Scientific Research Publishing

Open Journal of Applied Sciences, 2023, 13, 1581-1587
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Numeric Identifier Transmission Algorithm Using Hash Functions

Vladyslav Kutsman

Computer Sciences Department, Information Technologies and Computer Engineering Faculty, Vinnytsia National Technical University, Vinnytsia, Ukraine
Email: kutsmanvlad@gmail.com

How to cite this paper: Kutsman, V. (2023) Numeric Identifier Transmission Algorithm Using Hash Functions. *Open Journal of Applied Sciences*, **13**, 1581-1587.
https://doi.org/10.4236/ojapps.2023.139125

Received: August 30, 2023
Accepted: September 23, 2023
Published: September 26, 2023

Abstract

When developing programs or websites, it is very convenient to use relational databases, which contain powerful and convenient tools that allow to work with data very flexibly and get the necessary information in a matter of milliseconds. A relational database consists of tables and records in these tables, each table must have a primary key, in particular, it can be a number of BIGINT type, which is a unique index of a record in the table, which allows to fetch operation with maximum speed and O (1) complexity. After the operation of writing a row to the table of database, the program receives the row identifier ID in the form of a number, and in the future this ID can be used to obtain this record. In the case of a website, this could be the GET method of the http protocol with the entry ID in the request. But very often it happens that the transmission of an identifier in the clear form is not safe, both for business reasons and for security reasons of access to information. And in this case, it is necessary to create additional functionality for checking access rights and come up with a way to encode data in such a way that it would be impossible to determine the record identifier, and this, in turn, leads to the fact that the program code becomes much more complicated and also increases the amount of data, necessary to ensure the operation of the program. This article presents an algorithm that solves these problems "on the fly" without complicating the application logic and does not require resources to store additional information. Also, this algorithm is very reliable since it is based on the use of hash functions and synthesized as a result of many years of work related to writing complex systems that require an increased level of data security and program performance.

Keywords

Cryptography, Security, Coding, Hash Functions, Algorithms, Fintech, Banking, Golang, PostgreSQL

1. Introduction

When working with data in relational databases, the fastest data access is guaranteed when the primary key PK is a unique number. In this case, the complexity will always be O (1), which in turn guarantees the maximum speed of obtaining data on a specific record from the table. In most of the existing databases, this value is auto increment, that is, it automatically increases by one each time than, in fact, uniqueness is guaranteed. For example, if this is a website database, then when creating a product in the database, it is assigned a unique record identifier (ID) and to obtain data on the product, its identifier must be passed. But the situation is different when it comes to a resource where some kind of information security or distributed access is needed. That is, when a certain object should be accessed only by a certain group of users or by a single user, or in the case of a site on the order page, arise need to hide the order number that the competitor could not determine the turnover of the online store per day. In this case, the transmission of the identifier in the clear form is not suitable, it is necessary to exclude the possibility of random selection of the key or deliberate selection by an attacker. Or a bank client's web-site, when the client enters the web-page and sees the data on his agreements and for detailed information he needs to go to the agreement page, in this case, if the program operates with an identifier in clear form, then the ability to access the entity (in this case, the entity will be client agreement) will be unsafe because by substituting a different number, it possible to see information about the agreement that is assigned to another client. But in order to avoid such situation, needs checks access rights on the server side, which in a certain sense complicates the application code, or encode identifiers in such a way that it would be impossible to pick up a contract identifier. As a matter of fact, these problems are solved by the algorithm proposed in this article, namely, the task of checking access rights to the object and the task of encrypting the key so that it would be impossible to select the value of the key and that, upon receipt of the key, its values could be obtained without searching for matches in the database, which in turn eliminates the need for unnecessary queries to the database, in the case of creating tables responsible for aliases of these identifiers to records or cache in RAM, and this, significantly increases the performance of the program. It is also very important that this method is based on the use of hash functions [1], which make it possible to calculate the hash sum and create a unique identifier that can be represented as a string that consists of digits from the hexadecimal number system, which is a very convenient and time-tested cryptographic way to obtain a checksum. Within the framework of this article, the sha1 encoding algorithm is used, based on the convenience of representing the hash code, presented as a string, it has a length of 40 characters, but also possible to choose a newer encoding algorithm, the algorithm does not depend on a specific cryptographic algorithm.

2. Description of the Algorithm

In this section described the algorithm for generating an encrypted string from a

number and the algorithm for obtaining a number from an encrypted string. The idea and implementation is quite simple, which makes this method very reliable. For example, need to get a cipher for the number 123012. To do this, must be an array of digit positions of the original number, for example [3] int{1,1,2}, shift values and a special string that acting as a salt, this can be a string like "hash ^& _= +!2wes". The number itself is also represented as a string, then the hash of the string is calculated using the sha1 cryptographic algorithm, that is, in this case it will be sha1 ("123012" + "hash^& _= +!2wes") = de2c68e64ac5f0d194ac50e2450052ddb54266e3, then this hash is added the initial number at the positions that are specified in the array of positions, taking into account the shift, which indicates the number of characters to be skipped. In the case of this example, the following string will be obtained: de2c1628e36041ac25f0d194ac50e2450052ddb54266e3 having a length of 46 characters, that is, 40 characters is the length of the hash string itself (if it is encoded by the sha1 algorithm) and 6 characters is the length of the number character by character, the positions of the digits of the encoded number are marked in red, the important thing is that the first digit of the number is inserted after 1 character, taking into account the shift value, the second counts 1 from the last inserted third 2 from the last inserted and the fourth is already 1 and so on in a circle until the whole number is written to the hash string, after the encoded value of the number is formed, it can be transmitted as a value an identifier for a specific object on a browser web page. To get a number from a hash string, the program works in the following way in a loop concatenates the characters located at the positions specified in the array of positions, taking into account the shift value, until the length of the string is equal to the length, which is equal to the length of the incoming hash code minus the length of the string of the algorithm itself, as written earlier in the case of sha1, this is 40 characters. Then the resulting string is passed through the encoding function and checked for a match in the hash values, and in case of replacing/deleting/adding at least one character, the function will return an error. Program 1 implemented in the programming language Golang [2] implements the algorithm described above, which allows to more clearly understand the essence.

Program 1. Basic algorithm for creating a hash string from number and getting a number from a string

In this program, the **NumberDecoder** structure has 4 fields. **Salt** is a field that stores a string that acts as a salt, **HashLength** is a field indicating how long the value is generated by the hash algorithm, **Sequence** is an array of digit positions of the number to be encrypted and transmitted, **Shift** is the initial shift value after which the calculation of positions for writing digits begins the original number into a hash string. As shown in the main function, the **getCode** method returns a string for the number 123012 which is then decoded into a number using the **getId** method to get the original value.

type NumberDecoder **struct** {

```
        Salt string
        Shift int
        HashLength int
        Sequence []int
}
func (N *NumberDecoder) getCode(value string) string {
        h := sha1.New()
        h.Write([]byte(value + N.Salt))
        b := h.Sum(nil)
        hash := fmt.Sprintf("%x", b)
        Res := hash[:N.Shift]
        j, k, i := 0, 0, 0
        for _, v := range hash[N.Shift:] {
                Res += string(v)
                k++
                if i < len(value) && k == N.Sequence[j] {
                        Res += string(value[i])
                        i++
                        k = 0
                        if j+1 < len(N.Sequence) {
                                j++
                        } else {
                                j = 0
                        }
                }
        }
        return Res
}
func (N *NumberDecoder) getId(value string) int64 {
        LN := len(value)
        if LN <= N.HashLength {
                return 0
        }
        Res := ""
        j := 0
        i := N.Shift + N.Sequence[j]
        for i < LN && len(Res) < LN-N.HashLength {
                Res += string(value[i])
                if j+1 < len(N.Sequence) {
                        j++
                } else {
                        j = 0
                }
```

```
            i += N.Sequence[j] + 1
        }
        if value != N.getCode(Res) {
            return 0
        }
        number, err := strconv.ParseInt(Res, 10, 64)
        if err != nil {
            return 0
        }
        return number
    }
    func main() {
        var NumberDecodeEncode NumberDecoder
        NumberDecodeEncode.Salt = " hash^& _= +!2wes"
        NumberDecodeEncode.HashLength = 40
        NumberDecodeEncode.Shift = 3
        NumberDecodeEncode.Sequence = []int{1, 1, 2}
        // result is de2c1628e36041ac25f0d194ac50e2450052ddb54266e3
        fmt.Println(NumberDecodeEncode.getCode(fmt.Sprintf("%v",
123012)))
        // result is 123012
        fmt.Println(NumberDecodeEncode.getId("de2c1628e36041ac25f0d194
ac50e2450052ddb54266e3"))
    }
```

3. Discussion

This algorithm allows to transfer numeric identifiers in clear form with a checksum calculated for a specific identifier. An example could be the following link https://[examples-domen.com]/acts/de2c1628e36041ac25f0d194ac50e2450052ddb54266e3 pointing to a contract or a file or any other object. The link can be on the site page itself in the href attributes of the "a" tag or the src attributes of the "img" tag in the site's html markup. The main advantage of this algorithm is its simplicity and high reliability, as well as a significant increase in the speed of the program. For the purpose of this article, the sha1 cryptographic algorithm is used to obtain the hash code, but as mentioned earlier, newer cryptography algorithms can be used. It is also very important that at any time possible change the value of the Salt field and, accordingly, the value of the hash code will already be completely different. In the same way, possible create a positional array to accommodate the digits of the original number, which makes it difficult for an attacker to decode, also its possible change the value of the initial shift. Also, with this method of data transfer, as it was written earlier, alias tables for objects in the database or a special cache in RAM are not required. Another very important aspect is that references to the same object can be change very often by changing the value of the Salt field, changing the value of the initial shift, or

changing the values of the Sequence positional array, but not necessary to overwrite the values in the database or the values in the cache, because that encoding and decoding going on directly on the fly. An example of this can be a client account in a bank, implemented as a website where each user should have access only to the resources to which he has access (contracts, files). And when using this algorithm, and the server receives the object identifier, the need for an additional check of access rights to the object is eliminated, because it is impossible to generate a link to the identifier without information about what shift, positional array value and salt. Another example of the application of this algorithm can be an order page on an online store website, where each site visitor should have access only to his orders, and when generating a link to an order, the proposed algorithm eliminates the need for additional verification of access rights to view an object when it is requested. It is also very important that the program operates exclusively with numeric identifiers for objects, which guarantees the fastest selection operations by the primary key from the database. Another not unimportant thing is the simplification of the logic of building an application, because such tasks as checking access rights to an object and storing additional data do not even arise, they are solved at the stage of generating an identifier from a code line. Also, this algorithm can be modified by adding a suffix or prefix to the input string, which can confuse when trying to crack. It is possible add information bytes directly to the hash string (adding it to the salt first) itself, which significantly expands the information which can be crypted and safely transmit.

4. Results

This chapter presents the results of applying the algorithm in a real application (CRM system of a leasing company) requiring separate access to data and changing the value of a numeric identifier.

CRM system Table 1 stores aliases for task identifiers for office employees. The indexed code field contains hash values obtained by the sha1 function that identify the entry. To store table and index data, 186 Mb is used for 1 million records, and when checking the existence of a value in a table, at best, 31,007,000 nanoseconds of time are needed. When using the algorithm, there is no need to allocate space for storing data, since the value is formed "on the fly" and the numeric value of the identifier is written to the hash string, and the time needed to check the received code for validity is 0 nanoseconds. The situation is slightly different when using a cache in RAM and put data for 1 million records in RAM, then in this case the time to check the value will be 0 nanoseconds, as in the case of using the algorithm, but data storage will require 24,000,016 bytes for 1 million records. It is very important that in the given examples the memory values are specified for the sha1 cryptographic algorithm, and in the case of using cryptographic algorithms that form longer string keys, the memory consumption will increase, and in the case of using the algorithm, the length of the hash string does not matter. The reliability of the algorithm is guaranteed and depends on the cryptographic algorithm that is used to form the hash string.

Table 1. Aliases for linking an object with its identifier.

Column	Description
Id	Bigint PK
Code	Character varying (40) - token

5. Conclusions

The advantages of this algorithm are as follows:

1) No need to store aliases in the database or in the cache storage of RAM.

2) Ability to frequently change object references, which improves reliability.

3) No need to check access rights to an object if the program must provide separate access to certain objects.

4) Easy to implement.

5) Use of time-tested cryptographic algorithms, which guarantees the reliability of the algorithm.

6) Ability to use various cryptographic algorithms, no dependence on a specific cryptographic algorithm.

7) A significant increase in the performance of the program/site due to the absence of the need to check access rights and the absence of a match search.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Rescorla, E. (2011) US Secure Hash Algorithms (SHA and SHA-Based HMAC and HKDF). RFC 6234.

[2] Donovan, A.A.A. and Kernighan, B.W. (2015) The GO Programming Language. Addison-Wesley, Boston, 380 p.