-1.

i	ビット $a_{i,j}$ の条件			
	31 ... 24	23 ... 16	15 ... 8	7 ... 0
1	α00-----	--------	1-----$\alpha\alpha$	1-0α11$\alpha\alpha$
2	01110---	------1-	0$\alpha\alpha\alpha$-0--	011-001-
3	0-100---	-0-$\alpha\alpha\alpha$0-	--0111--	01110-01
4	10010---	α1---011	10011010	10011-10
5	00100---	--01-000	10001111	-010-11-
6	1-0-0011	1-1001-0	111011-1	α10-000-
7	0---1011	1α0111--	101--010	-10-11-0
8	-01---10	000000$\alpha\alpha$	001$\alpha\alpha$111	---01-1-
9	-00-----	10001000	0000000-	---11-1-
10	0-------	1111111-	11100000	0-----0-
11	--------	------10	11111101	1-α--0--
12	0-------	--------	--------	10--11--
13	--------	--------	--------	11----10
14	-0------	--------	--------	----0-1-
15	10------	--------	--------	----1-0-
16	--1-----	--------	--------	----0-0-
17	0-0-----	--------	--------	------1-
18	--1-----	--------	--------	----α---
19	--β-----	--------	--------	------0-
20	--------	--------	--------	-----α--

(α: $a_{i,j} = a_{i-1,j}$, β: $a_{19,29} = a_{18,31}$)

表 6 は表 4 の差分パスが Collision を生じるための作業変数 a_i の条件を示しており，表中の 'α' は $a_{i,j} = a_{i-1,j}$ を 'β' は $a_{19,29} = a_{18,31}$ を意味している．たとえば，$a_{8,25} \sim a_{8,18} = 10000000 (= 2^{25})$ は $\Delta a_8 = -2^{18}$ が $-2^{25} + 2^{24} + \ldots + 2^{18}$ に拡張されるための条件であり，これにより $\Delta a_8 = a'_8 - a_8$ は -2^{25} の項を持ち，必然的に残りの項 $+2^{24} + \ldots + 2^{18}$ も含むことになる．

さらに Wang らは表 6 の条件を満たすために，Message Modification と呼ぶ以下のような手法を導入している．a_i （$i = 1 \sim 16$）は式 (3) と (6) によって，作業変数 $a_{i-1} \sim e_{i-1}$ と入力メッセージワード M_i から次式で計算される．

$$a_i = (a_{i-1} <<< 5) + f_{i-1}(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + M_{i-1} + K_{i-1} \quad (11)$$

したがって，ビット $a_{i,j}$ に対する条件が満たされていない場合は，ビット $M_{i-1,j}$ を反転すればよいことが分かる．このビット反転はそれ以降の作業変数の値に影響を与えるので，$i = 1 \rightarrow 16$ の順番に行う必要がある．またビット $M_{i-1,j}$ を反転すると，キャリーの伝播によって a_i の上位ビットが影響を受けることが考えられるが，これはメッセージワードを適切に選ぶこ

とで回避することが可能である．ステップ 17〜20 の条件および式 (9)，(10) から導出される別の条件も同様の手法で満足させることができる．たとえば，条件 $a_{17,31} = 0$ が満たされていないときは $W_{16,31}$ でなく $M_{15,26}$（$= W_{15,26}$）を変更して，まずビット $a_{16,26}$ を反転させる．これによって次のステップで $a_{17,31}$ が反転することになる．このような Message Modification により，ステップ 17〜22 の条件もすべて満たすことができる．

Wang らの SHA-1 の攻撃は，式 (12) のように 2 つの連続する 512 ビットのメッセージブロック $M0$ と $M1$ を用いる．なお，ここで $\Delta H0$ は式 (2) の初期値に対する差分なので 0 となる．

$$\begin{aligned}&\Delta H0(=0)\\ &=^{<M0,M0'>}\!\!\longrightarrow \Delta H1 \;^{<M1,M1'>}\!\!\longrightarrow \Delta H(=0)\end{aligned} \quad (12)$$

そして，最初のメッセージブロックの組 $< M0, M0' >$ は次のような差分 $\Delta H1$ を持つ Near Collision を生成する．

$$\begin{aligned}\Delta H1 &= H1' - H1\\ &= (h0' + H0) - (h0 + H0)\\ &= h0' - h0 = \Delta h0\end{aligned} \quad (13)$$

ここで $\Delta h0$ は式 (14) に示したように，$M0$ と $M0'$ に対するハッシュ演算の最終ステップにおける作業変数の差分である．この作業変数は式 (8) においてハッシュ値を更新するのに用いられる．

$$\begin{aligned}\Delta h0 &= h0' - h0\\ &= (a'_{80}, b'_{80}, c'_{80}, d'_{80}, e'_{80})_{M0'}\\ &\quad -(a_{80}, b_{80}, c_{80}, d_{80}, e_{80})_{M0}\end{aligned} \quad (14)$$

次に 2 番目のメッセージブロック対 $< M1, M1' >$ で $\Delta h0$ を打ち消すような差分 $\Delta h1$ が得られれば，式 (15) のように差分 0 の真の Collision を作り出すことができる．

$$\begin{aligned}\Delta H &= (h1' + H1') - (h1 + H1)\\ &= (h1' - h1) + (H1' - H1)\\ &= \Delta h1 + \Delta h0 = 0\end{aligned} \quad (15)$$

ハッシュ演算の最後の 5 ステップで使われる表 3 の 4 つの Disturbance Vector（40，20，8，80）は，第 1 ブロックから第 2 ブロックへと差分 $\Delta h0$（$= \Delta H1$）を通じて伝播していく．差分 $\Delta h0$ の影響は先に説明したビット演算やキャリーの効果によって，最初の 16 ステップで消すことができる．そして第 2 メッセージブロック $M1$ に対しては，式 (15) に示したように $\Delta h0$ と $\Delta h1$ の各ビットが打ち消し合うように条件が設定される．このとき第 2 ブロックに対する条件の数が増えることはないので，処理の複雑さは第 1 ブロックと同じである．

Message Modification によってステップ 1〜22 の条件が満たされたとき，ステップ 23〜80 には式 (9)，(10) から導かれる 73 個の条件がまだ残っている．ステップ 78 は条件を含まず，また最後の 2 ステップの 3 つの条件は差分 $\Delta h0$ あるいは $\Delta h1$ を持つ Near Collision の生成において無視することができる．したがって，1 ブロックの Near Collision を生成するためには 70 個の条件を満たせばよい．それにはまず，最初の 10 ステップの条件を満たすようなメッセージワード $M0_0 \sim M0_9$（そして $M0'_0 \sim M0'_9$）を求め，またそれに続く 6 ワード $M0_{10} \sim M0_{15}$ は，Near Collision が見つかるまで次々に変更を加えるフリー変数とする．このフリー変数を更新するたびに，ステップ 11〜22 の条件が満たされるように Message Modification を行う．それに続いてステップ 23 と 24 の 5 つの条件[25]をチェックし，それが満たされていなければフリー変数を変更するためにステップ 10 に戻る．また確率 2^{-5} で満たされた場合は残りの 56 ステップの SHA-1 処理を実行し，その結果として Near Collision が得られたかどうかを調べる．ステップ 11〜24 では 14 回の Message Modification が必要とされるので，ここまでフリー変数の更新ごとに平均で $14 + 56 \times 2^{-5} \cong 16$ ステップのハッシュ処理を行うことになる．またこれに加えてステップ 17〜22 の条件を満たすために，$M0_{16} \sim M0_{21}$ ではなく $M0_{10} \sim M0_{15}$ への 6 回の Message Modification を行う必要がある[25]．したがって，フリー変数の変更 1 回につき 16+6=22 ステップのハッシュ処理を行うことになり，これは 80 ステップの SHA-1 のおよそ 1/4 の演算量となる．以上の議論から，確率 1/2 で成立するステップ 23〜77 の 70 の条件をすべて満たす Near Collision が見つかるまでの計算量は，$2^{70} \times 1/4 = 2^{68}$ 回の SHA-1 演算に相当する．Wang らの攻撃では，2 つのメッセージブロックに対する Near Collision によって真の Collision を作り出すので，全体でおよそ $2^{68} \times 2 = 2^{69}$ 回の SHA-1 に相当する演算が必要となる．

また Wang らは攻撃法を改良し，計算量のオーダを 2^{63}，つまり $2^{63}/2^{69} = 1/64$ に下げることができるという報告も行っている[22]．これは Collision 生成の差分パスが異なるだけで，2^{69} オーダの場合と同じ手法を用いるものと見られるが，詳細はまだ公開されていない．そこで，次章では 2^{69} オーダの攻撃法に基づくハードウェア・アーキテクチャを提案する．

4. SHA-1 攻撃用ハードウェア

3 章で述べたように Wang らの攻撃に必要な 2^{69} 回のハッシュ演算とは，単に SHA-1 演算を 2^{69} 回繰り返すことではなく，Message Modification などを含めた全体の演算量が SHA-1 の 2^{69} 回に相当するという意味である．**図 2** はそれらを実行する SHA-1 攻撃用のハードウェア・アーキテクチャを示しており，コアの部分は筆者らが文献 26) において提案した小型・高速な SHA-1 回路をベースにしている．

16 ワードのメッセージ M_i（$i = 0 \sim 15$）は左側のポートからワードごとにメッセージメモリに入力される．そして入力メッセージは，16 ワードレジスタ，XOR，巡回シフトから構成されるメッセージ拡大部で，式 (4) に従って 80 ワード W_i（$i = 0 \sim 79$）に拡大された後，次々とメッセージ圧縮部へと送られる．

データ圧縮部はキャリー先見加算器（CLA: Carry Look-ahead Adder）とキャリー保存加算器（CSA: Carry Save Adder）2 種類の加算器を用いている．加算は mod 2^{32} 上で行われるので，33 ビット目にあたる MSB のキャリーを計算する必要はない．CSA は非常に高速であるが，その出力は 64 ビットの冗長 2 進数となるため途中結果の計算に使用し，32 ビットレジスタ a への入力を求めるときに CLA で 32 ビットの 2 進数に戻している．データパスは式 (8) のハッシュ値更新のための加算器を含んでいないが，この更新が必要となるのは最初のメッセージブロックに対して確率 2^{-68} で Near Collision が見つかったときだけなので，これはソフトウェアで行ったほうがコストの面で有利だからである．Near Collision が見つかったならば，作業変数 $a \sim e$ は *Dout* から出力され，ハッシュ値の更新を含む第 2 ブロックに対するパラメータの設定は外部ソフトウェアで処理される．

表 4 に示したような差分 $\Delta W_0 \sim \Delta W_{15}$ を持つ任意のメッセージ対 $< M0, M0' >$ を選んだならば，表 6 の $a_1 \sim a_{10}$ に対する条件が満たされるように Message Modification を行う．これは 1 ブロックの Near Collision を見つけるための 2^{68} 回のハッシュ演算に先立って 1 回だけ行えばよいので，これも外部のソフトウェアで実行する．そして修正されたメッセージブロック $M0$' と，それに対応する作業変数 $a_{10} \sim e_{10}$ は，2 組の 5 ワードレジスタ $a \sim e$ と $a_{10} \sim e_{10}$，の双方にセットされる．次いで，条件判定回路と 10 ビット ×42 のメモリにセットされた条件データにより，ステップ 11〜22 の 42 回の Message Modification が行われる．表 6 の各条件を表現するには，4 種類の条件 $< 0,\ 1,\ \alpha,\ \beta >$ を区別するのに 2 ビット，32 ビットワード a 中のビット位置の指定に 5 ビット，そして現在のステップ i から数えて何ステップ目なのかという距離（0〜7）を表すのに 3 ビットの計 10 ビットあれば十分である．たとえば条件 $a_{11,8}$=1 は {01 01000 000} のように表現され，これはビットの条件が 1，そのビット位置が 8，そして距離 0 は現在のステップ i=11 で修正されるべき条件であることを示している．次の条件 $a_{11,9}$=0 は {00 01001 000} と表現されており，これはビット条件が 0，ビット位置 9，そして距離 0 からやはり現在のステップ i=11 であることが分かる．これら表 6 の条件 $a_{i,j}$（$i = 11 \sim 22$）をすべて保持するには，サイズ 10 ビット ×42 のメモリで十分である．同じステップ i に属するすべての条件は同時にチェックされるが，このとき 32 ビットの信号 X は満たされていない条件のビット位置に 1 がセッ

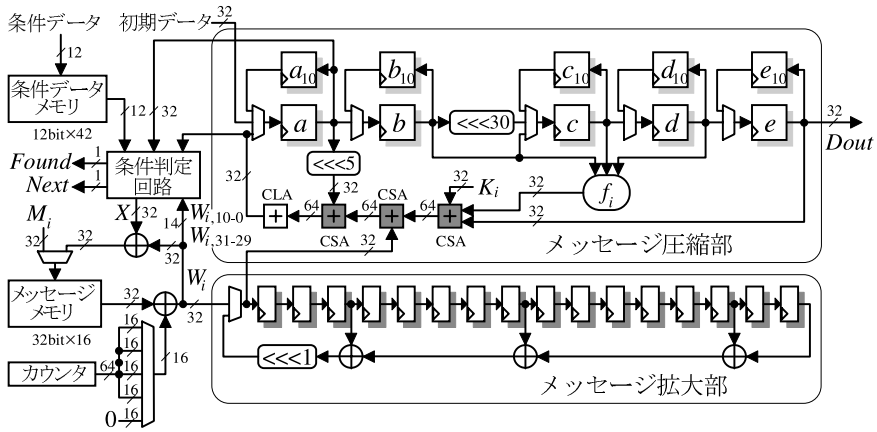

図 2 SHA-1 攻撃用ハードウェアのアーキテクチャ
Fig. 2 Hardware architecture for SHA-1 attack.

表 7 0.13 μm CMOS スタンダードセル・ライブラリによる SHA-1 のコアのハードウェア性能
Table 7 Hardware performance of SHA-1 core using a 0.13 μm CMOS standard cell library.

面積		最大遅延パス	動作周波数	処理サイクル数	処理時間	最適化
(gates)	(mm^2)*	(ns)	(MHz)		(ns/Block)	
8,266	0.0794	6.45	155.0	80	516	面積
10,242	0.0983	2.50	400.0	80	200	速度

*80%の配線効率を仮定

トされ，これをメッセージワード W_i と XOR することでメッセージの修正つまり Message Modification が実行される．Message Modification が行われているとき，レジスタ a とメッセージメモリは次々と更新されるのに対し，レジスタ $b \sim e$ とメッセージ拡大部のレジスタは変更されない．条件 $a_{17} \sim a_{22}$ を満足するには，ステップ 10〜15 に戻ってメッセージワード $W_{10} \sim W_{15}$ を修正する必要があり，そのときのビット反転の影響は a_{10} から次のステップへと伝播してゆく．5 ワードのレジスタ $a_{10} \sim e_{10}$ は，このように前のサイクルに戻るときのために用意されている．

ステップ 22 までの修正が終わったならば，残りの 58 ステップを順に実行しながら Near Collision を生成するための条件がすべて満たされているかどうかをチェックする．Wang らの手法では，ステップ 23 と 24 の条件が満たされていた場合は，最終ステップまでの処理が実行され，その結果として Near Collision が生成されたかどうかをチェックする．それに対して図 2 のハードウェアでは，すべてのステップにおいて条件をチェックするので，1 つでも満たされていなければ条件判定回路が信号 *Next* を上げて現在の処理を停止し，新たなメッセージブロックに対する検索をただちに始めることができる．ステップ 23〜78 では Joux らの 6 ステップの Local Collision の式 (9) と (10) から導かれるメッセージワード W_i に関する条件が適用され，$W_{i,j}$（$j = 31 \sim 29, 10 \sim 0$）の 14 ビットがチェックの対象となる．これらの式から導かれる条件は規則正しいため，表 6 に示した条件のようにメモリに保存する必要はない．文献 21) ではこれら条件がすべて列挙されているわけではないが，それらを含めても条件判定はシンプルでコンパクトな回路で実装することができる．

信号 *Next* が上がると新たな検索を始めるために，レジスタ $a \sim e$ はレジスタ $a_{10} \sim e_{10}$ 内のデータで初期化される．フリーな 6 ワード $M_{10} \sim M_{15}$ は 64 ビットカウンタによって更新するが，このとき複数のハードウェアマクロを用いて，Collision の並列検索を行うので，2 ワード M_{10} と M_{11} はマクロ間で検索領域を分離するために用い，残りの 4 ワード $M_{12} \sim M_{15}$ を 64 ビットカウンタ出力と 16 ビットずつ XOR する．1 つのマクロで 2^{68} 回のハッシュ演算を行うわけではないので，個々のマクロは 2^{64} ブロックのメッセージが検索できれば十分である．ところで，これらのメッセージの更新で変更されたビットが，それに続く Message Modification で元に戻され，同じメッセージブロックを再度検索してしまう可能性がある．これを避けるためにカウンタ出力の XOR は，表 6 で条件が何も課せられていない $M_{12} \sim M_{15}$ のビット $j = 8 \sim 23$ に対して実行する．

確率 2^{-68} で最初の Near Collision が見つかると，信号 *Found* が立ち上がり，レジスタ $a \sim e$ の 5 ワードのデータはポート *Dout* を通じてハードウェアの外部に出力される．そしてソフトウェアによってそのデータから 2 番目のメッセージブロックに対する初期データと条件を生成し，それを複数の SHA-1 攻撃マクロに同時に設定することで，2 番目の Near Collision の検索をすべてのマクロで同時に開始される．

次に SHA-1 攻撃マクロの性能を評価するが，ステップ 23〜78 の条件が詳細に示されていないので，条件判定回路の部分を正確に見積もることは困難である．そこで，メッセージ拡大部とメッセージ圧縮部を含む主要部のデータパスと，普通の SHA-1 演算を含む制御回路を設計した．しかし，最大遅延パスには条件判定回路などが含まれている．そこで速度の見積りをできるだけ正確にするために，これらの部分にはダミーの比較回路などを挿入して評価を行った．**表 7** は SHA-1 攻撃回路のコアを，0.13 μm の CMOS スタンダードセル・ライブラリ[27] で面積優先と速度優先の 2 つのオプションで論理合成したときの性能を示している．

論理合成ツールはゲート素子の駆動能力を大きくしたり，並列処理を行ったりすることで処理速度を向上させる．このとき増加する回路は主に演算部であるが，SHA-1 のコアでは回路規模の大小にかかわらず速度性能にほとんど影響しないレジスタが大きなエリアを占めている．このため，速度優先の実装は面積優先の 2 倍以上速いにもかかわらず，回路規模はわずか 25%の増加にとどまっている．

表 7 に含まれない，メッセージや条件データを保持

する 12 ビット ×42 と 32 ビット ×16 の 2 つのメモリはレジスタアレイとして実装するが，その規模は両者でおよそ 10 K ゲートとなる．したがって，これらのメモリと条件判定回路などを含めても，速度優先の実装が全体で 30 K ゲートを超えることはない．次章では，この速度優先の実装をベースに，SHA-1 を攻撃するためのコストと処理時間の見積りを行う．

5. コストと処理時間

速度優先の SHA-1 攻撃マクロは 1 つのメッセージブロックを 2.5 ns × 80 サイクル = 200 ns で処理するので，64 個のマクロを 1 チップに実装した場合，1 日に処理できるブロック数は次のようになる．

$$24 \times 3{,}600\,\mathrm{s}/200\,\mathrm{ns} \times 64 = 2.76 \times 10^{13} \quad (16)$$

64 個のマクロの回路規模は合計で 30 K ゲート × 64=1.92 M ゲートであり，通常の配線効率は 90%以上であるが，ここで低めに 80%と仮定しても面積はわずか 18.4 mm^2 である．したがって 1 チップのサイズは，I/O や周辺回路を含めて 25 mm^2 も見積もれば十分である．LSI チップの価格は回路面積だけでなく様々な要因に左右されるが，このような小さなチップは開発費を除外すれば，大量生産時で 50 ドル以下と見てよい．0.13 μm CMOS プロセスの消費電力はおよそ 9 nW/MHz/gate なので[27]，この SHA-1 攻撃チップの消費電力は次のようになる．

$$9 \times 10^{-9}\,\mathrm{W/MHz/gate} \times 400\,\mathrm{MHz} \times 1.92 \times 10^{6}\,\mathrm{gates} = 7\,\mathrm{W} \quad (17)$$

32 個のチップを USB インタフェースの PC ボードに実装するとき，ボードのコストは基盤や周辺回路を含めておよそ 2,000 ドルと見積もれる．消費電力はおよそ 7 W×32 = 224 W なので，20 ドルほどの安価な電源ユニットで供給することができる．また，外部から PC ボードに対する制御は，2 つの Near Collision を探すために初期設定を 2 回行うだけなので，性能の低い 400 ドル程度の PC でも十分である．USB インタフェースで 127 個のクライアントを接続することができるが，ここでは扱いやすさの観点から，16 枚のボードを 1 台の PC につなぐものとする．この SHA-1 攻撃用の PC システムのコストは 1 セットがおよそ 33,000 ドルで，1,000 万ドルの予算があれば，ランニングコストを除いて 303（$= 10^7/33{,}000$）セット用意することができる．そして SHA-1 を破るための 2^{69} のハッシュ処理は，次式に示したように 138 日で実行することができる．

$$2^{69}\text{回}/(2.76 \times 10^{13}\text{回/日} \times 512\ \text{チップ} \times 303\ \text{セット}) = 138\ \text{日} \quad (18)$$

さらに Wang らの計算量 2^{63} オーダの新たな攻撃法が[22]，提案アーキテクチャに適用可能であれば，1/10 の予算の 100 万ドルで SHA-1 をわずか

$$2^{63}\text{回}/(2.76 \times 10^{13}\text{回/日} \times 512\ \text{チップ} \times 30\ \text{セット}) = 22\ \text{日} \quad (19)$$

で破ることができる．

超並列処理では，演算ユニット間のデータ転送，処理の同期，エラーの対処などが大きな問題となるが，本アーキテクチャでは各演算コアは独立に動作するので，これらの問題が生じることはない．PC はそれぞれの LSI（およびその中の 64 個の SHA-1 コア）に 10 ステップの Message Modification をほどこした第 1 メッセージブロック $M0'$（また 2 つ目の Near Collision 検索では $M1'$）と，それに対応する作業変数 $a10 \sim e10$，そして表 6 の条件などをセットし，検索範囲を分けるために SHA-1 演算コアごとにユニークな値を $M0'_{10}$ と $M0'_{11}$ に割り当てる．そしてその後は，いずれかのコアで Near Collision が見つかるのを待つだけである．いずれか 1 つのマクロが差分 $\Delta H1$ の最初の Near Collision を見つけたならば，次のメッセージブロック $M1'$ に対するパラメータを 303 個の PC システムすべてにセットし，式 (12) に示したように $\Delta H = 0$ とするための 2 つ目の Near Collision の検索を続行する．

共通鍵暗号 DES（Data Encryption Standard）を破った総当り攻撃では[28]，DES の処理を最大 2^{56} 回繰り返すことで，2^{56} 個の 56 ビット鍵の中から唯一の正しい鍵を探し出している．これに対してハッシュ関数では Collision を生成するメッセージ対は無数に存在するので，いずれかの Collision が確率的に見出されれば十分である．したがって，ある LSI（あるいは演算コア）に故障が起きて誤った計算が行われていても，その分の処理能力が削られるだけで，処理全体への影響はほとんどない．またハードウェアには演算のエラー検出機能も不要である．計算を誤った LSI（あるいはコア）が Collision を見つけたという信号を出したとき，その結果をソフトウェアで再検証して誤りが判明すれば，その LSI（あるいはコア）を以降の処理から外すことができる．

ムーアの法則[29] に従って，コンピュータ・システムの性能はおよそ 5 年ごとに 10 倍の向上を続けているため，ハッシュ関数の解読法に進展がなくとも，単純計算では 5 年後には 10 万ドルのコストとわずか 3 週間という時間で SHA-1 の Collision が発見されることになる．このように，本稿の結果は最も広く使用されているハッシュ関数 SHA-1 に対する大きな脅威

となるであろう．

6. むすび

本稿では，Wang らのアルゴリズムをベースとする SHA-1 攻撃専用ハードウェアのアーキテクチャを提案し，0.13 μm CMOS スタンダードセル・ライブラリによる LSI チップを複数用いた並列システムのコストと処理時間に対する考察を行った．64 個の SHA-1 コアと Wang のアルゴリズムの要である Message Modification が実装されたチップのサイズは 1.92 M ゲートで，400 MHz 動作時の消費電力は 7 W と見積もられ，1 日に 2.76×10^{13} 回のハッシュ処理を行うことができる．1,000 万ドルの予算があれば 32 チップを搭載した USB インタフェースのボードを 16 個つないだ PC システムを 303 セット用意することができる．このシステムでは，全体で 9,928,704 個のコアが並列に動作し，2 ブロック構成の SHA-1 の Collision を 138 日で生成可能である．さらに Wang らの最新の研究結果がそのまま適用できるならば，100 万ドルの予算とわずか 22 日の演算時間で Collision を見つけることが可能である．

ハードウェアの性能はムーアの法則に従って，5 年で 10 倍の向上を続けており，この事実も SHA-1 にとって大きな脅威となる．さらにハッシュ関数の攻撃に対する研究が急速に進んでいることから，より安全なハッシュ関数 SHA-224/-256/-384/-512 などへの移行が加速されるものと見られる．Rivest が MD4 を考案して以来，SHA-1 や SHA-224/-256/-384/-512 を含むほとんどのハッシュ関数が MD4 にならった構成をとっている．したがって，新たなコンセプトに基づくハッシュ関数の提案も今後重要な研究テーマの 1 つとなるであろう．

謝辞 本研究に対し貴重なコメントをいただいた Yiqun Lisa Yin 博士に感謝の意を表します．

参考文献

1) Yuval, G.: How to swindle Rabin, *Cryptologia*, Vol.3, No.3, pp.187–189 (1979).
2) NIST: Secure Hash Standard, FIPS PUB 180 (May 1993).
3) NIST: Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard (July 1994).
4) NIST: Secure Hash Standard, FIPS PUB 180-1 (Apr. 1995).
5) NIST: Secure Hash Standard (SHS), FIPS PUB 180-2 (Aug. 2002).
6) NIST: FIPS 180-2, Secure Hash Standard Change Notice 1 (Feb. 2004). http://csrc.nist.gov/publications/fips/fips180-2/FIPS180-2_changenotice.pdf
7) Rivest, R.L.: The MD4 Message Digest Algorithm, *CRYPTO '90*, LNCS 537, pp.303–311 (1991).
8) Rivest, R.L.: The MD4 Message Digest Algorithm, RFC 1186 (Oct. 1990).
9) Rivest, R.L.: The MD4 Message Digest Algorithm, RFC 1320 (Apr. 1992).
10) Rivest, R.L.: The MD5 Message Digest Algorithm, RFC 1321 (Apr. 1992).
11) Chabaud, F. and Joux, A.: Differential Collisions in SHA-0, Advances in Cryptology, *CRYPTO '98*, LNCS 1462, pp.56–71 (Feb. 1998).
12) Joux, A.: Collisions for SHA-0, *CRYPTO 2004* rump session (Aug. 2004). http://www.mail-archive.com/cryptography%40metzdowd.com/msg02554.html
13) Biham, E. and Chen, R.: Near-Collisions of SHA-0, Cryptology ePrint Archive: Report 2004/146 (Jun. 2004). http://eprint.iacr.org/2004/146
14) Biham, E. and Chen, R.: New results on SHA-0 and SHA-1, *Crypto 2004 Rump Session* (Aug. 2004).
15) Biham, E., Chen, R., Joux, A., Carribault, P., Lemmuet, C. and Jalby, W.: Collisions of SHA-0 and Reduced SHA-1, *Eurocrypt 2005*, LNCS 3494, pp.36–57 (May 2005).
16) Wang, X., Freng, D., Lai, X. and Yu, H.: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, Cryptology ePrint Archive: Report 2004/199 (Aug. 2004). http://www.infosec.sdu.edu.cn/paper/199.pdf
17) Wang, X., Lai, X., Feng, D., Chen, H. and Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD, *Eurocrypt 2005*, LNCS 3494, pp.1–18 (May 2005).
18) Wang, X. and Yu, H.: How to Break MD5 and Other Hash Functions, *Eurocrypt 2005*, LNCS 3494, pp.19–35 (May 2005).
19) Wang, X., Yin, Y.L. and Yu, H.: Collision Search Attacks on SHA1 (Feb. 2005). http://www.infosec.sdu.edu.cn/paper/sha-attack-note.pdf
20) Wang, X., Yin, Y.L. and Yu, H.: Efficient Collision Search Attacks on SHA-0, *Crypto 2005*, LNCS 3621, pp.1–16 (Aug. 2005).
21) Wang, X., Yin, Y.L. and Yu, H.: Finding Collisions in the Full SHA-1, *Crypto 2005*, LNCS

2621, pp.17–36 (Aug. 2005).
22) Wang, X., Yao, A. and Yao, F.: New Collision Search for SHA-1, *Crypto 2005 Rump Session* (Aug. 2005). http://www.iacr.org/conferences/crypto2005/rumpSchedule.html
23) Matusiewicz, K. and Pieprzyk, J.: Finding Good Differential Patterns for Attacks on SHA-1, Cryptology ePrint Archive: Report 2004/364 (Dec. 2004). http://eprint.iacr.org/2004/364
24) Rijmen, V. and Oswald, E.: Update on SHA-1, Cryptology ePrint Archive: Report 2005/010 (Dec. 2004). http://eprint.iacr.org/2005/010.pdf
25) Yin, Y.L., Personal communication (July 2005).
26) Satoh, A. and Inoue, T.: ASIC-Hardware- Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS, *Proc. ITCC 2005* (*International Conference on Information Technology*), Vol.1, pp.532–537 (Apr. 2005).
27) IBM Cu-11 Standard Cell / Gate Array ASIC. http://www-03.ibm.com/chips/products/asics/products/cu-11.html
28) Curtin, M. and Dolske, J.: A Brute Force Search of DES Keyspace, login:online The USENIX Association Magazine on the Web, Special Issue on Security (May 1998). http://www.usenix.org/publications/login/1998-5/curtin.html
29) Moore, G.E.: Cramming More Components Onto Integrated Circuits, *Electronics*, Vol.38, No.8 (Apr. 1965).

(平成 17 年 10 月 6 日受付)
(平成 18 年 5 月 9 日採録)

佐藤　証（正会員）

昭和 39 年生．平成 1 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所入所．平成 11 年早稲田大学より博士（工学）授与．情報セキュリティに関するアルゴリズムおよび，その高性能 VLSI 実装方式に関する研究に従事．電子情報通信学会会員．