SHA-256 PAPER

Eduard Torres Chaves and Juan José Solano Quesada

***Abstract*—Define what's the SHA-256 cryptographic method, what it does and how it works. Emperimentation on how long it takes for the method to generate a result for different input sizes.**

***Index Terms*—hash, cryptographic, bit, logic operation**

I. Introduction

In this paper we will talk about the study and experimentation of the SHA-256 hash function, is important to know that SHA-256 isn't an encryption algorithm, it is a 'one-way' cryptographic function. SHA-256 is a member of the SHA-2 family, which was designed by the United States National Security Agency (NSA).

It's one of the successor hash functions to SHA-1, but is not much complex.The uses of SHA-256 goes from hash tables, integrity verification, challenge handshake authentication to digital signatures, etc.

The fact of being a cryptographic hash functions is their collision resistance: nobody should be able to find two different input values that result in the same hash output.

SHA-256 creates an 256-bit key (from there comes the name), that make it perfect partner-function for AES(Advanced Encryption Standard) and other's alike.

II. Methodology

The chosen model is the qualitative one, because it allows us to use experimentation to obtain the necessary data to reach conclusions about the function, we will make distinct experiments around the Java code of the SHA-256 function, once we finish the tests, we will compare then to know the trend of the function with different sizes of input text and the time spent in "digesting" them.

heil The string list used in the experiments is: ["abc","test", "one after one", "Heil Hitler", "today is gonna be a beautiful day", "I don't think you trust In my self righteous suicide I cry when angels deserve to die!"].

In the figure 1, is the result of a python code that counts the number of characters which has each string in the previous list, the difference in numbers of characters is important for the test diversity(the presence of uppercase and lowercase letters should not affect the digesting time).

Next is the Java code (figure 2) that is used in the test realization, the code has a string which is digested and comes out in a byte vector, which is manipulated to be transformed into a readable string.

E. Torres Chaves Computer Engineering student, Instituto Tecnológico de Costa Rica

J.J. Solano Computer Engineering student, Instituto Tecnológico de Costa Rica

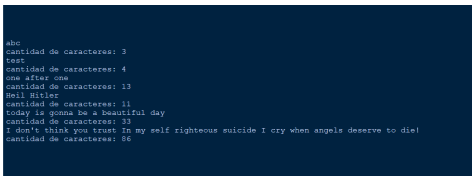
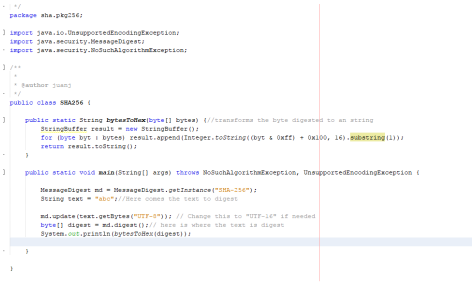


Fig. 1. Result of the character counter in Python



```
package sha.pkg256;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/**
 *
 * @author juanj
 */
public class SHA256 {

    public static String bytesToHex(byte[] bytes) {//transforms the byte digested to an string
        StringBuffer result = new StringBuffer();
        for (byte byt : bytes) result.append(Integer.toString((byt & 0xff) + 0x100, 16).substring(1));
        return result.toString();
    }

    public static void main(String[] args) throws NoSuchAlgorithmException, UnsupportedEncodingException {

        MessageDigest md = MessageDigest.getInstance("SHA-256");
        String text = "abc";//Here comes the text to digest

        md.update(text.getBytes("UTF-8")); // Change this to "UTF-16" if needed
        byte[] digest = md.digest();// here is where the text is digest
        System.out.println(bytesToHex(digest));
    }
}
```

Fig. 2. SHA-256 Java Code

III. Experimentation

The algorithm gets the message from the input and processes it to get the bits of it, then adds a number 1 to the end and appends number 0s to make the message a 512 multiple length and finally the length of the message is appended at the end in 64-bit big-endian format. Then, for every 512 bit chunk a series of logical operations (shown in figure 3) are applied to the first 16 words of the chunk to extend the words across the entire 512 bit segment.

$$\begin{aligned}
\mathrm{Ch}(E,F,G) &= (E \wedge F) \oplus (\neg E \wedge G) \\
\mathrm{Ma}(A,B,C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\
\Sigma_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22) \\
\Sigma_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)
\end{aligned}$$

Fig. 3. Logical operations performed to 512 bit chunks

The complexity used to generate the hash code is $O(2^{64})$ because of the size of the word that divides every chunk, which is 64 bits.

IV. Results

V. Discussion

VI. Conclusions