ABSTRACT
In this paper, we present the sha1 compression function and the Merkel and Haifa mode of operation, we then proceed to describe the principle of the differential attack on the compression function and of the dean’s attack on the mode of operation.

Keywords
Hash function, Sha, differential path, local collision, Merkel, Haifa

1. INTRODUCTION
Hash function are an algorithm that maps data of variable length to data of a fixed length. This fixed length value is called a digest and has many different use. They are used in authentication, encryption, signature schemes and serve as a building block in many higher-level cryptographic functionality. In order to be of any use, a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ need to fulfill the following requirement:

- One-way. Given $h \in \{0, 1\}^n$, finding a preimage $M$ such that $H(M) = h$ should not be significantly faster than hashing $2^n$ random messages with $H$.
- Second-Preimage Resistant. Given $M \in \{0, 1\}^*$, finding a second preimage $M'$ such that $H(M) = H(M')$ should not be significantly faster than hashing $2^n$ random messages with $H$.
- Collision-Resistant. Finding two distinct messages $M$ and $M'$ such that $H(M) = H(M')$ should not be significantly faster than evaluating $H$ about $2^{n/2}$ times.

A hash function is considered secured if the most efficient attack known on it is the brute force attack. We will present two type of attacks in this paper. The first is an attack on the collision-resistance of sha1 using differential path in order to lower its complexity. The second is an attack on the mode of operation.

The complexity of the attacks will be counted in terms of sha1 compression function call.

2. DIFFERENTIAL ATTACK ON THE COMPRESSION FUNCTION OF SHA1

2.1 Sha1 presentation
Sha1 is not a standard hash function anymore, but it was and is still widely used. We will now explain quickly how the sha1 function works. The initial message is cut into blocks of 512 bits. Each block is processed in the compression function (see figure 1) that we will call $H$.

$$H : \{0, 1\}^{512} \times \{0, 1\}^{160} \rightarrow \{0, 1\}^{160}$$ (1)
The compression function takes two arguments: the current message block and the result of the previous call to the compression function. (the digest obtained with the previous message block) The first call to the compression function use an initial value (IV) defined by the standard FIPS-180-2. first of all, the current block message is cut in 16 words of 32 bits each numbered from 0 to 15: \( W_0, \ldots, W_{15} \). These 16 words are then extended in 80 words according to the following formula:

\[
W_i = W_{i-16} \oplus W_{i-14} \oplus W_{i-8} \oplus W_{i-3} \ll 1, 16 \leq i \leq 79
\]  

(2)

Where \( \ll \) means a circular permutation of 1 bit to the left. The second step is to cut the intermediate hash \( H_{i-1} \) where \( t \) is the number of block that have already been processed in five 32 bits integers in order to obtain the initial state \( (A_0, B_0, C_0, D_0, E_0) \) of the five registers that represent the internal state of \( H \). The following transformation is then applied 80 times, \( i \) from 0 to 79:

\[
\begin{align*}
A_{i+1} &= (A_i \ll 5 + f_i(B_i, C_i, D_i) + E_i + K_i + W_i) \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \ggg 2 \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i
\end{align*}
\]

(3)

Where the \( K_i \) are predefined constants and the \( f_i \) are boolean function defined table 1. The last step is called Feed-Forward: The sum modulo \( 2^{32} \): \((A_0 + A_{80}) \oplus (B_0 + B_{80}) \oplus (C_0 + C_{80}) \oplus (D_0 + D_{80}) \oplus (E_0 + E_{80}) \) are concatenated in order to form the result of the compression function: \( H_t \)

Our goal is to find two messages which will updates differently the registers \( A_i, B_i, C_i, D_i, E_i \) but ultimately find the same \( A_{80}, B_{80}, C_{80}, D_{80}, E_{80} \) and so the same digest.

### 2.2 What’s a local collision

The differential cryptanalysis was invented by Eli Biham and Adi Shamir. The main goal of the differential cryptanalysis was to perturb the input of a function in a certain way and to observe the change in the output of the function.

**Definition 1.** The differential path is a set of differences that will be applied to a message in order to construct another message. What we want to obtain is a differential path that will give us, with a good probability, a collision or a near collision.

**Definition 2.** the introduction of a perturbation in a \( W_i \) can be corrected by the introduction of other perturbations in the \( W_{i+1}, \ldots, W_{i+6} \). These perturbations are called correction. The set of these perturbations form a differential path which leads to the same state of the registers \( A, B, C, D \) and \( E \) or local collision.

In order to find what sort of differential path we need to construct to find local collisions, we are going to construct a linear version of the sh\( 1 \) compression function. The update of the register is simplified because we get rid of all the nonlinear things like the addition modulo \( 2^{32} \) or the AND. Here is the new formula:

\[
\begin{align*}
A_{i+1} &= (A_i \ll 5 + B_i \oplus C_i \oplus D_i \oplus E_i \oplus K_i \oplus W_i) \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \ggg 2 \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i
\end{align*}
\]

(4)

The second simplification of this ideal model of the sh\( 1 \) compression function is that we can change all the \( W_i \) without caring about the message expansion. This simplified sh\( 1 \) represent the ideal behavior of the sh\( 1 \) function. The local collision we will define later have a probability of 1 of happening. The goal is to find a local collision that also has a good probability of happening with the real sh\( 1 \) compression function.

### 2.3 Use of the local collisions to find a real collision

The construction of a collision is based on the use of the local collisions. Because of the expansion of the message (the \( W_i \) for \( 16 \leq i \leq 79 \)) the bits we used in order to create the local collisions perturb the result. We will describe the local collision with their first step and the bit that is modified. We then introduce the concept of disturbance vector which represents a set of local collision. Let \( V' = (V_0, \ldots, V_{79}) \) be this disturbance vector. It is build with 80 words of 32 bits

<table>
<thead>
<tr>
<th>( x \ldots x' )</th>
<th>(0,0)</th>
<th>(1,0)</th>
<th>(0,1)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( x )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( u )</td>
<td>-</td>
<td>( \checkmark )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( n )</td>
<td>-</td>
<td>-</td>
<td>( \checkmark )</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( # )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>( \checkmark )</td>
<td>-</td>
<td>( \checkmark )</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>-</td>
</tr>
<tr>
<td>( A )</td>
<td>-</td>
<td>( \checkmark )</td>
<td>-</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( B )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( C )</td>
<td>-</td>
<td>-</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( D )</td>
<td>( \checkmark )</td>
<td>-</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>( E )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
</tbody>
</table>

Table 2: Notations used in [2] in order to represent the different states possibles for a couple of bits \( x \) and \( x' \)

<table>
<thead>
<tr>
<th>( \text{pas} )</th>
<th>( \text{differential path of } W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \ldots --------)</td>
</tr>
<tr>
<td>1+1</td>
<td>( \ldots --------)</td>
</tr>
<tr>
<td>1+2</td>
<td>( \ldots --------)</td>
</tr>
<tr>
<td>1+3</td>
<td>( x\ldots x'--------)</td>
</tr>
<tr>
<td>1+4</td>
<td>( x\ldots x'--------)</td>
</tr>
<tr>
<td>1+5</td>
<td>( x\ldots x'--------)</td>
</tr>
<tr>
<td>1+6</td>
<td>( x\ldots x'--------)</td>
</tr>
</tbody>
</table>

Table 3: differential path leading to a local collision
Table 1: Boolean functions and constants used in SHA-1

<table>
<thead>
<tr>
<th>ronde</th>
<th>pas i</th>
<th>f(B,C,D)</th>
<th>K_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 ≤ i ≤ 19</td>
<td>f_{IF} = (B \wedge C) \oplus (B \wedge D)</td>
<td>0x5a827999</td>
</tr>
<tr>
<td>2</td>
<td>20 ≤ i ≤ 39</td>
<td>f_{XOR} = B \oplus C \oplus D</td>
<td>0xed6ebea1</td>
</tr>
<tr>
<td>3</td>
<td>40 ≤ i ≤ 59</td>
<td>f_{MAJ} = (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)</td>
<td>0x8fabbcde</td>
</tr>
<tr>
<td>4</td>
<td>60 ≤ i ≤ 79</td>
<td>f_{XOR} = B \oplus C \oplus D</td>
<td>0xca62c1d6</td>
</tr>
</tbody>
</table>

Figure 2: mean value on 1000 tries of the number of call to the compression function in order to obtain a local collision.

According to the following formula:

V_i : \begin{cases} 
(2^j) & \text{if we initiate a local collision at step i on the j^{th} bit} \\
0 & \text{else} 
\end{cases} \quad (5)

In order to find a real collision, the disturbance vector must follow the same recurrence formula as the expansion of the message because we don’t have direct control over the expanded words of the message.

Local collision are fundamental in this attack, so we tried to understand how the complexity of the research of local collision evolves through time. The theory says that the research of a local collision has a probability of $2^{-5}$ of success. We tried to find out how the placement of another local collision impacts the probabilities. The result is that in the 16 first steps of the compression function, the complexity is higher than if the two local collisions were independant, but the result is not far from an independant behavior. However, if we mix the local collision, meaning that we begin a collision before the other one is finished, then many composition create either impossible differential paths or non reachable complexity of those differential paths.

Here is an exemple of a local collision on sha1 restricted to 16 steps. In this version a local collision equals a collision. M1:42c1985c a276c272 95f75311 dd761e95 ffca17cd fb-dadf42 22e15e73 cf17b345 d7987630 6a66e555 feac4e16 f7922bc4 30369068 14035a36 e7ad7158 27b5fc9 c9 M2:42c1985c a276c270 95f75351 dd761e97 7fca17cd 7bdadf42 22e15e73 cf17b345 d7987630 6a66e555 feac4e16 f7922bc4 30369068 14035a36 e7ad7158 27b5fc9 c9 Hash de M1:67b790f5 3667a8e6 7fe3007b fc69d7e6 c720efdc Hash de M2:67b790f5 3667a8e6 7fe3007b fc69d7e6 c720efdc

Figure 3: mean value on 1000 tries of the number of call to the compression function in order to obtain two separated local collision.

The greatest difficulty in this attack is to take into consideration the expansion of the message. Several techniques exist in order to find good disturbance vector that will take into consideration all the factors and have a reachable complexity. A collision on sha1 has yet to be found, but this type of attack give a better complexity than the brute force attack and the collisions found on reduced version of sha1 are little by little affecting more and more of the steps of the compression function. The last collision found is on 70 steps in the function. 10 to go and sha1 will be cracked. That’s why a new standard has been decided in order not to perturb the security of the authentication the day it will happen.

3. ATTACK ON THE MODE OF OPERATION

3.1 Presentation of Merkel and Haifa

As we talked about the Sha-1 security weakness, we will know focus on the mode of operation of hash functions.

3.1.1 Some definitions

Throughout this paper, we will use some notations. n will denote the digest size, and H will denote the hash function.

3.1.2 Merkle-Damgard

Until Sha-3 competition, the mode of operation used was built with the seminal works of Merkle and Damgard in 1989 and called "Merkle-Damgard construction". In this mode of operation, the hash function is built by iterating a compression function

\[ f : \{0 : 1\}^n \ast \{0 : 1\}^m \rightarrow \{0 : 1\}^n. \quad (6) \]

The hash process works this way:

- Pad and split the message M you want to hash into r blocks $x_0 ... x_{r-1}$ of m bits each.
• Set $h_0$ to the initial value $IV$.
• For each block, compute $h_i = f(h_{i-1}, x_i)$.
• Output $H(M) = h_r$.

The Padding scheme used is pretty simple:
• Pad a single bit of 1.
• Pad as many 0 as necessary such that the length of the padded message is congruent modulo $n$ to $n-t$.
• Pad the message length encoded in $t$ bits.

The following scheme represents the Merkle-Damgard construction:

![Diagram of Merkle-Damgard construction]

### 3.1.3 HAIFA

Some results showed that the Merkle-Damgard construction gives collision resistant hash functions (if the compression function is already collision resistant). However, in 2007, Biham and Dunkelman proposed another mode of operation harder to break, called Hash Iteration FrAmework (HAIFA).

This mode of operation works on the same principle than Merkle-Damgard construction, except that the compression function has more input and the padding scheme is slightly different. Here, the compression function $f$ takes into account the number of bits hashed so far, and salt is also added to the arguments. Thus, we have, for each block $x_i, h_i = f(h_{i-1}, x_i, \#bits, salt)$. The padding scheme is also a bit different and works this way:

• Pad a single bit of 1.
• Pad as many 0 as necessary such that the length of the padded message is congruent modulo $n$ to $(n-(t+r))$.
• Pad the message length encoded in $t$ bits.
• Pad the digest size encoded in $r$ bits.

Obviously, if the size of the last block is greater than $n-(t+r)$, an additional block is created, so the message length and the digest size can be padded. We note that in this case, the compression function is called on this block with $\#bits=0$.

This changes to the mode of operation make second preimage attacks more difficult on the HAIFA framework than on the Merkle-Damgard construction, as Bouillaguet and Fouque demonstrated in their paper. Indeed, because of the values added to the compression function, a second preimage adversary can’t test all the $n$ values $h_i$, because each block has a different image from $f$ depending on his position in the global message.

### 3.2 Theoretical results on security

#### 3.2.1 Fix-Points based attacks

Assume that the compression function $f$ is such that finding fix-points is easy (i.e finding $(h, M)$ satisfying $h = f(h, M)$), and this is the case for most of the compression function used so far. Then it is easy to find collisions on the hash function, and these fix-points are the base of Dean’s attack on iterative mode of operation. This attack has three main steps:

1. Finding $O(2^{n/2})$ fix-points, noted $F=(h,m)$
2. Selecting $O(2^{n/2})$ single blocks and computing their chaining value denoted by $C = f(IV, m, m)$
3. Finding a collision between a fixed point and a chaining value, i.e between $F$ and $C$.

Once such a collision is found (let’s note the colliding chaining value $h, m_1$, the message block corresponding in $F$ and $m_2$ the block associated with $h$ in $C$). Then, the attacker repeats this attack, starting from the message $m_2||m_1$. Then he can fix the length of the message by iterating the fixed point as many time as necessary.

This attack is however much harder to use on HAIFA, because of the $\#bits$ input in the compression function. Indeed, when the attacker has found a fixed point (i.e $(h, m)$ such that $h = f(h, m, \#bits, salt)$), he cannot concatenate this fixed point to itself to fix the length, because it is highly unlikely that it can be repeated, the $\#bits$ value having changed.

Dean’s attack is one of the pitfalls of Merkle-Damgard construction corrected by HAIFA, thanks to the adding of inputs in the compression function.

#### 3.2.2 Generic attacks on the mode of operation

In their paper, Bouillaguet and Fouque focused on the second preimage resistance. They considered a second preimage resistance problem, based on an adversary $A$ having access to an oracle on the compression function $F$. $A$ receives a randomly generated message $M$ divided in $l$ blocks and tries to find a message $M'$ with $H(M') = H(M)$, with $H$ an iterative mode for $F$ (such as HAIFA or Merkle-Damgard).

Bouillaguet and Fouque established that in this conditions, the probability for such an adversary to break $H$ after at most $q$ queries in lower-bounded by a value $\epsilon$. They proved that, for Merkle-Damgard construction,

$$\epsilon \leq q.l/2^n$$  \hspace{1cm} (7)$$

And, for HAIFA,

$$\epsilon \leq q/2^{n-1}$$  \hspace{1cm} (8)$$

Where $n$ is the size of the digest produced by $H$. 
3.3 Experimental results found by our implementation

In this paper, we will try to experimentally demonstrate Bouillaguet and Fouque’s results about second preimage resistance of haifa. Throughout the paper, we will consider a simplified implementation of haifa mode of operation. Actually, we will consider the hash function as a simple random function. The thing is, we will imagine this function has been computed taking into account the salt value and the #bits value we described before. In our implementation, this doesn’t change anything, but we will use this result in the search of collisions.

The brute force attack we made uses a special structure to show up this pseudo-dependance. For the Merkle-Damgard scheme, we kept in memory all the n intermediate values given by f (all the $h_i$). Thus, when we try to find a collision using random messages, we can compare them to all of these intermediate hash. For the Haifa model, we considered this was impossible, and each random message can only be compared to a unique hash value, due to the additional components of the compression function. Moreover, for obvious computing power reasons, we decided to limit the size of the digest. Thus, our work is limited to 16 bits (two bytes). But the results can be extended to longest messages. The principle is the same, but the time necessary for computation was too big for our personal machines.

The next graphics summarize the results we obtained and make a comparison between haifa and a classical Merkel-Damgard scheme for second preimage resistance. The first graphic represents the number of queries necessary to find a collision in the first two bytes of the hash image in a merkel-damgard scheme, while the second one represents the same values for HAIFA.

This results were obtained on a sample of about 1,000 adversaries.

As we can see, finding a collision on the first byte is much harder for a haifa mode of operation than for a classical Merkle-Damgard scheme. In the Merkle-Damgard, the median value of the number of queries necessary to find a collision on the first byte is about 5,845 queries, whereas for HAIFA this value is about 18,549 queries.

3.4 Explanation of these result

This results can be extended to a fix-points attack. Unfortunately, we didn’t have the possibility to implement such an attack, but we could imagine a comparison between a Dean’s attack on HAIFA and on Merkle-Damgard. The problem would be slightly different: Here, we use the fact that a collision on an intermediate hash value is more difficult to exploit on HAIFA than on Merkle-Damgard, because on the #bits input. To operate such a collision, we would need it to be on blocks at the same position in messages of same length.

In a fixed points attack, the problem is to find fixed points. And, given that the inputs of the compression function change depending on the position of the actual block, a fixed point for the i-th block is unlikely a fixed point for the $i+1$-th. On the other hand, a fixed point for the i-th block in a Merkle-Damgard construction is always a fixed point for the $i+1$-th block (because the compression function is independent of the position in the message).

4. CONCLUSIONS

The differential attacks on the compression function and the Dean’s attack on the mode of operations are the two main
attacks launched on the sha1 but also sha2 and md5 hash functions. This vector of attack can't be ignored because it brings lower than the brute force attacks complexity, and thus jeopardizes the security of the hash function. Countless upgrades on these attacks are found each year, in order to make them more efficient and faster. That's why there was a need of a new standard: sha3 that was thought to counter these attacks. Shal has yet to be cracked, but it is not safe anymore, because methods with reachable complexity exist and it's only a matter of time before a collision is found.

5. ANNEXES

5.1 Finding a collision on a Merkle-Damgard construction

/*fonction de comparaison
 *elle compare les premiers bits
 *de la requete de l'attaquant
 *et du hach du message initial */
void Compare(Context *c1, Context *c2, int *tab, int t) {
  int i = 0;
  //valeur initiale, on compare les 1 premiers bits
  //au debut : nbbits vaut
  //1000 0000 0000 0000...
  uint32_t nbbits = 0x80000000;
  //valeur de laquelle on va augmenter notre nombre
  //pour ne garder que les
  //premiers bits
  uint32_t puissance = 0x40000000;
  uint32_t mask = 1;
  for (uint32_t j = 0; j < c1->length; j++) {
    while (((c1->intermediate_hash[j][0] & mask) == (c2->final_hash[0] & mask)) && i<31)
    {
      //printf("%x\n", nbbits);
      //operation de decalage, si les n premiers
      //bits sont &l;gux, on compare les
      //n+ieme bits
      nbits += puissance;
      puissance = puissance >> 1;
      //si le tableau vaut 0, c'est qu'on a
      //pas encore trouve de collisions
      //jusqu'a ce nombre de bits
      if (tab[i] == 0){
        //affichage_binaire(mask);
        tab[i] = t;
        //printf("collisions des %i premiers bits
        //: %i tours. \n", i+1, t);
        //affichage_binaire( c2->final_hash[0]);
      } i++;
      mask = 1<<i;
    }
  }
  //tableau qui contiendra le nombre de requetes
  //necessaires pour obtenir les i premiers bits
  //identiques
  int indices[32] = {0};
  int tours = 1;
  Context contest;
  srand(time(NULL));
  //attaque par force brute,
  //on cherche un message ayant un hash identique a
  //1/8 des hash intermediaires du message de base
  while (indices[7] == 0){
    generate_rand_message(&contest);
    contest.final_hash[0] = (uint32_t) rand();
    contest.final_hash[1] = (uint32_t) rand();
    contest.final_hash[2] = (uint32_t) rand();
    contest.final_hash[3] = (uint32_t) rand();
    contest.final_hash[4] = (uint32_t) rand();
    Compare(&context, &contest, indices, tours);
    tours++;
  }
  for (int i = 0; i<8; i+1){
    printf("%i %i\n", i, indices[i]);
  }
  return 0;
}

6. BIBLIOGRAPHY
[1] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lembert and William Jalby, LNCS 3494 Collisions of SHA-0 and Reduced SHA1
Wang, Yin and Yu, LNCS 3621 Finding collision in the full sha1
Bouillaguet and Fouque, Practical Hash Functions Constructions Resistant to Generic Second Preimage Attacks Beyond the Birthday Bound
FIPS 180-1 norme sha1