Do We Need to Vary the Constants? Methodological Investigation of Block-Cipher Based Hash Functions

Donghoon Chang[1] and Moti Yung[2]

[1] Center for Information Security Technologies(CIST), Korea University, Korea
dhchang@cist.korea.ac.kr

[2] RSA Laboratories and Department of Computer Science, Columbia University, New York, USA
moti@cs.columbia.edu

Abstract. The recent collision attacks on the MD hash function family do not depend on the constants used in the function, but rather on its structure (i.e., changing the constants will not affect the differential analysis based attacks). Thus, is seems that the role of constants in maintaining security and preventing these attacks is unclear, at best, for this case and in particular fixing or varying the constants will not matter for these analyses.

In this work we present a methodological investigation into the case of block-cipher based PGV hash functions family, and investigate the importance of constants in securing these designs. To this end we consider the twelve variants of the PGV family that yield secure hash in the generic ideal cipher case (as was shown by Black, Rogaway and Shrimpton), but consider them under concrete instantiation. To investigate the role of constant in the key derivation procedure we just ignore the constants. In this more uniform setting we further consider a very regular cipher, namely AES modified to have Mixcolumn also in the last round (which should still be a strong cipher). Analyzing this modified-AES based hashing, we show that with about 16% probability we can find collisions with complexity 2^{49} (much smaller than the birthday attack complexity 2^{64}).

While we do not claim to break the AES based version, this nevertheless shows that constants in block cipher have an important role in resisting collision attack (in particular there is a need to vary the constant). It also shows that (in the symmetric modified version) merely the concrete AES structure does not guarantee the security of AES-based hash function (shown secure under the ideal cipher model). This is undesirable and non-robust, because this means that even though a block cipher has complicated structures in its round function and its key scheduling algorithm, we can not have a confidence about the security of hash functions based solely on it (note that there are several block ciphers such as IDEA, 3-key triple DES which do not use any constants). Given the above methodological findings, we suggest new AES-based hash function constructions (essentially modified PGV) which can be generalized to any block cipher. The functions inherit the security under the ideal cipher model on the one hand, while, on the other hand, concretely assure in their structure that the weakness exhibited herein is dealt with.

Keywords : Hash Function, Collision Attack, Block Cipher.

1 Introduction.

Nowadays, the need to understand and design secure hash functions is of high priority, which became clear due to the attacks by Wang and her coauthors [16–21]. We need to protect against these attacks, have robust design possibly resisting unknown attacks, and design secure simple hash functions to be used potentially in low power device environment. Till now there have been numerous concrete analyses on MD4-style hash functions [16–21]. On the other hand, in case of block cipher based hash function, several papers focus only on generic security analysis, i.e., when the underlying block cipher is the ideal block cipher [2, 9] or pseudorandom [8, 7, 4].

Recall that there is always a gap between generic analysis and the confidence we have about the security of concrete block cipher based hash functions. This paper, therefore, takes a first step into concrete security of practical block cipher based hash function. It is a step towards designing simple and secure hash functions whose security can be based on the underlying iterated block cipher structure. Note, for example, that recently RC4-hash, based on RC4 stream cipher, was suggested [3].

We show that the bare AES structure (slightly modified) does not guarantee the security of AES-based hash function (in a structure that is generically secure) and, in fact, that the security of AES-based hash functions depend on constants used in its key scheduling algorithm. This may be undesirable since a robust design should remain secure as constant change. This exhibits a difference between MD4-style hash functions and AES-style block cipher based hash function (as far as known attacks are concerned), because the (in)security of MD4-style hash functions does not depend on constants but merely on their structures. Thus, a very complicated cipher may not be secure due to lack of constants when employed in a hash function scheme, a fact that has to be considered in hash functions design.

In light of the above concrete unveiled issues, we contribute an initial step in block-cipher based hash function design as well and propose two kinds of simple modifications. These involve adding round keys and modified operations in the PGV constructions which prevent the attacker exhibited in this paper from finding collisions of modified AES-based hash functions. These modifications show the possibility of designing simple hash functions which has repetitions of a structure due to regular cipher structure, yet are not broken by our method which takes advantage of the cipher's repetitious structure.

The rest of the paper is organized as following. In Section 2 we give a description of AES and introduce constructions of block cipher based hash functions.

The security of a modified AES-based hash function is analyzed in Section 3. Then in Section 4 we suggest two modifications of block cipher based hash functions. We conclude in Section 5.

2 Preliminaries.

We first describe the AES algorithm [1] and block-cipher based hash function constructions.

2.1 AES Algorithm.

This paper focuses on AES-128 whose key size is 128-bit and message block size is 128-bit. S is S-box and M is MixColumn matrix operation. KSA is Key Scheduling Algorithm. Let $|x|$ be the bit-length of x. Here, $k_0 = k$ and $\mathsf{RotByte}(a_1||a_2||a_3||a_4)$=$(a_2||a_3||a_4||a_1)$ such that $|a_i|$=8. $\mathsf{SubByte}(a_1||a_2||a_3||a_4) = (\mathsf{S}(a_1)||\mathsf{S}(a_2)||\mathsf{S}(a_3)||\mathsf{S}(a_4))$. $\{Rcon_1, \cdots, Rcon_{10}\}$=$\{$0x01000000, 0x02000000,$\cdots$, 0x1b000000, 0x36000000$\}$. And M is defined as follows.

$$\mathsf{M}(a_0||a_2||a_3||a_4) = (a_0||a_2||a_3||a_4)\begin{pmatrix} 02\ 01\ 01\ 03 \\ 03\ 02\ 01\ 01 \\ 01\ 03\ 02\ 01 \\ 01\ 01\ 03\ 02 \end{pmatrix}$$

In matrix operation, each column of the matrix and $(a_0||a_2||a_3||a_4)$ are described in $0 \sim 3$ degree polynomial with coefficients in GF(2^8) which is also described in $0 \sim 7$ degree polynomial with coefficients in GF(2). For the addition and the multiplication, two irreducible polynomials, $x^8 + x^4 + x^3 + x + 1$ and $x^4 + 1$, are used for each modulo operation.

```
$\mathsf{AES}_{128}(k, m) = c$

$\mathsf{KSA}(k)$=$k_0||k_1||\cdots||k_{10}$, $|k_i|$=128.
$y = m \oplus k_0 = y_0||y_1||\cdots||y_{15}$, $|y_i|$=8.
for  $r$=1 to 9
  $t = \mathsf{SubByte}(y) = t_0||t_1||t_2||t_3||t_4||\cdots||t_{15}$ for $t_i = \mathsf{S}(y_i)$.
  $v = \mathsf{ShiftRow}(t) = t_0||t_5||t_{10}||t_{15}||t_4||t_9||t_{14}||t_3||t_8||t_{13}||t_2||t_7||t_{12}||t_1||t_6||t_{11}$.
  $w = \mathsf{MixColumn}(v) = \mathsf{M}(v_0||\cdots||v_3)||\mathsf{M}(v_4||\cdots||v_7)||\mathsf{M}(v_8||\cdots||v_{11})||\mathsf{M}(v_{12}||\cdots||v_{15})$.
  $y = w \oplus k_r$.
$t = \mathsf{ByteSub}(y)$.
$v = \mathsf{ShiftRow}(t)$.
$c = v \oplus k_{10}$.
```

Fig. 1. AES-128 Algorithm. k is the master key and m is any plaintext and c is the corresponding ciphertext to m.

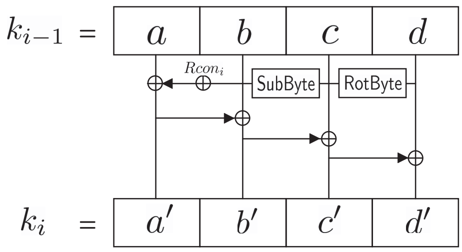

Fig. 2. Key Scheduling Algorithm of AES-128.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0b	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 1. S-box of AES. For example, S(53)=ed.

2.2 Constructions of Block-Cipher based Hash Functions.

There are numerous known constructions of hash functions based on underlying block ciphers. Among them, PGV constructions are perhaps the most known. These family was designed in 1993 by Preneel, Govaerts and Vandewalle [10] who considered 64 block cipher based hash functions. In 2002, Black, Rogaway and Shrimpton [2] proved that 20 of 64 PGV-hash functions are collision resistant in case of using fixed initial value and assuming that the underlying block cipher is a random block cipher (i.e., a generic analysis). Among the 20 constructions, 12 constructions (indexed by $f_1 \sim f_{12}$) are secure against the free-start collision attack but $f_{13} \sim f_{20}$ are not secure against the free-start collision attack [2]. Recall that given a compression function f, We say that (h_{i-1}, m_i) and (h'_{i-1}, m'_i) are a *free-start collision pair* when $f(h_{i-1}, m_i) = f(h'_{i-1}, m'_i)$ and $(h_{i-1}, m_i) \neq (h'_{i-1}, m'_i)$.

3 Collision Attacks on a Modified AES-based Hash Functions

In this section, we describe a slightly modified AES, AES*.

3.1 A Slightly Modified AES : AES*

We ignore constants in the key scheduling algorithm of AES and add Mixcolumn operation to the last round. We call this algorithm AES*. The basic goal is to have a very regular structure, and both factors contribute to it. We explain the reason why *a-priori* AES* seems to be secure against any collision attack.

Constants. Known collision attacks such as Dobbertin's attack and Wang's attack are applied against hash functions regardless of the values of constants in these functions design (i.e., the attack will work with any constant). Thus, constants seem not to help the hash functions to be secure against known collision attack. In fact, the attacks mentioned above depends on and exploit the structures of the hash functions rather than the value of constants in their design.

For example, the general structures of MD4/5 [12, 13], HAVAL [22], RIPEMD [11] and SHA-0/1 [14, 15] are very similar. We call them MD4-style hash functions. Since MD4 was analyzed by Dobbertin [5] and then Wang, all MD4-style hash functions were eventually analyzed. (RIPEMD consists of two parallel algorithms which are same except constants.) On the other hands, there is no attack on RIPEMD-128/160 [6] which consists of two parallel algorithms which have different message reordering, different boolean functions, different shift rotations and different constant at each step. Given the state of the methodology of attacks, we can easily say that the security against any of the avialable collision attacks does not depend inherently on constants but rather on weaknesses of the structure of the hash functions. Thus, and this is very natural, we can methodologically deduce as a design principle that hash function should be secure from their structural viewpoint (and adversarial modification of constants should not reduce their strength).

Adding Mixcolumn. The reason why there is no Mixcolumn operation in last round of AES is that the encryption and the decryption of AES would be similar in structure. This simplifies implementations in general, and allows the same basic components to be reused in hardware implementations. However, any block cipher based hash function uses only the encryption process, so we do not care about the decryption operation in this case. Also, we certainly hope that adding Mixcolumn does not really reduce the security of AES as a block cipher.

3.2 Collision Attacks on AES*-based Hash Functions

In this subsection, we describe collision attacks on AES*-based hash functions. These attacks show that the structure of AES (by itself) cannot guarantee the security of AES-based hash functions. It further demonstrates that the constants of the key scheduling algorithm of AES have an important role in making the hash functions secure.

Our attack consists of three parts: analysis of the key scheduling algorithm, analysis of the round transformation and analysis of the constructions of block cipher based hash function in [2]. Note that our attack method can not be used to attack properly modified MD4-style hash functions (i.e., such functions without the constants).

Analysis of The Key Scheduling Algorithm of AES*

Fig 3 shows the key scheduling algorithm of AES*. We want to find the value of k_{i-1} and k_i such that $k_{i-1} = k_i$. It is easy to find the value. With S-box of table 1, we can know that the value is (0x00000000, 0x00000000, 0x00000000, 0x52525252). We call this 128-bit value 'a'.

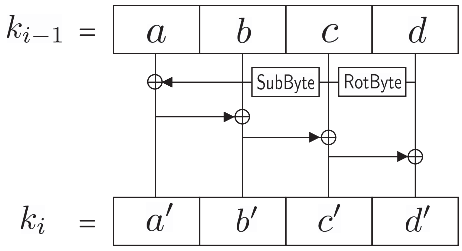

Fig. 3. Key Scheduling Algorithm of AES*-128.

Analysis of Round Transformation of AES*

Each round transformation consists of $\mathsf{SubByte}$, $\mathsf{ShiftRow}$, $\mathsf{MixColumn}$ and adding roundkey. We define the i-th round transformation by $\mathsf{Round}_i(k_i,\ y)$ where k_i is the i-th round key and y is the input. Then we want to find y's such that $\mathsf{Round}_i(a,\ y)$=y in complexity smaller than the birthday attack complexity. We denote y as $(y_0||y_1||y_2||\cdots||y_{15})$ where $|y_i|$=8 and denote a as $(a_0||a_1||a_2||\cdots||a_{15})$ where $|a_i|$=8. $\mathsf{Round}_i(a,\ y)$=y can be described in detail as following:

1. $(y_0 \oplus a_0||y_1 \oplus a_1||y_2 \oplus a_2||y_3 \oplus a_3)$=$\mathsf{M}(\mathsf{S}(y_0)||\mathsf{S}(y_5)||\mathsf{S}(y_{10})||\mathsf{S}(y_{15}))$
2. $(y_4 \oplus a_4||y_5 \oplus a_5||y_6 \oplus a_6||y_7 \oplus a_7)$=$\mathsf{M}(\mathsf{S}(y_4)||\mathsf{S}(y_9)||\mathsf{S}(y_{14})||\mathsf{S}(y_3))$
3. $(y_8 \oplus a_8||y_9 \oplus a_9||y_{10} \oplus a_{10}||y_{11} \oplus a_{11})$=$\mathsf{M}(\mathsf{S}(y_8)||\mathsf{S}(y_{13})||\mathsf{S}(y_2)||\mathsf{S}(y_7))$
4. $(y_{12} \oplus a_{12}||y_{13} \oplus a_{13}||y_{14} \oplus a_{14}||y_{15} \oplus a_{15})$=$\mathsf{M}(\mathsf{S}(y_{12})||\mathsf{S}(y_1)||\mathsf{S}(y_6)||\mathsf{S}(y_{11}))$

Now we try to compute the complexity of finding y's satisfying the above four equations. In case of the first equation, we expect 2^{24} solutions because y_0 is used in both of its sides. So, with expected complexity 2^{32}, we get 2^{24} solutions. Then, for each solution of the first equation, since y_3 and y_5 are fixed already in the first equation and y_4 is used in both sides of the second equation, the complexity of finding solutions of second equation is 2^{24} and we expect 2^8 such solutions. Thus, we get 2^{32} solutions satisfying the first and the second equations with complexity 2^{48}. Then, for each solution among the 2^{32} solutions satisfying the first, the second and the third equations, since y_2, y_7, y_9 and y_{10} are fixed already and y_8 is used in both sides, the complexity of finding solutions of the second equation is 2^{16} and we expect 2^{-8} solutions (i.e. one in 2^8). So we get 2^{24} solutions satisfying the first, second and third equations with complexity 2^{48}. Then, for each solution among the 2^{24} solutions, since y_2, y_7, y_9 and y_{10} are fixed already and y_8 is used in both sides, the complexity of finding solutions of the second equation is 2^8 and we expect 2^{-24} solutions. So on average we get one solution satisfying first, second, third and forth equations with complexity 2^{32}. Therefore, the total complexity is about $2^{49}(\approx 2^{32}+2^{48}+2^{48}+2^{32})$.

Next, we try to compute roughly the probability that there exist at least two y's. We expect one y on average, because y is 128-bit and we do the exhaustive search of 2^{128} candidates of y. If we assume that this follows the binomial distribution, then the average is 1 and the standard deviation is $\sqrt{1-2^{-128}}(=\sqrt{2^{128}\cdot 2^{-128}\cdot(1-2^{-128})})$. Thus, we can deduce that there exist at least two y's with about 16% probability according to the standard normal distribution table. We denote such two y's as y and y'.

Analysis of AES*-based hash functions in [2]

Through previous subsections, with about 16% probability we can get $(a,\ a \oplus y)$ and $(a,\ a \oplus y')$ such that $\mathrm{AES}^*_{128}(a,\ a \oplus y)$=$y$ and $\mathrm{AES}^*_{128}(a,\ a \oplus y')$=$y'$ in Fig 4 because for each i-th round $\mathsf{Round}_i(a,\ y)$=y and $\mathsf{Round}_i(a{,}y')$=y' and $\mathrm{AES}^*_{128}(a,\ b)$=$\mathsf{Round}_{10}(a, \cdots \mathsf{Round}_3(a, \mathsf{Round}_2(a, \mathsf{Round}_1(a, a \oplus b)))\cdots)$. Fig 5 shows that the above two pairs can be applied to twelve PGV constructions ($f_1 \sim f_{12}$) in order to get free-start collisions. Especially, in the case of $f_1 \sim f_4$, if the initial value is a, we can get collisions of AES*-based hash functions. In the case of $f_{13} \sim f_{20}$, Fig 6 shows that the above two pairs are useless even for getting free-start collisions. This shows that we can not deduce that $f_1 \sim f_{12}$ are no worse constructions than $f_{13} \sim f_{20}$ only by reasoning that $f_1 \sim f_{12}$ are free-start collision resistant generically, i.e., when the underlying block cipher is the ideal block cipher.

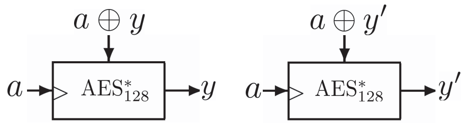

Fig. 4. Two Undesirable Properties of AES*-128.

4 Suggestion of New Block Cipher based Hash Functions

Here, we give preliminary design suggestion for kinds of modifications that prevent the attacker described above from finding a collision on block ciphers, relying the security on the functions' structures. These modifications may help designing simple and secure hash functions (especially for low power device environments such as RFID and sensor network) because we may be able to use a small fixed component repeatedly and also we can simplify the key scheduling part of block cipher for designing block cipher based hash functions. The idea is that the modification will retain the generic security analysis while coping with attacks based on the exploitation of the XOR function for accelerating collision finding (as demonstrated above).

Modification of Adding round key in Block Cipher

Block ciphers such as AES use only shift rotations and XOR operation except S-box. This may cause us to have $y \oplus y$ and $y' \oplus y'$ outputting the same value while $y \neq y'$. Thus, we suggest that the addition operation and XOR operation are used in an alternate fashion, rather than using only XOR operation.

Modification of the PGV constructions

The PGV constructions use only the XOR operation. We thus comment that from the point of view of the attack, it may be wise to change the XOR operation into the addition operation. After modification the functions' generic security can be proved in the same way as in [2].

We believe that modifications to structures as above, which contribute to having more robust designs in general, are interesting area of research.

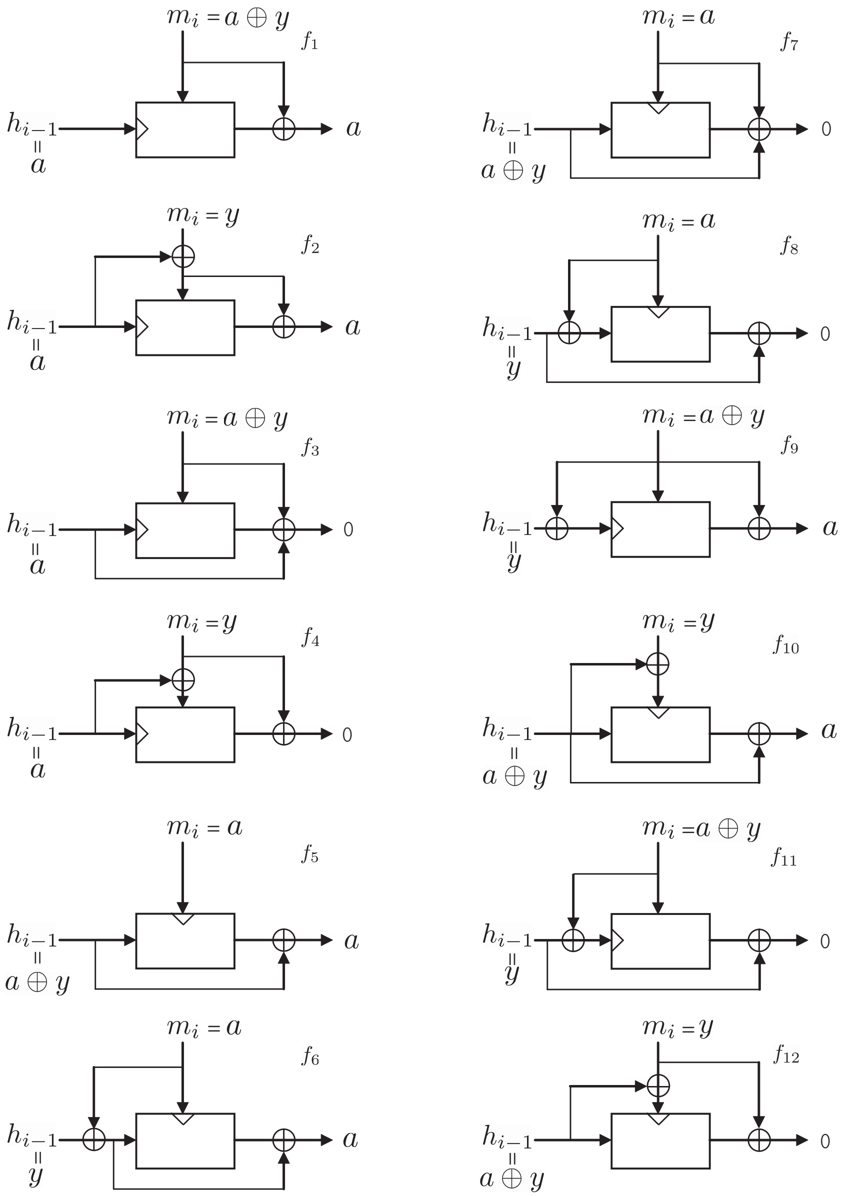

Fig. 5. Twelve PGV Constructions $f_1 \sim f_{12}$. The box is AES*-128. Here, we consider only y. We can apply y' in the same way.

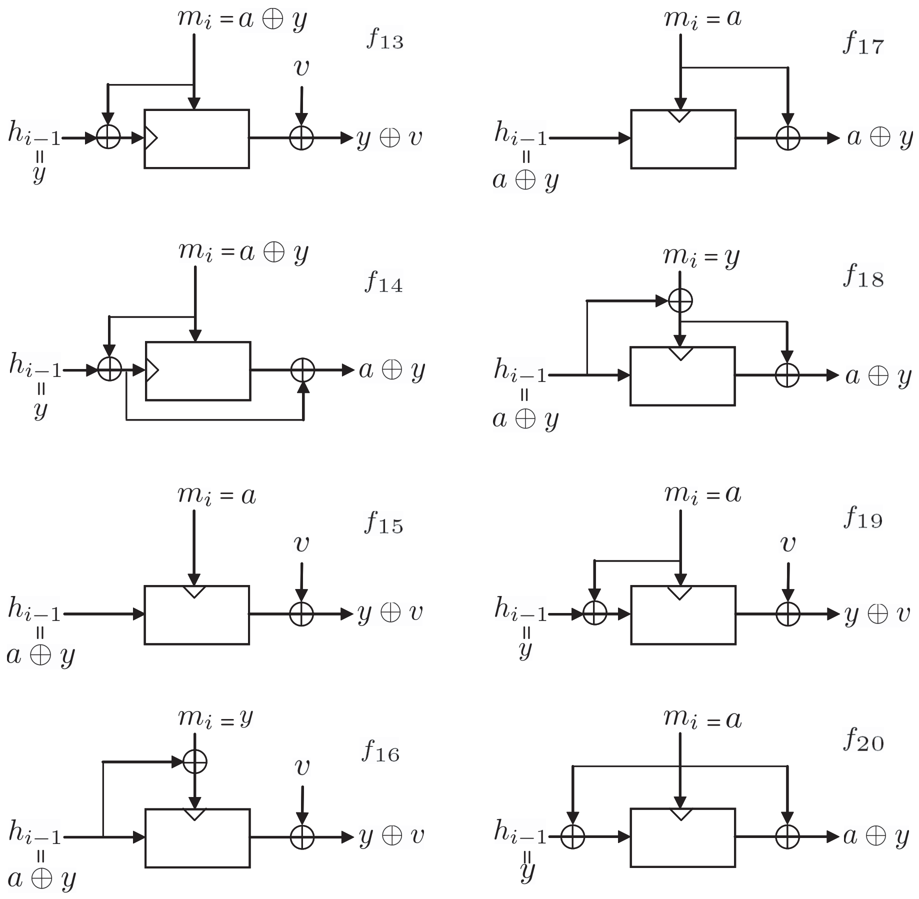

Fig. 6. Eight PGV Constructions $f_{13} \sim f_{20}$. The box is AES*-128. Here, we consider only y. We can apply y' in the same way.

5 Conclusion

Nowadays, we need to design secure hash functions (secure against attacks like Wang's, and robust in general). This means we need to understand some basic design principles on major methods for constructing robust hash functions. This paper contributes methodologically to such general understanding. We also need to design very light hash function if we want function suitable to modern environments. The paper's overall goal was to give some new insight into designing hash function based on block ciphers. The paper further attempts to show that care is needed and that, possibly, there may be concrete weaknesses in the combination of the cipher structure and the hashing iteration. The paper, in fact, suggests that we employ carefully block ciphers when we construct hash functions based on them, especially if we want robust protection based on the overall structure (rather than protection based on specific constants employed). Unfortunately, the AES structure when slightly modified does not guarantee the security of AES-based hash function which actually depends on the choice of constants in the key scheduling algorithm. We believe that the work as a whole may lead to better understanding of design principles and that the two kinds of modifications of hash function suggested in this paper help designing simple and secure hash functions.

References

1. Federal Information Processing Standards Publication 197, *ADVANCED ENCRYPTION STANDARD (AES)*, 2001.
2. J. Black, P. Rogaway and T. Shrimpton, *Black-box analysis of the block-cipher-based hash function constructions from PGV*, Advances in Cryptology - CRYPTO'02, LNCS 2442, Springer-Verlag, pp. 320-335, 2002.
3. D. Chang, K. C. Gupta and M. Nandi, *RC4-Hash : A New Hash Function based on RC4 (Extended Abstract)*, Indocrypt'06, to appear.
4. D. Chang, W. Lee, S. Hong, J. Sung, S. Lee and S. Sung, *Impossibility of Construction of OWHF and UOWHF from PGV Model Based on Block Cipher Secure Against ACPCA*, Indocrypt'04, LNCS 3348, Springer-Verlag, pp. 328-342, 2004.
5. H. Dobbertin, *Cryptanalysis of MD4*, FSE'96, LNCS 1039, Springer-Verlag, pp. 53-69, 1996.
6. H. Dobbertin, A. Bosselaers and B. Preneel, *RIPEMD-160: A Strengthened Version of RIPEMD*, FSE'96, LNCS 1039, Springer-Verlag, pp. 71-82, 1996.
7. S. Hirose, D. Chang and W. Lee, *A Note on Defining Pseudorandom Function Ensembles*, ISEC 2004, The Institute of Electronics, Information and Communication Engineers. Vol.104, No.53, pp. 1-6, May 2004.
8. S. Hirose, *Secure Block Ciphers Are Not Sufficient for One-Way Hash Functions in the Preneel-Govaerts-Vandewalle Model*, SAC'02, LNCS 2595, Springer-Verlag, pp. 339-352, 2003.
9. W. Lee, M. Nandi, P. Sarkar, D. Chang, S. Lee and K. Sakurai, *A Generalization of PGV-Hash Functions and Security Analysis in Black-Box Model*, ACISP'04, LNCS 3108, Springer-Verlag, pp. 212-223, 2004.

10. B. Preneel, R. Govaerts and J. Vandewalle, *Hash functions based on block ciphers:A synthetic approach*, Advances in Cryptology - CRYPTO'93, LNCS 773, Springer-Verlag, pp. 368-378, 1994.

11. RIPE, Integrity Primitives for secure Information systems, Final report of RACE Integrity Primitive Evaluation (RIPE-RACE 1040) *Lecture Notes in Computer Science*, Springer-Verlag, 1995.

12. Ronald L. Rivest. The MD4 message-digest algorithm. In *Crypto'1990*, volume **537** of *Lecture Notes in Computer Science*, pages 303–311, Springer-Verlag, 1991.

13. Ronald L. Rivest. The MD5 message-digest algorithm. Request for comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.

14. NIST. Secure hash standard. Federal Information Processing Standard, FIPS-180, May 1993.

15. NIST. Secure hash standard. Federal Information Processing Standard, FIPS-180-1, April 1995.

16. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 1–18, Springer-Verlag, 2005.

17. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology-Eurocrypt'2005*, volume **3494** of *Lecture Notes in Computer Science*, pages 19–35, Springer-Verlag, 2005.

18. X. Wang, H. Yu and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 1–16, Springer-Verlag, 2005.

19. X. Wang, Y. L. Yin and H. Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology-Crypto'2005*, volume **3621** of *Lecture Notes in Computer Science*, pages 17–36, Springer-Verlag, 2005.

20. H. Yu, X. Wang, A. Yun and S. Park. Cryptanalysis of the Full HAVAL with 4 and 5 Passes. To appear in *FSE'2006*, Springer-Verlag, 2006.

21. H. Yu, G. Wang, G. Zhang and X. Wang. The Second-Preimage Attack on MD4. In *CANS'2005*, volume **3810** of *Lecture Notes in Computer Science*, pages 1–12, Springer-Verlag, 2005.

22. Y. Zheng, J. Pieprzyk and J. Seberry HAVAL - A One-Way Hashing Algorithm with Variable Length of Output In *ASIACRYPT 1992*, *Lecture Notes in Computer Science*, pages 83–104, Springer-Verlag, 1992.