Cryptanalysis of Block Ciphers Based on SHA-1 and MD5[*]

Markku-Juhani O. Saarinen

Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
mjos@tcs.hut.fi

Abstract. We cryptanalyse some block cipher proposals that are based on dedicated hash functions SHA-1 and MD5. We discuss a related-key attack against SHACAL-1 and present a method for finding "slid pairs" for it. We also present simple attacks against MDC-MD5 and the Kaliski-Robshaw block cipher.

Keywords: SHA-1, SHACAL, MD5, MDC, Slide attacks, Dedicated hash functions.

1 Introduction

One of the most widely used ways of creating cryptographic hash functions is the so-called Davies-Meyer mode. Let $Y = E(M, X)$ be a compression function that takes in a message block M with an input variable X and produces a partial digest Y of equal size to X. To compute a message digest of message $M_1 \mid M_2 \mid \cdots \mid M_n$, we simply set X_0 to some predefined initialization vector and iterate for $i = 1, 2, \ldots, n$:

$$X_i = E(X_{i-1}, M_i) \oplus X_{i-1}$$

The resulting message digest is X_n. See [7, 14, 16, 17] for further discussion of hash function modes of operation. Notably, in 2002 Black, Rogaway, and Shrimpton proved that this mode is secure if E is secure [3].

It has been observed that the compression functions of most widely-used dedicated hash functions MD5 [18] and SHA [19, 20] are in fact based on this construction. In these hash functions the exclusive-or operation has been replaced by wordwise addition modulo 2^{32} of the chaining variables. It has also been observed that the compression function E of MD5 and SHA can be efficiently computed in both directions; given $Y = E(X, M)$ and M, the original chaining value X can be recovered using an inverse transform $X = E^{-1}(Y, M)$. However, it is believed to be more difficult to recover X_{i-1}, given X_i and M.

[*] This work has been supported by the Finnish Defence Forces.

T. Johansson (Ed.): FSE 2003, LNCS 2887, pp. 36–44, 2003.

2 The SHACAL Block Ciphers

One of the proposals for the NESSIE project was the block cipher SHACAL, which is essentially the SHA-1 compression function with the Davies-Meyer chaining peeled off [11]. In this proposal the message block M is viewed as the "key", the chaining variable X acts as the plaintext block and $Y = E(X, M)$ is the corresponding ciphertext block.

Later, a "tweak" was submitted where the original SHACAL was renamed SHACAL-1 and a new block cipher SHACAL-2 (based on SHA-256) was also proposed [12]. We refer the reader to [11, 12, 20] for detailed specifications of the SHACAL block ciphers. The basic structure is also used as a part of the HORNET stream cipher [15].

A detailed analysis of differential and linear properties of SHA-1 in encryption mode can be found in [10], where it is conjectured that a linear cryptanalytic attack would require at least 2^{80} known plaintexts and a differential attack would require at least 2^{116} chosen plaintexts.

2.1 Sliding SHA-1 and SHACAL

Slide attacks against block ciphers were introduced by Biryukov and Wagner in 1999 [4, 5], although similar techniques had previously been used by others.

To our knowledge, slide attacks against hash functions have not been previously considered in the literature. Indeed it is difficult to see if and how "slid pairs" in the compression function can be exploited to find collisions for the hash function. This remains an open question.

However, it is interesting to consider the question whether or not slid pairs (which are essentially linear relations between two inputs and outputs) can be easily found for SHA-1. This is also related to Anderson's classification of hash functions [1]. David Wagner has considered a slide attack on 40 iterations of SHA-1 in unpublished work [21].

SHA-1 exhibits some properties which are useful when mounting slide attacks.

a) The SHA-1 compression function consists of four different "rounds". For 20 iterations of each round the nonlinear function F_i and the constant K_i are unchanged. There are only three transitions between different iteration types (see Figure 1).
b) The key schedule (i.e. message expansion) can be slid. We simply choose $W'_i = W_{i+1}$ for $0 \leq i \leq 14$ and $W'_{15} = (W_1 \oplus W_7 \oplus W_{12} \oplus W_{15}) \lll 1$. It is easy to see that after the key expansion $W'_i = W_{i+1}$ for $0 \leq x \leq 78$.

We note that these properties are not exhibited by SHA-256 (or SHA-512), thus making SHACAL-2 more resistant to slide attacks.

2.2 Related-Key Attacks

We shall consider the difficulty of distinguishing related keys. We assume that we are given chosen-plaintext access to two SHACAL-1 encryption oracles ("black boxes") whose keys are related in the way described in the previous section. The main question becomes how many chosen plaintexts are needed.

For the transition iterations between different types of functions we wish to find inputs that produce the same output word for both types ("round collisions"). Experiments have confirmed that the round functions behave sufficiently randomly for us to use 2^{-32} as the probability of a round collision. Since there are three transitions, a simple distinguisher will require approximately 2^{128} chosen plaintext pairs.

As pointed out by an anonymous program committee member, this can be improved to 2^{96} by using "structures"; first perform 2^{32} encryptions of (A, B, C, D, x) on the first oracle, where $x = 0, 1, 2, \cdots, 2^{32} - 1$ and A, B, C, D are some constants. Then do another 2^{32} encryptions of $(y, A, B \ggg 2, C, D)$ on the second "slid" oracle for $y = 0, 1, 2, \cdots, 2^{32} - 1$. Since each entry in the first set corresponds to some slid entry in the second set, the first collision is effectively obtained for free, and only 2^{96} pairs are required to distinguish the related keys.

A version of SHACAL-1 reduced to three rounds (60 iterations) will require 2^{64} pairs (only two transitions).

2.3 An Algorithm for Finding Slid Pairs

A method exists for finding slid pairs with roughly 2^{32} effort. The method is rather technical, so we can only give an overview of the key ideas used.

The general strategy is as follows. The algorithm doesn't start by choosing the plaintext or the ciphertext, but from the "middle", iterations 20 and 40. We find collisions in these positions with $O(1)$ effort and then work towards iterations $25 - 28$, where we perform a partial meet-in-the middle match.

Round Collisions. We note that in iteration i, not all input words affect the possibility of a round collision; only B, C, and D are relevant, since A and E only affect the output word linearily. Furthermore, the key word W_i has no effect to the probability of collision in iterations i or $i + 1$.

For iteration pair 19/20 (select-parity transition) we use [1]:

$$(B, C, D) = (\boxminus K_{20}, \boxminus K_0, \boxminus K_0)$$

It is easy to see that

$$\begin{aligned} (B \wedge C) \vee (\neg B \wedge D) &= \boxminus K_0 \\ B \oplus C \oplus D &= \boxminus K_{20} \end{aligned}$$

Thus the constant (K_0/K_{20}) is canceled out in both cases and a round collision occurs.

[1] We use the boxed plus and minus symbols $\boxminus$ and $\boxplus$ to denote twos complement addition and subtraction operations $\bmod 2^{32}$. $\boxminus x$ can be read as $0 \boxminus x$. Other symbols denote are word-wise binary operators as follows: $\neg$ not, $\wedge$ and, $\vee$ or, and $\oplus$ exclusive-or.

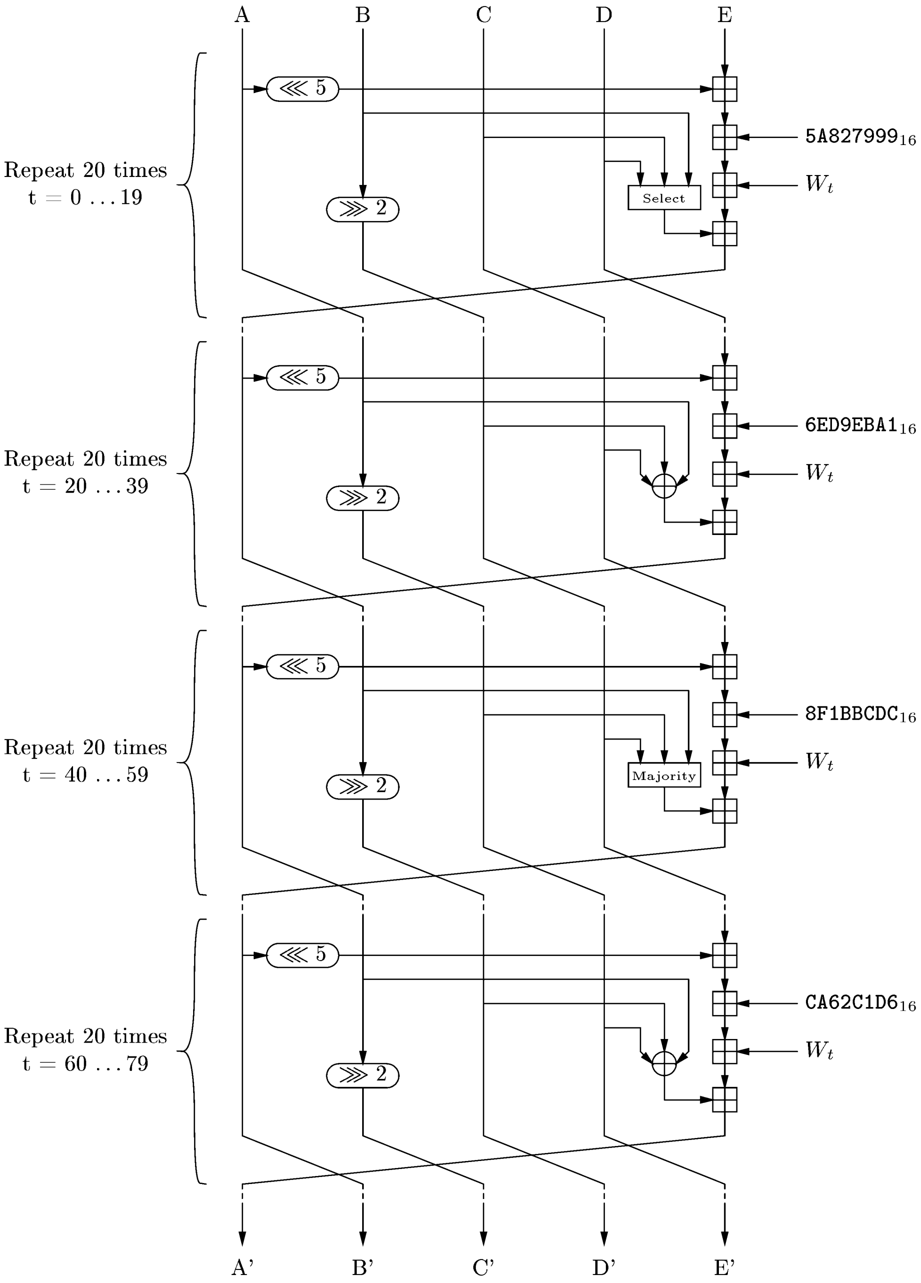
A
B
C
D
E
Repeat 20 times
t = 0 . . . 19
⋘ 5
≫ 2
Select
5A827999$_{16}$
W_t
Repeat 20 times
t = 20 . . . 39
⋘ 5
≫ 2
6ED9EBA1$_{16}$
W_t
Repeat 20 times
t = 40 . . . 59
⋘ 5
≫ 2
Majority
8F1BBCDC$_{16}$
W_t
Repeat 20 times
t = 60 . . . 79
⋘ 5
≫ 2
CA62C1D6$_{16}$
W_t
A'
B'
C'
D'
E'

Fig. 1. SHACAL.

Similarly, for iteration pair 39/40 (parity-majority transition) we use:

$$(B, C, D) = (\boxminus K_{20}, \boxminus K_{40}, \boxminus K_{40})$$

Again we see that a collision occurs:

$$\begin{aligned} B \oplus C \oplus D &= \boxminus K_{20} \\ (B \wedge C) \vee (C \wedge D) \vee (B \wedge D) &= \boxminus K_{40}. \end{aligned}$$

Keying. The key-expansion LFSR is sparse. This helps us to stretch the 16-word span of the key schedule window to cover two collisions at iterations 20 and 40.

We note that all 80 key words can be easily computed from any 16 consecutive words of the expanded key. In our attack we choose keys $W_{21 \ldots 36}$. We start by forcing a collision at iteration 20 and then running the cipher forward to iteration 25.

We then pick (A, B, C, D, E) after iteration 38 so that a collision occurs at iteration 40, and then run the cipher backwards to iteration 28. Key words $W_{21} \ldots W_{24}$ and $W_{29} \ldots W_{36}$ are set to zero. Therefore

$$\begin{aligned} W_{37} &= (W_{33} \oplus W_{29} \oplus W_{23} \oplus W_{21}) \lll 1 = 0 \\ W_{38} &= (W_{34} \oplus W_{30} \oplus W_{24} \oplus W_{22}) \lll 1 = 0. \end{aligned}$$

Note that W_{39} and W_{40} do not affect collision at iteration 40.

We can choose four keying words $W_{25 \ldots 28}$ without messing up the two round collisions! Unfortunately we cannot control all five words so we are forced to use a large lookup table to find a match the fifth word of the running state. This works because SHA-1 requires five iterations before all words are non-linearly changed.

After we have collisions in iterations 20 and 40, and sixteen keying words $W_{21 \ldots 36}$, we simply run the compression function forward to iteration 59/60 and see if a collision occurs there also. Since two of the three necessary collisions can be found with essentially $O(1)$ effort and the third requires $O(2^{32})$ operations, the overall complexity of finding slid pairs is $O(2^{32})$. The method has been implemented; see the Appendix for an example of a slid pair for full SHACAL-1.

This is a surprising property, but we are have not discovered a direct way to transform it into a practical attack against SHA-1 and SHACAL.

3 Block Ciphers Based on MD5

We may also consider block ciphers derived from other dedicated hash functions. One obvious candidate is MD5 [18]. The MD5 compression function consists of four “rounds”, each of which has 16 iterations. Because each of the 64 iterations has a different constant, the MD5 compression function doesn’t seem to be subject to sliding attacks. Figure 2 shows the structure of a single MD5 iteration.

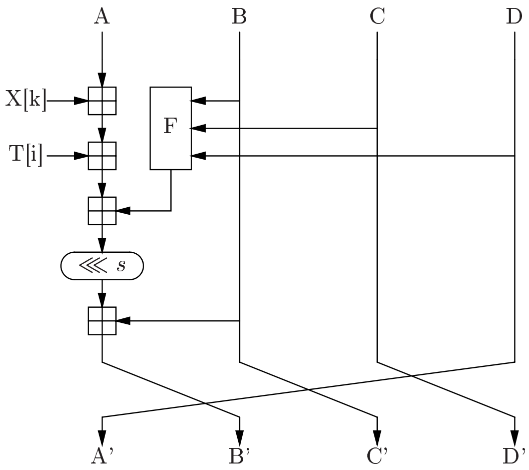

Fig. 2. The MD5 iteration.

The MD5 compression function is known to not be collision-resistant [8]. This indicates the existence of "fixed points" in the corresponding block cipher but doesn't really tell us much its security. Some differential cryptanalysis of the hashing mode has been attempted, but we are not aware of any cryptanalysis of MD5 in encryption mode [2]. However, there exists at least one high-probability differential characteristic:

$$\begin{aligned} P \oplus P' &= \texttt{80000000 80000000 80000000 80000000} \\ &\downarrow \\ E(P,K) \oplus E(P',K) &= \texttt{80000000 80000000 80000000 80000000} \end{aligned}$$

With probability 2^{-16} this characteristic will penetrate 16 iterations of rounds 1, 2, and 4. The characteristic holds with $P = 1$ for round 3, yielding a total product probability of 2^{-48}. Note that if the chaining variable is added, the output xor becomes zero. This attack is closely related to the collision attacks discussed in [6].

3.1 Message Digest Cipher

The Message Digest Cipher (MDC) encryption mode for iterated hash functions was proposed by Gutmann in 1993 and is used in his Secure FileSystem software (in conjunction with the SHA-1 compression function) [9]. MDC can be defined as

$$\begin{aligned} C_0 &= IV \\ C_i &= P_i \oplus (E(C_{i-1}, K) \boxplus C_{i-1}) \quad \text{for } i = 1, 2, \ldots, n. \end{aligned}$$

Here the boxed plus symbol ($\boxplus$) denotes wordwise addition modulo 2^{32}, IV is an initialization vector, P_i are the plaintext blocks, and C_i are the corresponding ciphertext blocks. If we ignore the addition operation, MDC is equivalent to running the compression function in CFB (cipher feedback) mode.

The decryption operation can be written as:

$$P_i = C_i \oplus (E(C_{i-1}, K) \boxplus C_{i-1})$$

This allows us to select the input to the compression function. Using the differential characteristic described in the previous section, we can distinguish MDC-MD5 from a "perfect" 128-bit block cipher in CFB mode with about 2^{48} blocks (2^{55} bits) in a chosen ciphertext attack. Computational cost of key recovery can be substantially reduced using this, and other, differential properties of the MD5 compression function. Such attacks depend on the details of key scheduling mechanism used.

3.2 The Kaliski-Robshaw Cipher

Another proposal based on MD5 is the Kaliski-Robshaw cipher [13]. The main purpose of the proposal apparently was to activate discussion of very large block ciphers. However, the paper gave enough details about this proposal for us to mount a cryptanalytic attack. This cipher has 8192-bit blocksize and its basic iteration is closely related to that of MD5. However, the overall structure is radically different.

It turns out that flipping bit 26 (`0x04000000`) of one of the 256 plaintext words will result in equivalence of at least 64 ciphertext words with experimental probability $0.096 = 2^{-3.4}$. This is due to the fact that this particular bit often only affects three words in first round. In each of the consequent rounds the number of affected words only can only quadruple, resulting $3 * 4 * 4 * 4 = 192$ affected words in the end, and leaving 64 words untouched.

This immediately leads to a distinguisher requiring no more than dozen chosen plaintext blocks. Analysis of key recovery attacks is made a little bit more difficult by the sketchy nature of the description of key schedule. If we assume that the key can be effectively recovered from permutation P, we believe that a key recovery attack will not require more than 2^{16} chosen plaintext blocks and neglible computational effort.

4 Conclusions

We have presented attacks against block ciphers that have been directly derived from dedicated hash functions. Section 2 discusses slide attacks against SHA-1 and SHACAL-1 and section 3 describes simple attacks against MDC-MD5 and the Kaliski-Robshaw cipher.

Compression functions are meant to be only ran in one direction. The security properties of compression functions can be different when ran in the opposite direction ("decryption"). Furthermore a key-scheduling mechanism suitable for a dedicated hash function may be insufficient for a block cipher.

Based on evidence in hand, we assert that since the design criteria of compression functions and block ciphers are radically different, adaptation of even a secure compression function as a block cipher is often not a wise thing to do.

Acknowledgements

The author would wish to thank Kaisa Nyberg, Helger Lipmaa, Matt Robshaw, and several anonymous reviewers for useful comments.

References

1. R. Anderson. *The Classification of Hash Functions.* Proc. Codes and Cyphers: Cryptography and Coding IV, Institute of Mathematics & Its Applications, pp. 83 – 93. 1995.
2. T. A. Berson. *Differential Cryptanalysis Mod* 2^{32} *with Applications to MD5.* Advances in Cryptology – Proc. Eurocrypt '92, LNCS 0658. pp. 71 – 80. Springer-Verlag, 1993.
3. J. Black, P. Rogaway, and T. Shrimpton *Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV* Advances in Cryptology – Proc. Crypto '02. LNCS 2442, pp. 320 – 335. Springer-Verlag, 2002.
4. A. Biryukov and D. Wagner. *Slide attacks* Proc. Fast Software Encryption '99, LNCS 1636. pp. 245 – 259. Springer-Verlag, 1999.
5. A. Biryukov and D. Wagner. *Advanced Slide Attacks.* Advances in Cryptology – Proc. Eurocrypt '00, LNCS 1807. pp. 589 – 606. Springer-Verlag, 2000.
6. B. den Boer and A. Bosselaers. *Collisions for the compression function of MD5.* Advances in Cryptology – Proc. Eurocrypt '93, LNCS 0765. pp. 293 – 304. Springer-Verlag, 1994.
7. I. Damgård. *A Design Principle for Hash Functions.* Advances in Cryptology – Proc. Crypto '89, LNCS 0435. pp. 399 – 416. Springer-Verlag, 1990.
8. H. Dobbertin. *Cryptanalysis of MD5 Compress.* Presented at Eurocrypt '96 rump session, May 14, 1996.
9. P. C. Gutmann. *SFS Version 1.0 Documentation.* Available form `http://http://www.cs.auckland.ac.nz/~pgut001/sfs/`
10. H. Handschuh, L. R. Knudsen, and D. Naccache. *Analysis of SHA-1 in Encryption Mode.* Proc. RSA Cryptographers' Track 2001, LNCS 2020. pp. 70 – 83. Springer-Verlag, 2001.
11. H. Handschuh and D. Naccache. *SHACAL.* Submission to the NESSIE project, 2000. Available from `http://www.cryptonessie.org`.
12. H. Handschuh and D. Naccache. *SHACAL: A Family of Block Ciphers.* Submission to the NESSIE project, 2002. Available from `http://www.cryptonessie.org`.
13. B. S. Kaliski and M. J. B. Robshaw. *Fast Block Cipher Proposal.* Proc. Fast Software Encryption 1993, LNCS 0809. pp. 33 – 40. Springer-Verlag, 1994.
14. Ralph Merkle. *One Way Hash Functions and DES.* Advances in Cryptology – Proc. Crypto '89, LNCS 0435. pp. 428 – 446. Springer-Verlag, 1990.
15. R. K. Nichols and P. C. Lekkas. *Wireless Security – Models, Threats, and Solutions.* McGraw-Hill, 2002.
16. B. Preneel, R. Govaerts, and J. Vandewalle. *Hash Functions Based on Block Ciphers: A Synthetic Approach.* Proc Crypto '93. LNCS 0773, pp. 368 – 378. Springer-Verlag, 1993.
17. B. Preneel. *Cryptographic Primitives for Information Authentication – State of the Art.* State of the Art in Applied Cryptography. Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 1997. Revised Lectures. LNCS 1528, pp. 49 – 130. Springer-Verlag, 1998.

18. R. Rivest. *The MD5 Message-Digest Algorithm* Network Working Group RFC 1321, 1992.
19. U.S. Department of Commerce. *FIPS PUB 180-1: Secure Hash Standard.* Federal Information Processing Standards Publication, April 1995.
20. U.S. Department of Commerce. *FIPS PUB 180-2: Secure Hash Standard*, Draft Federal Information Processing Standards Publication, 2001.
21. D. Wagner. *A Slide Attack on SHA-1.* Unpublished manuscript and personal communication. June 4, 2001.

A A Slid Pair for SHA-1

The algorithm for finding slid pairs was implemented in the C programming language (588 lines). Test run required roughly 2 hours of CPU time on a 1 GHz Pentium III computer (GCC/Linux).

Triplet A.

$$(A, B, C, D, E) = \texttt{02AAD5C2~DC766713~19C66B2F~7CEAE5B1~CC08CC0B}$$

$$W_{0\ldots15} = \begin{array}{l}\texttt{8DA3F8F6~BBA5050C~99D3C3DC~BBA5050C~99D3C3DC}\\ \texttt{E42BAFB3~37DF640F~1ABABEEA~8DA3F8F6~E42BAFB3}\\ \texttt{37DF640F~B57DEBB5~5AA5AB1F~44ED8DA0~1B63271F}\\ \texttt{EAE12A73}\end{array}$$

$$(A', B', C', D', E') = \texttt{FC56BE44~03A42CDA~F68056F0~960F5286~32985CD9}$$

Triplet B.

$$(A, B, C, D, E) = \texttt{4258DA7D~02AAD5C2~F71D99C4~19C66B2F~7CEAE5B1}$$

$$W_{0\ldots15} = \begin{array}{l}\texttt{BBA5050C~99D3C3DC~BBA5050C~99D3C3DC~E42BAFB3}\\ \texttt{37DF640F~1ABABEEA~8DA3F8F6~E42BAFB3~37DF640F}\\ \texttt{B57DEBB5~5AA5AB1F~44ED8DA0~1B63271F~EAE12A73}\\ \texttt{BA7C9CF9}\end{array}$$

$$(A', B', C', D', E') = \texttt{58BB28F0~FC56BE44~C0E90B35~F68056F0~960F5286}$$

It is easy to see that Triplet B has been slid “right” by one position compared to Triplet A. The message block has been slid “left” correspondingly.

Output arrays (A', B', C', D', E') include the final addition of chaining variable. If this final “chaining” is removed, this is also a slid pair for SHACAL-1.