Weaknesses in the HAS-V Compression Function

Florian Mendel* and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria

{Florian.Mendel,Vincent.Rijmen}@iaik.tugraz.at

Abstract. HAS-V is a cryptographic hash function which processes 1024-bit message blocks and produces a hash value of variable length. The design of the hash function is based on design principles of the MD4 family. Recently, weaknesses have been shown in members of this family. Therefore, the analysis of the HAS-V hash function is of great interest. To the best of our knowledge this is the first article that investigates the security of the HAS-V hash function. In this article, we point out several structural weaknesses in HAS-V which lead to pseudo-collision attacks on HAS-V with tailored output. Furthermore, we show that (second) preimages can be found for HAS-V with a complexity of about 2^{162} hash computations.

1 Introduction

Recently, weaknesses in many commonly used hash functions, such as MD5 and SHA-1 have been found [1,2,9,10]. These breakthrough results in the cryptanalysis of hash functions are the motivation for intensive research in the design and analysis of hash functions. In this article, we will study the HAS-V hash function in detail. It is an iterated hash function that processes 1024-bit message blocks and produces a hash value of variable length: 128, 160, 192, 224, 256, 288, and 320 bits. The HAS-V hash function was proposed by Park *et al.* at SAC 2000 [7]. The design of the hash function is very similar to the design principles of HAS-160, HAVAL, and RIPEMD. Since in all of these hash functions (recently) weaknesses have been shown [3,5,8,11], a detailed analysis of the HAS-V hash function is needed to get a good view on the security margins of the hash function. We are not aware of any other security analysis of the HAS-V hash function.

In this article, we show that the HAS-V hash function has several weaknesses that can be exploited to construct pseudo-collisions for the HAS-V hash function for all output sizes. Furthermore, we show that by using an alternative description of the hash function, which gives more insights in the design of HAS-V, we can construct (second) preimages with a complexity of about 2^{162} hash computations. Note that for an ideal hash function with an n-bit output size, one would expect a complexity of about 2^n. Hence, the security margins for HAS-V

* This author is supported by the Austrian Science Fund (FWF), project P18138.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 335–345, 2007.
http://dx.doi.org/10.1007/978-3-540-76788-6_27

with output size 192, 224, 256, 288, and 320 bits are not as high as one would expect.

The remainder of this article is structured as follows. A description of the hash function is given in Section 2. In Section 3, we give an alternative description of HAS-V, which gives more insights in the design of the hash function. We will use this description in Section 4 to show how pseudo-collisions can be constructed. In Section 5, we present a (second) preimage attack with a complexity of about 2^{162} hash computations. Finally, we present conclusions in Section 6.

2 Description of the HAS-V Hash Function

HAS-V is an iterative hash function that processes 1024-bit input message blocks and produces a hash value of variable length: $128, 160, \cdots, 288$, and 320 bits. The design of HAS-V is similar to the design principle of HAVAL, RIPEMD and HAS-160. It consists of two parallel streams. In each stream the state variables are updated (in 5 rounds) according to the expanded message words. After each round the state variables of the left and the right stream are interchanged, see Fig. 1. After the last round, the initial values and the output values of each stream are combined, resulting in the final value of one iteration.

After the last message block has been processed an output tailoring method is applied to construct a hash value of length 128, 160, 192, 224, 256, 288 or 320 bits. For a detailed description of this method we refer to [7].

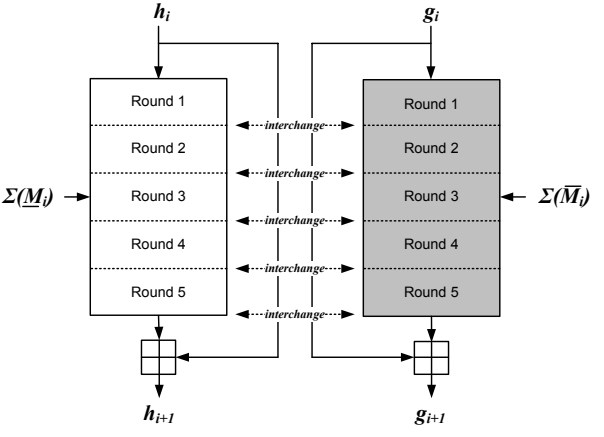

Fig. 1. The HAS-V compression function.

In the following, we briefly describe the hash function. It basically consists of two parts: the message expansion and the state update transformation.

2.1 Message Expansion.

The message expansion of HAS-V is a permutation of 20 expanded message words w_i in each round of a stream. First, the message block M_i of 1024 bits is split into 2 parts $\underline{M_i}$ and $\overline{M_i}$, where $\underline{M_i}$ are the lower 512 bits and $\overline{M_i}$ are the upper 512 bits of the message block M_i. $\underline{M_i}$ is then used to generate the expanded message words for the left stream and $\overline{M_i}$ is used to generate the expanded message words for the right stream. The 20 expanded message words w_i used in each round are constructed from the 16 input message words m_i as follows. The 16 message words m_i are copied to w_i. The remaining 4 expanded message words w_{16}, w_{17}, w_{18} and w_{19} are generated as shown below.

	w_{16}	w_{17}	w_{18}	w_{19}
Round 1	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$
Round 2	$w_3 \oplus w_6 \oplus w_9 \oplus w_{12}$	$w_{15} \oplus w_2 \oplus w_5 \oplus w_8$	$w_{11} \oplus w_{14} \oplus w_1 \oplus w_4$	$w_7 \oplus w_{10} \oplus w_{13} \oplus w_0$
Round 3	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$
Round 4	$w_7 \oplus w_2 \oplus w_{13} \oplus w_8$	$w_3 \oplus w_{14} \oplus w_9 \oplus w_4$	$w_{15} \oplus w_{10} \oplus w_5 \oplus w_0$	$w_{11} \oplus w_6 \oplus w_1 \oplus w_{12}$
Round 5	$w_{15} \oplus w_9 \oplus w_5 \oplus w_3$	$w_{12} \oplus w_8 \oplus w_6 \oplus w_2$	$w_{13} \oplus w_{11} \oplus w_7 \oplus w_1$	$w_{14} \oplus w_{10} \oplus w_4 \oplus w_0$

For the ordering of the expanded message words w_i in each round the following permutation is used:

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Round 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Round 2	18	3	6	9	12	19	15	2	5	8	16	11	14	1	4	17	7	10	13	0
Round 3	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Round 4	18	7	2	13	8	19	3	14	9	4	16	15	10	5	0	17	11	6	1	12
Round 5	18	15	9	5	3	19	12	8	6	2	16	13	11	7	1	17	14	10	4	0

2.2 State Update Transformation.

The state update transformation of HAS-V starts from a (fixed) initial value IV of ten 32-bit registers (5 for each stream) and updates them in 5 rounds of 20 steps each. Figure 2 shows one step of the state update transformation of the HAS-V hash function.

$$
\begin{aligned}
A_{i+1} &= A_i \lll s + f(B_i, C_i, D_i, E_i) + W_i + K_i \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \ggg 2 \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i
\end{aligned}
$$

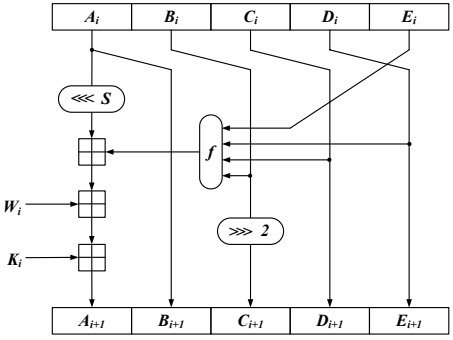

Fig. 2. The step function of HAS-V.

The function f is different in each round. f_j is used for the j-th round in the left stream, f_{4-j} is used for the j-th round in the right stream ($j = 0, \ldots, 4$).

$$
\begin{aligned}
f_0(B, C, D, E) &= (B \wedge C) \oplus (\neg B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \\
f_1(B, C, D, E) &= (B \wedge D) \oplus C \oplus E \\
f_2(B, C, D, E) &= (B \wedge C) \oplus (\neg B \wedge E) \oplus D \\
f_3(B, C, D, E) &= B \oplus (C \wedge D) \oplus E \\
f_4(B, C, D, E) &= (\neg B \wedge C) \oplus (B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E)
\end{aligned}
$$

A step constant K_i is added in every step; the constant is different for each round. For the actual values of the constants we refer to [7]. The rotation value s is different in each step of a round.

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s	5	11	7	13	15	6	13	9	5	11	7	12	8	15	13	8	15	6	7	14

After the last step of the state update transformation, the initial value and the output values of the last step are combined, resulting in the final value of one iteration. In detail, the feed forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

3 Alternative Description of HAS-V

We present here a slightly different description of the HAS-V hash function, which gives more insights in the design of the hash function. By interchanging round 2 and round 4 between the two streams instead of interchanging the state variables after each round, we get an alternative description of the HAS-V hash

function, where the two streams are independent, see Fig. 3. Note that a new (different) message expansion is used for the left and the right stream, denoted by Σ^1 and Σ^2.

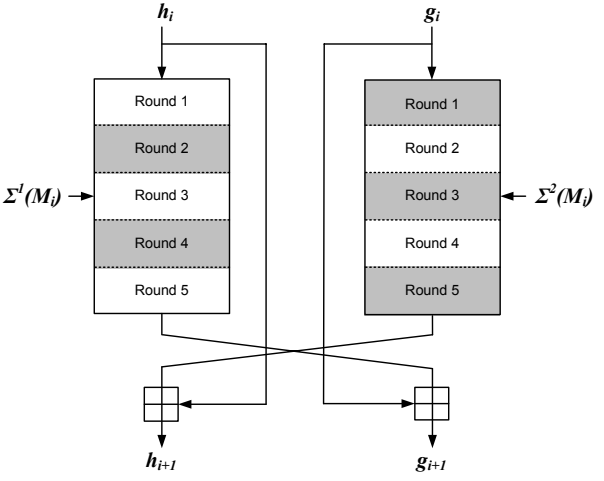

Fig. 3. Alternative description of the HAS-V compression function.

Let $h(M_i, h_i)$ and $g(M_i, g_i)$ denote the state update function in the left and right stream. Then the compression function of HAS-V can be described as follows:

$$h_{i+1} = h_i + g(M_i, g_i)$$
$$g_{i+1} = g_i + h(M_i, h_i)$$

For the remainder of this article, we will use this description of the HAS-V hash function. First, we show that a pseudo-collision can be constructed for HAS-V with a complexity lower than $2^{n/2}$ (which one would expect for an ideal hash function with an n-bit hash value). Second, we show that (second) preimages can be found for HAS-V with a complexity of about 2^{162} hash computations.

4 Pseudo-Near-Collisions and Pseudo-Collisions for the HAS-V Hash Function

In this section, we will show how pseudo-near-collisions and pseudo-collisions can be constructed for the HAS-V compression function by exploiting structural weaknesses in the step function of the hash function. The attack is based on the following two observations.

Observation 1 *The properties of the Boolean function f (used in the first round of both streams) can be used to absorb differences at its input.*

Observation 2 *In the step function of HAS-V state variable E is only used as input to the non-linear Boolean function f (see Fig. 2).*

Hence, a difference δ in state variable E can easily be canceled by exploiting the properties of the Boolean function f to absorb differences at its input. It is easy to see, that a difference δ in state variable E in the first round of the left stream will always cancel out if $C = D$.

$$f_0(B, C, D, E) = C \quad \Leftrightarrow \quad C = D$$

In a similar way also a difference δ' in E in the first round of the right stream will cancel out if $C = D$.

$$f_4(B, C, D, E) = C \quad \Leftrightarrow \quad C = D$$

We can use this to construct a pseudo-near-collision in the HAS-V hash function. By choosing the initial value of the left and right stream in the following way

$$h_0 = A_0^1||B_0^1||C_0^1||D_0^1||E_0^1 \quad \text{with} \quad C_0^1 = D_0^1$$

and

$$g_0 = A_0^2||B_0^2||C_0^2||D_0^2||E_0^2 \quad \text{with} \quad C_0^2 = D_0^2$$

we will always get a collision after the first step of the state update transformation in both streams for an arbitrary difference δ in E_0^1 in the left stream and δ' in E_0^2 in the right stream. Of course, the feed forward after the last step of the state update transformation will destroy the collision, resulting in a pseudo-near-collision for the HAS-V hash function:

$$(0, 0, 0, 0, \delta, 0, 0, 0, 0, \delta') \rightarrow (0, 0, 0, 0, \delta, 0, 0, 0, 0, \delta')$$

Note that this holds with probability 1 and is independent of the message M. The pseudo-near-collision for HAS-V can be turned into a pseudo-collision for HAS-V with tailored output. This is described in more detail in the next section.

4.1 Pseudo-Collisions in HAS-V with tailored output

The designers of HAS-V defined several variants of HAS-V with tailored output. For instance, to get a hash value of 160 bits the output of the left an right stream are combined in the following way:

$$h + g = A^1 + A^2||B^1 + B^2||C^1 + C^2||D^1 + D^2||E^1 + E^2$$

For this variant, we can construct a pseudo-collision by using the pseudo-near-collision described in the previous section. By choosing a difference δ in E_0^1 in the left stream and a difference $-\delta$ in E_0^2 in the right stream, we get a pseudo-collision after adding the values of both streams (output tailoring).

$$E^1 + E^2 = (E^1 + \delta) + (E^2 - \delta)$$

This holds with probability 1 in the HAS-V hash function and is independent of the message M. Note that pseudo-collisions can be constructed for the other variants of HAS-V (with tailored output) in a similar way.

4.2 Pseudo-Collisions in HAS-V

We can turn the pseudo-near-collision for HAS-V into a pseudo-collision by using a generic birthday attack. The main idea is to use the degree of freedom we have in the choice of the differences in E_0^1 and E_0^2 in the left and right stream to decrease the work needed for a birthday attack. The attack can be summarized as follows:

1. Choose random values for the initial value of the left and right stream of the hash function with $C_0^1 = D_0^1$ and $C_0^2 = D_0^2$.
2. Do a generic birthday attack to find a message pair (M, M^*) such that $\Delta A^1 = \cdots = \Delta D^1 = 0$ and $\Delta A^2 = \cdots = \Delta D^2 = 0$. This has a complexity of about 2^{128} evaluations of the compression function of HAS-V.
3. To cancel the difference in E^1 and E^2 we use the fact that we can inject an arbitrary difference δ in E_0^1 and δ' in E_0^2 to cancel the differences in E^1 and E^2. This leads to a pseudo-collision in HAS-V hash function.

Hence, we can construct a pseudo-collision in the HAS-V hash function with a complexity of about 2^{128} instead of 2^{160} hash computations.

5 (Second) Preimages for the HAS-V Hash Function

In this section, we will describe two methods to construct a (second) preimage for the HAS-V hash function. Both attacks are based on structural weaknesses in the HAS-V hash function. For the analysis, we will use the description of HAS-V given in Section 3. With the first method a (second) preimage can be constructed with a complexity of about 2^{241} hash computations instead of 2^{320}, as one would expect for a hash function with a 320-bit hash value. The first attack is based on the following observation.

Observation 3 *The compression function of HAS-V is invertible with respect to the chaining variables.*

Algorithm 1. A way to invert the compression function.

Input: The final hash value $h_{i+1}||g_{i+1}$ and an arbitrary intermediate hash value h_i.
Output: The intermediate hash value g_i such that HAS-V$(M_i, h_i||g_i) = h_{i+1}||g_{i+1}$.

- Guess M_i and calculate:
$$g_i = g_{i+1} - h(M_i, h_i)$$
- Check if the following equation holds:
$$h_{i+1} = h_i + g(M_i, g_i).$$

Hence, a correct choice for M_i can be found in 2^{160} hash computations.

Based on Algorithm 1, the HAS-V hash function is invertible with respect to the chaining variables. We can find the intermediate hash value g_i and the message block M_i in about 2^{160} applications of the compression function of the HAS-V hash function. We can use this to construct a (second) preimage in the HAS-V hash function with a complexity of about 2^{241} hash computations using the following observation.

Observation 4 *An unbalanced meet-in-the-middle attack can be used to find a (second) preimage for the HAS-V hash function.*

Algorithm 2. A (second) preimage in the HAS-V hash function

Input: The final hash value $h_{i+2}||g_{i+2}$ and the intermediate hash value $h_i||g_i$
Output: The message $M = M_i||M_{i+1}$ such that HAS-V$(M, h_i||g_i) = h_{i+2}||g_{i+2}$.

- Calculate and store 2^{80} candidates for $h_{i+1}||g_{i+1}$ in the list L resulting from a backward computation of the compression function using Algorithm 1. ($2^{80} \cdot 2^{160}$ applications of the compression function)
- Calculate $h_{i+1}||g_{i+1}$ resulting from a forward computation of the compression function an check for a match in L.
- After calculating at most 2^{240} candidates for $h_{i+1}||g_{i+1}$ one expect find a matching entry (collision) in L and hence a (second) preimage for HAS-V. Note that a collision is likely to exist due to the birthday paradox.

Based on Algorithm 2, we can find a (second) preimage for the HAS-V hash function with complexity of about $2 \cdot 2^{240}$ applications of the compression function. Note that a similar attack was applied in the past on MDC-2 [4]. However, we can further improve the attack by using the following observation.

Observation 5 *Due to the structure of the HAS-V hash function it is easy to construct a fixed-point in the left stream of the HAS-V hash function for an arbitrary input value h_i.*

Algorithm 3. Constructing a fixed-point in the left stream of HAS-V.

Input: The intermediate hash value h_i.
Output: The pair (g_i, g_{i+1}) and M_i such that HAS-V$(M_i, h_i||g_i) = h_i||g_{i+1}$.

- Choose an arbitrary value for M_i and calculate:

$$g_i = g^{-1}(M_i, 0)$$

$$g_{i+1} = g_i + h(M_i, h_i)$$

Based on Algorithm 3, we can construct a fixed-point in the left stream of HAS-V for an arbitrary value of h_i. We can use this to construct a (second) preimage for the HAS-V hash function with a complexity of about 2^{162} hash computations. The main idea of the attack is to use Algorithm 3 to construct many fixed-points for the left stream of the hash function (with the same value of h_i) and save them in a list L and then combine these entries in such a way that we get a (second) preimage for the HAS-V hash function.

Assume we are given the final hash value $h_i||g_i$ and let $h_0||g_0$ denote the initial value of HAS-V. Then the attack can be summarized as follows:

1. Use Algorithm 1 with input $h_i||g_i$ and $h_{i-1} = h_0$ to get g_{i-1} and M_{i-1} such that HAS-V$(M_{i-1}, h_{i-1}||g_{i-1}) = h_i||g_i$. This step of the attack has a complexity of about 2^{160} hash computations and ensures that $h_{i-1} = h_0$. Note that this is needed for the attack to work.
2. Calculate and store $2 \cdot 2^{160}$ candidates for (g_{j-1}, g_j) in the list L using Algorithm 3 with input h_0. Note that each entry in the list L has a fixed-point in the left stream: HAS-V$(M_{j-1}, h_0||g_{j-1}) = h_0||g_j$. We will use this in the next step of the attack to construct a (second) preimage in the HAS-V hash function. Constructing the list L has a complexity of about 2^{161} hash computations. Furthermore, we expect to have always two entries in L where the first component g_{j-1} is equal and we also expect to have always two entries in L where the second component g_j is equal.
3. To construct a (second) preimage in the HAS-V hash function, we use the entries in the list L. Starting from g_0, we construct a tree using the entries in the list L. For each node in the tree we expect to get two new nodes on the next level (see Fig. 4), since we have two entries in list L where the first component is the same. Hence, the number of nodes at level k is 2^k.

   To construct a preimage for HAS-V we need to meet g_{i-1} fixed in the first step of the attack. Therefore, we construct a second tree starting from g_{i-1} also using the entries in the list L. Since we have always two entries in the list L, where the second component is equal, we get for each node on level k two new nodes on the next level. Hence, the number of nodes at level k is also 2^k.

   It is easy to see, that for $k = 80$ we get 2^{80} nodes in each of the two trees. Therefore, we expect to find a common node in both trees and hence, a (second) preimage for the HAS-V hash function. Note that a common node in both trees is likely to exist due to the birthday paradox.

With this method we can find a (second) preimage for HAS-V with a complexity of about 2^{162} hash computations and a message consisting of 161 (80+80+1) message blocks. Note that the same attack can be used to produce (second) preimages for HAS-V with tailored output. Hence, the security margins for the HAS-V hash function with output size of 192, 224, 256, 288, and 320 bits are not as high as one would expect for a hash function with that output size.

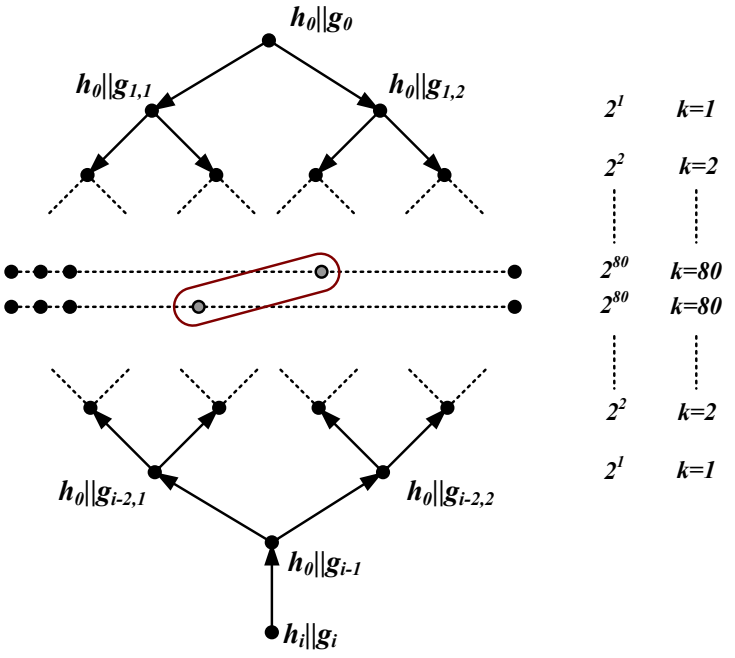

Fig. 4. Constructing a (second) preimage for the HAS-V hash function.

6 Conclusion

In this article, we showed several weaknesses in the HAS-V hash function. Based on these weaknesses, we presented a pseudo-collision for the HAS-V hash function with tailored output. As an example we show a pseudo-collision for HAS-V with a (tailored) output size of 160 bits. It has probability 1 to hold in the HAS-V hash function and is independent of the message M. Note that similar attacks (also with probability 1 and independent of the message) can be applied for the other variants of HAS-V with tailored output. Furthermore, we showed that also for the 320-bit variant of HAS-V a pseudo-collision can be found with a complexity of about 2^{128} hash computations.

Moreover, we presented a (second) preimage attack on the HAS-V hash function. In the attack we exploit the fact, that due to the design of the HAS-V hash function it is easy to construct a fixed-point in the left stream of the hash function for an arbitrary input value. The attack has a complexity of about 2^{162} hash computations. Hence, the security margins of HAS-V (with output size of 192, 224, 256, 288 and 320 bits) are not as high as one would expect for a hash function with an n-bit output size.

Acknowledgement

The authors wish to thank Norbert Pramstaller, and the anonymous referees for useful comments and discussions.

References

1. John Black, Martin Cochran, and Trevor Highland. A Study of the MD5 Attacks: Insights and Improvements. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2006.
2. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.
3. Hong-Su Cho, Sangwoo Park, Soo Hak Sung, and Aaram Yun. Collision Search Attack for 53-Step HAS-160. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC*, volume 4296 of *Lecture Notes in Computer Science*, pages 286–295. Springer, 2006.
4. Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.
5. Florian Mendel. Colliding Message Pair for 53-Step HAS-160. Cryptology ePrint Archive, Report 2006/334, 2006. `http://eprint.iacr.org/`.
6. Ralph C. Merkle and Martin E. Hellman. On the Security of Multiple Encryption. *Commun. ACM*, 24(7):465–467, 1981.
7. Nan Kyoung Park, Joon Ho Hwang, and Pil Joong Lee. HAS-V: A New Hash Function with Variable Output Length. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2000.
8. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.
9. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
10. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.
11. Hongbo Yu, Xiaoyun Wang, Aaram Yun, and Sangwoo Park. Cryptanalysis of the Full HAVAL with 4 and 5 Passes. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 2006.