Collision attack against Tav-128 hash function

Fajar Hariyanto and Bety Hayat Susanti
Sekolah Tinggi Sandi Negara
E-mail: harifajar1994@gmail.com, bety.hayat@stsn-nci.ac.id

Abstract. Tav-128 is a hash function which is designed for Radio Frequency Identification (RFID) authentication protocol. Tav-128 is expected to be a cryptographically secure hash function which meets collision resistance properties. In this research, a collision attack is done to prove whether Tav-128 is a collision resistant hash function. The results show that collisions can be obtained in Tav-128 hash function which means in other word, Tav-128 is not a collision resistant hash function.

1. Introduction
Hash function is a cryptographic technique that can be used to ensure the integrity and authenticity of data [8]. It is divided into two types, i.e. Message Authentication Code (MACs) or keyed hash functions and Modification Detection Codes (MDCs) or unkeyed hash function. As the input, MACs requires a secret key and a message, while MDCs just requires a message [6].

In 2008, Tav-128 hash function is designed for an authentication protocol Radio Frequency Identification (RFID) [10]. Tav-128 is a hash function based on Merkle-Damgard construction [11]. It is expected to be a hash function that fulfill the cryptographic security service [4].

A good hash function must satisfy three properties, i.e. preimage resistance, second preimage resistance, and collision resistance. Collision resistance means that it is computationally difficult to obtain two different inputs that have the same hash value [2]. Based on that, collision attack against Tav-128 was performed in 2010 by Kumar and Sanadhya [3]. The results show that collisions found in Tav-128.

In this research, collision attack will be applied on Tav-128 hash function with the different method from [3]. The goal of this research is to know the resistance of Tav-128 against collision attack.

2. Theoretical Background
2.1. Hash Function
Hash function maps an arbitrary length input to the fixed length output [5]. Based on [9], hash function which is denoted by \( h(m) \) must provide the following properties:

(i) Compression: for any input \( m \), output \( y = h(m) \) is small. In practice, the hash function produces output with a fixed size, regardless of the length of input, with the length of a typical output in the range of 128 to 512 bits.

(ii) Efficiency: it must be efficient to calculate \( h(m) \) of any input \( m \). Computational effort depends on the length of \( m \).

(iii) One-way: it is difficult to invert the hash, i.e. if \( y \) is given, it is difficult to find the value of \( m \) where \( h(m) = y \).
(iv) Weak collision resistance: given \( m \) and \( h(m) \), it is difficult to find \( w \), with \( w \neq m \), where \( h(w) = h(m) \).

(v) Strong collision resistance: it is hard to find a pair of \( m \) and \( w \), with \( m \neq w \), where \( h(m) = h(w) \).

2.2. Merkle-Damgard construction

Merkle-Damgard construction is the construction of a hash function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) constructed by iterating the compression function \( f : \{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\}^n \). The process is carried out as follows [7]:

(i) Pad and split the message \( M \) into \( k \) blocks \( m_1, ..., m_k \) with each block measuring \( t \) bits.

(ii) Set the value of \( h_0 \) as initialization value (IV).

(iii) Calculate \( h_k = f(h_{k-1}, m_k) \) for each message block.

(iv) Obtained output \( h^f(M) = h_k \).

Padding is usually done by adding a bit of ”1” followed by bits ”0” as much as needed to complete the \( k \) block. See [7] for more details discussion.

2.3. Tav-128

Tav-128 is a hash function that is designed for Radio Frequency Identification (RFID) authentication protocol. It has 160-bits internal state and generate 128-bits output [4].

The hash value of the input \( m \) of length \( k \times 32 \) bits obtained by calculating the value of output of the compression function \( f \) that is iterated \( k \) times by Merkle-Damgard construction and transformation \( g \). The compression function \( f \) constructed by four functions, i.e. \( A, B, C, \) and \( D \) functions. Each of \( A, B, C, \) and \( D \) functions process 64-bits input and produce 32-bits output. The output of the compression function \( f \) on the last block becomes the input to the transformation \( g \), so that the resulting output hash value that is \( 4 \times 32 \) last bits of input values at the transformation \( g \) [4]. The structure of Tav-128 hash function shown in Figure 1. See [4] for more details.

2.4. Collision Attack On Tav-128

In [4], Kumar et al. stated that the collision resistance of the hash function which has Merkle-Damgard construction affected by the compression function. Therefore, collision attack on Tav-128 is done by utilizing the compression function of it. Collision attack on Tav-128 can be applied by following steps [4]:

(i) Make a correspondence table with the size of \( 2^{32} \) for all messages \( m \) consisting of \( (h_0, h_1, m) \). \( h_0 \) and \( h_1 \) are output values after one application of \( C \& D \) functions.
(ii) Sort this table.
(iii) Find a pair in a row where $h_0$ and $h_1$ have the same value.

3. Research Method
In general, the collision attack in this research is based on Kumar et al. [4]. But the difference from Kumar et al., there is a grouping of $C&D$ functions output into classes based on a range of values. There are 64 classes where the range between the classes is $4000000000000000$ (hexadecimal). The first class that is $0000000000000000 - 03ffffffffffff$ and so on until the 64th class is $fc00000000000000 - ffffffff0fffffff$. The grouping is useful to find collisions at the output of $C&D$ functions efficiently.

After grouping is done, then sort the output values in each class. The sorting technique used in this research is merge sort [1]. The use of sorting is aimed to make the finding of collisions in each class becomes easier and faster. In details, to do collision attack against Tav-128, we use the following steps:

(i) Calculate the output from one application of $C&D$ functions using $2^{32}$ inputs.
(ii) Group all of the output values into classes according to a range of values.
(iii) Sort the output in each class with the order of the smallest value to the largest value.
(iv) After all the class is sorted, find output pairs which have the same value in each class.
(v) Keep the same output values into an output column ($h_0||h_1$) in correspondence table.
(vi) To determine the input that has the same output based on the correspondence table, generate all pairs of input and output values only to the class that contains the same output values.
(vii) Find the input values that produce output based on the correspondence table.
(viii) Finally, the values of the corresponding input has been found and accommodated into inputs ($m$) and outputs ($h_0||h_1$) column in the correspondence table.

This research was performed by implementing Tav-128 using C programming language and MinGW Dev C++ Compiler. The number of inputs used in this attack are $2^{32}$ which have 32 bits length of each input without padding process.

4. Results and Analysis
After the application of the collision attack using $2^{32}$ inputs against Tav-128, we obtain the collision result. Table 1 shows collision results of $2^{32}$ inputs. Class column shows list of classes that are used in grouping the output values that obtained from one application of $C&D$ functions. Output range column indicates the range of output values in each class. Collision column shows the number of collisions that obtained in each class.

From Table 1, it can be seen that the collision can be obtained at class 43 and 47. At class 43, we obtain one collision which means that there is one different inputs pair which have the same output value. In class 47 was obtained one collision. So that the total collision obtained are two collisions as shown in Table 2.

A collision can be obtained easily because output values from one application of $C&D$ functions, which are grouped into each class has been sorted. Hence, the process used to find collisions in each class becomes $2^{26}$ (because it is already divided into 64 classes).

With the discovery of a collision in the compression function $f$, then it will be shown how to find a collision in Tav-128 with a starting collision on the compression function $f$. From collision attack on $C&D$ functions, we have obtained two inputs that have the same output value in one application of them. Figure 2 shows that $m_1^1$ is a notation for input 1 and $m_2^1$ is a notation for input 2 where $m_1^1 \neq m_2^1$. 
Table 1. Collision Result of $2^{32}$ Input On First Iteration Of $C&D$ Function.

| Class | Output Range | Collisions |
|-------|--------------|------------|
| 1     | 000000000000000 - 03ffffffffffffffffff | 0          |
| 2     | 040000000000000 - 07ffffffffffffffffff | 0          |
| 3     | 080000000000000 - 0bffffffffffffffffff | 0          |
| 4     | 0c0000000000000 - 0fffffffffffffffffff | 0          |
| 5     | 100000000000000 - 13ffffffffffffffffff | 0          |
| ...   | ...          | ...        |
| 40    | 9c0000000000000 - 9fffffffffffffffffff | 0          |
| 41    | a0000000000000 - a3ffffffffffffffffff | 0          |
| 42    | a4000000000000 - a7ffffffffffffffffff | 0          |
| 43    | a8000000000000 - abffffffffffffffffff | 1          |
| 44    | ac0000000000000 - afffffffffffffffffff | 0          |
| 45    | b0000000000000 - b3ffffffffffffffffff | 0          |
| 46    | b4000000000000 - b7ffffffffffffffffff | 0          |
| 47    | b8000000000000 - bbffffffffffffffffff | 1          |
| 48    | be0000000000000 - bfffffffffffffffffff | 0          |
| 49    | c0000000000000 - c3ffffffffffffffffff | 0          |
| 50    | c4000000000000 - c7ffffffffffffffffff | 0          |
| ...   | ...          | ...        |
| 60    | ec0000000000000 - efffffffffffffffffff | 0          |
| 61    | f0000000000000 - f3ffffffffffffffffff | 0          |
| 62    | f4000000000000 - f7ffffffffffffffffff | 0          |
| 63    | f8000000000000 - fbffffffffffffffffff | 0          |
| 64    | fc0000000000000 - fffffffffffffffffff | 0          |

Table 2. Correspondence Table Of Input Pairs That Has The Same Output Value.

| No.  | Inputs (m) | Outputs (h0||h1) from one application of $C&D$ functions |
|------|------------|--------------------------------------------------------|
| 1    | 8fa00d2b   | abdf190b20c8f dac                                       |
|      | fb86cc90   | abdf190b20c8f dac                                       |
| 2    | 49390f23   | bb8e44aa1472b576                                       |
|      | 8099ba12   | bb8e44aa1472b576                                       |
In this stage, we want to prove that the compression function is not collision resistance, resulting that Tav-128 is also not collision resistance. We use the method by adding some value behind each inputs \((m^1_1\) and \(m^2_1\)) so that each input size of 256 bits. The use of 256-bits input only to indicate the property of compression on Tav-128. Two inputs after additional bits are:

Input 1 = \(m^1_1||m^1_2||m^1_3||m^1_4||m^1_5||m^1_6||m^1_7||m^1_8\), and

Input 2 = \(m^2_1||m^2_2||m^2_3||m^2_4||m^2_5||m^2_6||m^2_7||m^2_8\).

By processing the compression function \(f\) each of 32 bits, then input 1 and input 2 are processed within 8 blocks.

• First block:
  Assuming that \(m^1_1\) and \(m^2_1\) are two inputs which have the same output value of \(C&D\) functions so that:
  \[f(m^1_1, a^0_0) = f(m^2_1, a^0_0) = (a^1_0, S^1[0 \ldots 3])\]

• Second block:
  \[f(m^1_2, a^1_0) = (a^2_0, S^2[0 \ldots 3])\]
  \[f(m^2_2, a^0_0) = (a^2_0, S^2[0 \ldots 3])\]
  So that \(f(m^1_2, a^1_0) = f(m^2_2, a^0_0) = (a^2_0, S^2[0 \ldots 3])\) because \(m^1_2 = m^2_2\).

• Third block:
  \[f(m^1_3, a^2_0) = (a^3_0, S^3[0 \ldots 3])\]
application of $C$ that $Tav_{128}$ is also not collision resistance. Thus, we can conclude

Based on (1) and (2) we can see:

1. **Fourth block:**
   \[
   f(m_4^2, a_0^3) = (a_0^4, S^4[0 \ldots 3])
   \]
   So that $f(m_4^2, a_0^3) = f(m_3^2, a_0^3) = (a_0^3, S^3[0 \ldots 3])$ because $m_4^2 = m_3^2$.

2. **Fifth block:**
   \[
   f(m_5^2, a_0^3) = (a_0^4, S^4[0 \ldots 3])
   \]
   So that $f(m_5^2, a_0^3) = f(m_4^2, a_0^3) = (a_0^3, S^3[0 \ldots 3])$ because $m_4^2 = m_5^2$.

3. **Sixth block:**
   \[
   f(m_6^2, a_0^3) = (a_0^4, S^4[0 \ldots 3])
   \]
   So that $f(m_6^2, a_0^3) = f(m_5^2, a_0^3) = (a_0^3, S^3[0 \ldots 3])$ because $m_6^2 = m_5^2$.

4. **Seventh block:**
   \[
   f(m_7^2, a_0^3) = (a_0^4, S^4[0 \ldots 3])
   \]
   So that $f(m_7^2, a_0^3) = f(m_6^2, a_0^3) = (a_0^3, S^3[0 \ldots 3])$ because $m_7^2 = m_6^2$.

5. **Eighth block:**
   \[
   f(m_8^2, a_0^3) = (a_0^4, S^4[0 \ldots 3])
   \]
   So that $f(m_8^2, a_0^3) = f(m_7^2, a_0^3) = (a_0^3, S^3[0 \ldots 3])$ because $m_8^2 = m_7^2$.

Furthermore, the output of the compression function $f$ of eighth block becomes input of the transformation $g$. The output of the transformation $g$ is a hash value of each input on $Tav_{128}$, so that: Hash value for input 1 is

\[
g(a_0^8, S^8[0 \ldots 3]) = S^8[0 \ldots 3] \quad (1)
\]

Hash value for input 2 is

\[
g(a_0^8, S^8[0 \ldots 3]) = S^8[0 \ldots 3] \quad (2)
\]

Based on (1) and (2) we can see:

\[
Tav - 128(m_1^5 \parallel m_2^5 \parallel m_3^5 \parallel m_4^5 \parallel m_5^5 \parallel m_6^5 \parallel m_7^5 \parallel m_8^5) = Tav - 128(m_1^5 \parallel m_2^5 \parallel m_3^5 \parallel m_4^5 \parallel m_5^5 \parallel m_6^5 \parallel m_7^5 \parallel m_8^5) = S^8[0 \ldots 3] \quad (3)
\]

Finally, it is proven that input 1 and input 2 have the same hash value to $Tav_{128}$ hash function. This indicates that the compression function on $Tav_{128}$ is not a collision resistance. Thus, we can conclude that $Tav_{128}$ is also not a collision resistance.

Table 3 shows examples of two different 256-bits inputs which have the same output value of one application of $CkD$ functions. Then, if the two inputs are added by any bits at the end, it will still have the same hash value.
Table 3. Collisions For 256 Inputs Of Tav-128 Hash Function.

| No. | Inputs (m) 256 bits | Outputs from one application of $C&D$ functions | Hash value $H(M)$ 128 bits |
|-----|---------------------|---------------------------------------------|-----------------------------|
| 1   | $8fa00d2b\ldots$   | $abdf190b$                                  | $44d71501\ldots$           |
|     | $d66f56ab\ldots$   | $20c8fdac$                                  | $fca69199\ldots$           |
|     | $afde5459abffe\ldots$ | $44d71501\ldots$                          | $fca69199\ldots$           |
| 2   | $49390f23\ldots$   | $bb8e44aa$                                  | $6e6b26\ldots$             |
|     | $aaaaaaa\ldots$    | $1472b576$                                  | $9308ee\ldots$             |
|     | $3030303ababababab$|                                            |                             |
|     | $1f1f1f1fdedcedea$ | $bb8e44aa$                                  | $6e6b26\ldots$             |
|     | $8099ba12\ldots$   | $1472b576$                                  | $9308ee\ldots$             |
|     | $aaaaaaa\ldots$    |                                            |                             |
|     | $3030303ababababab$|                                            |                             |
|     | $1f1f1f1fdedcedea$ |                                            |                             |

5. Conclusion
In this paper, collision attack on Tav-128 was performed. The results show that it is obtained two collisions from the total of $2^{32}$ inputs. We use different methods to do collision attack on Tav-128. Those methods are output grouping and merge sorting. Collisions can be obtained with $2^{26}$ computing. This proves that this method is faster and easier compared with the actual computing which is $2^{128}$. Finally, it can be concluded that the Tav-128 is not a collision resistant hash function.

Acknowledgment
This research was supported by Sekolah Tinggi Sandi Negara, Bogor, West Java. We would like to thank to the reviewers for their valuable comments.

References
[1] Dalal A C 2004 Searching and Sorting Algorithms Supplementary Lecture Notes
[2] Menezes A J, van Oorschot P C and Vanstone S A 1997 Handbook of Applied Cryptography (Boca Raton : CRC Press LLC)
[3] Kumar A and Sanadhya S K 2010 Attacking the Tav-128 Hash Function IIIT-Delhi Technical Report
[4] Kumar A, Sanadhya S K, Gauravaram P, Safkhani M and Naderi M 2010 Cryptanalysis of Tav-128 Hash Function 11th Int. Conf. Crypto. INDOCRYPT 2010 LNCS 6212
[5] Ege B 2010 Structural Testing Of Hash Function Applied Mathematics Of Middle East Technical University
[6] Preenel B 2003 Analysis and Design of Cryptographic Hash Functions Thesis Katholieke Universiteit Leuven
[7] Andreea E, Bouillaguet C, Fouque P, Hoch J J, Kelsey J, Shamir A and Zimmer 2008 Second Preimage Attack on Dithered Hash Functions 27th Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2008 LNCS 4965
[8] Saarinen M O 2009 Cryptanalysis of Dedicated Cryptographic Hash Functions  Departement of Mathematics, University of London

[9] Stamp M and Richard L. M 2007 Applied Cryptanalysis Breaking Ciphers in the Real World  (John Wiley & Sons : San Jose State University)

[10] Lopez P P, Castro J C C, Tapiador J M E and Ribagorda A 2008 An Efficient Authentication Protocol for RFID Systems Resistant to Active Attacks  Computer Science Department, Carlos III University of Madrid

[11] Lopez P P 2008 Lightweight Cryptography in Radio Frequency Identification (RFID) Systems  Computer Science Department, Universidad Carlos III De Madrid