On The Generation of Alphanumeric One Time Passwords

Shubham Srivastava
Department of Computer Science and Engineering
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham
Amrita University, India
shubhamsri810@gmail.com

Sivasankar M
Department of Mathematics
Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham
Amrita University, India
m_sivasankar@cb.amrita.edu

***Abstract*—One Time Password (OTP) is a type of password that is valid for only one time use (use them once and throw them away). It is an automatically generated string of numbers that will authenticate a user for a single session or transaction. In this paper we propose a way of generating alphanumeric one time passwords (OTPs) using automaton theory with linear functions which enhances the level of security.**

***Index Terms*—One Time Password, Automata Theory, Linear Functions, Hashing**

I. Introduction

Present day communications totally dependent on the internet for almost all kinds of transactions that includes net banking, session initialization and online payments. For security reasons with every login/transaction the user has to authenticate himself to the server. Most of the security concerns of numerous industries are controlled by a password mechanism [1]. Systems rely on static passwords for the verification of user's identity. However, use of static passwords is not advisable anymore as it comes with major security issues. Any hacker/user can easily crack, guess, steal the passwords (through key loggers or shoulder surfing) and can use them to gain access to anybody's private network and can steal their sensitive email, financial information.

So, for a transaction to be secure, users proposed various encryption/authentication methods [2-3]. OTP is one of the authentication method (dynamic password) and by far the most successful way to provide entry for a user. In OTP schema, passwords are generated at the server side and sends to the user, so that the user can send back the received password to complete the online transaction successfully. Even if an attacker succeeds in acquiring an OTP, he may not be able to predict the next OTP. This greatly reduces the risk of an unauthorized user/intruder gaining access to the account. One of the most important advantages of OTP over static passwords is that it is not vulnerable to replay attack.

There are different methods by which we can generate OTPs. Some of which are (i) Time based OTPs (TOTPs) , in which a password is calculated from the shared secret key and the current time (ii) HMAC based OTPs (HOTPs) which are calculated on the HMAC of shared secret and a counter.

II. Conventional OTP Generators

Lamport's Algorithm [4] is one of the familiar techniques for generating OTPs. Lamport's algorithm is a mathematical algorithm which generates a sequence of values called "passkey". It is a sequence in which each successive value depends on the previous value. It has a registration phase and an authentication phase. Registration is done only once whereas authentication will be executed every time the user logs on to the system.

In registration phase, the client begins by choosing a seed value say X, when registering in the system. The server calculates $F^M(X)$ equals $F(F(\cdots F(X)))$ using the seed X (F is a hash function), for say 1000 iterations and stores the entire sequence. When the user requests a logon for the first time, the server passes the 1000th value, $F^M(X)$ through a secure communication channel. When the user sends back the value say θ, the server authenticates the user only if $F(\theta)$ equals $F^M(X)$ and the next process is executed. If not the server sends $F^{M-2}(X)$ and the process is repeated and the server denies the users login request after *k* unsuccessful user side attempts (usually *k* is 3). The next time the user request a login, the last unused OTP in the list is send by the server and the process is repeated. Fig. 1 explains this process.

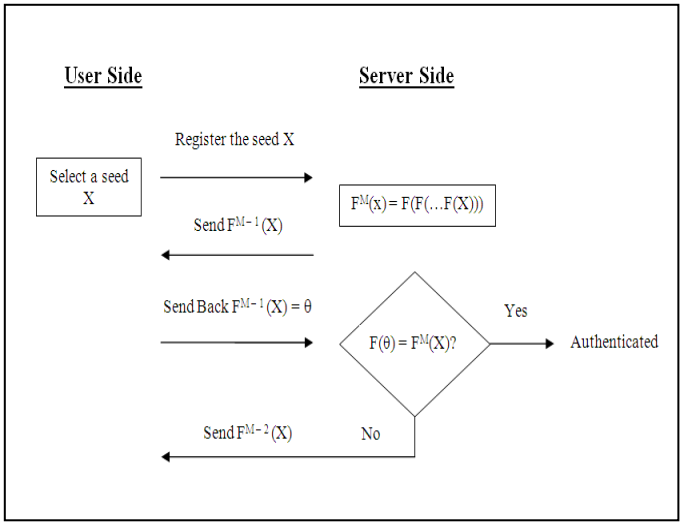

Fig.1. Lamport's algorithm

When the chain of Lamport's OTPs runs out (say, after all the 1000 OTPs are used), the client must create a new seed and send it to the server to calculate a new chain. For a variant of the above scheme [5] can be referred.

III. PREREQUISITES

For generating alphanumeric OTPs we use automation theory and affine transformation. In the forthcoming subsections we give a brief recall of both.

A. The Theory of Automata

The building block of the theory of formal languages is the theory of Automata. It is a theoretical branch of computer science which involves the study of abstract machines. Automata are generally used to model the hardware of a computer. An automaton consists of states and transitions. As a automaton sees an input symbol it makes a transition to another state depending on the transition function to give a final output. It may also have some temporary storage and can make decision in transforming input in to the output. It plays a vital role in compiler design, formal verification in text processing and in hardware design.

Formal language theory is a part of Automata theory. An automaton is a finite representation of a formal language. A formal language is an abstraction of general characteristics of programming language. It consists of a set of symbols and some rules of formation in which these symbols can be combined in to entities called sentence. We note that symbols are simply characters and are an abstraction which is meaningless when they stand alone. In general a language is a set of words from a given alphabet which is in turn a set of finite symbols. A special kind of automata specifically defined for a language is called a formal grammar.

A deterministic finite automaton consists of five components $(Q, \Sigma, \delta, q_0, F)$ where Q is a set of all the states in which a transition is possible, Σ is a set of symbols/alphabets, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function, q_0 is the starting state, F is a set of acceptable final states of Q.

We refer the reader [6] for a wide study of these concepts.

B. Affine Transformation

Very often we want to move from one domain to another domain for achieving simplicity and for operational convenience. Linear transformations are very useful in this regard with an additional advantage of preserving the characteristics of the quantities being transformed. In general a linear transformation of '*n*' quantities $x_1, x_2, \ldots, x_n$ to '*m*' quantities $y_1, y_2, \ldots, y_m$ is achieved by an *m* by *n* matrix $A = [a_{ij}]$, i = 1,2…m and j= I,2,…n The linear equations associated with this transformation can be written in matrix form as,

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix} = AX = \begin{pmatrix} a_{11} & a_{12} \cdots & a_{1n} \\ a_{21} & a_{22} \cdots & a_{2n} \\ \cdots & \cdots \cdots & \cdots \\ a_{m1} & a_{m2} \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$$

and in equation form as,

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ y_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ &\cdots \cdots \cdots \cdots \\ y_m &= a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n . \end{aligned}$$

If the linear equations are independent (in other words if det(A) is nonzero) we can define the inverse transformation also as below.

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} = A^{-1}Y = \begin{pmatrix} a_{11} & a_{12} \cdots & a_{1n} \\ a_{21} & a_{22} \cdots & a_{2n} \\ \cdots & \cdots \cdots & \cdots \\ a_{m1} & a_{m2} \cdots & a_{mn} \end{pmatrix}^{-1} \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix}.$$

In particular if $m = n$ we say the transformation is a square transformation, as the matrix A will be a square matrix [7].

As all of the present day digital communications are done through binary numbers, we will be concentrating on transformations with coefficients from $Z_2 = \{0,1\}$. In other words the coefficient matrix A will be a binary matrix (a zero one matrix) and the usual addition '+' will be replaced by the XOR operation '⊕'. In general, as every entry can be assigned any of the two values 0 or 1, discarding the all-zero-matrix (the zero transformation) there are $2^{mn} - 1$ transformations possible. Out of these, if we restrict down to linearly independent square transformations, then we have $(2^n - 1)(2^n - 2)$.

C. One Way Hash Functions

One way hash function is one of the important tools in cryptography [8]. A one way function is a mathematical function which has an image, y and it is difficult to find pre image, x such that f(x) = y. One way functions are named so as we can compute an image very easily but it will be difficult to find pre image for that image. All modern hash algorithm produces hash value of 128 bits and higher.

Generally a one way function has a main function at their core that does complicated transformations on the block of input bits. The function is almost bijective (collision is very rare) and the function should be iterated enough number of times to make the input unrecognizable. Some of the popular hash functions used today are MD5 and SHA1 and SHA256.

IV. THE PROPOSED ALGORITHM

In our proposed algorithm we will use the concepts of languages, linear transformation and one way hash function. Fig. 2 and Fig.3 are schematic depictions of the same.

The below given steps are performed on the Server side.

A. OTP Generation and Sending Phase

Step 1: Initially we select *n* languages $L_1, L_2, \ldots, L_n$ all defined over the alphabet say {a,b,…z,0, 1, …9, @, #, !, $, %}, each capable of generating strings of length $k_1, k_2, \ldots, k_n$ respectively.

Step 2: A common string (S) of length $k_1 + k_2 + \ldots + k_n$ is obtained by concatenating all the strings $(S_1, S_2, \ldots, S_n)$ generated by these languages. This string will be used for generating OTPs for users. This common string S, will be changed periodically by regenerating the strings $S_1, S_2, \ldots, S_n$.

Step 3: When a user requests for a transaction completion a six character OTP is randomly selected from S and is sent to the user.

Step 4: Meantime these six characters are converted to ASCII values A1, A2,…,A6 and will undergo a linear transformation to obtain B1,B2,…,B6.

Step 5: B1 to B6 are XORed and the result is hashed and stored(H).

B. OTP Verification Phase

Step 1: The received six characters of the OTP' are converted to ASCII values $A1', A2', \ldots, A6'$ and will undergo the same linear transformation to obtain $B1', B2', \ldots, B6'$.

Step 2: $B1'$ to $B6'$ are XORed and the result is hashed to get H' and compared with the stored hash H.

Step 3: If $H' = H$ the user is authenticated successfully to complete the transaction.

C. Refreshing The Master String

After using a master string S for a sufficient period of time, S will be regenerated from the defined languages. This period of time will be defined by the server. Similarly the component languages (L_1 to L_n) used to generate the string S also will be redefined periodically after every stipulated time. It is noted that the regeneration of S will be more frequent than the redefinition of the component languages. For example a master string will be valid for one hour whereas the component languages will be valid for one day.

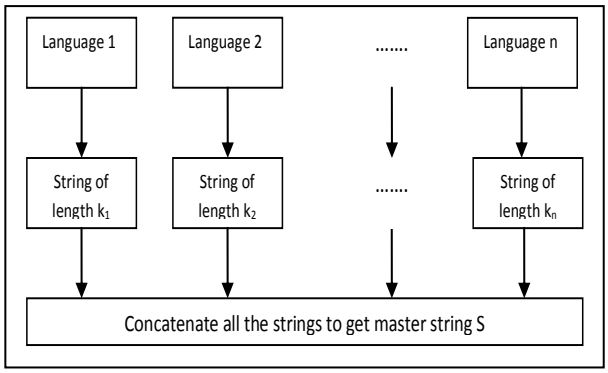

Fig. 2. Master Key Generation

V. Conclusion and Future Works

Our proposed algorithm mainly uses *n* independent languages for generating strings together with a linear transformation and a hashing function. The respective time complexities are discussed below [9]. As the languages are designed beforehand the complexity of a generating *n* strings is O(1). Computation of a linear transformation takes $O(n^2)$. One application of hashing function has the complexity O(1). Hence we can conclude the total complexity of this generation process is $O(n^2)$.

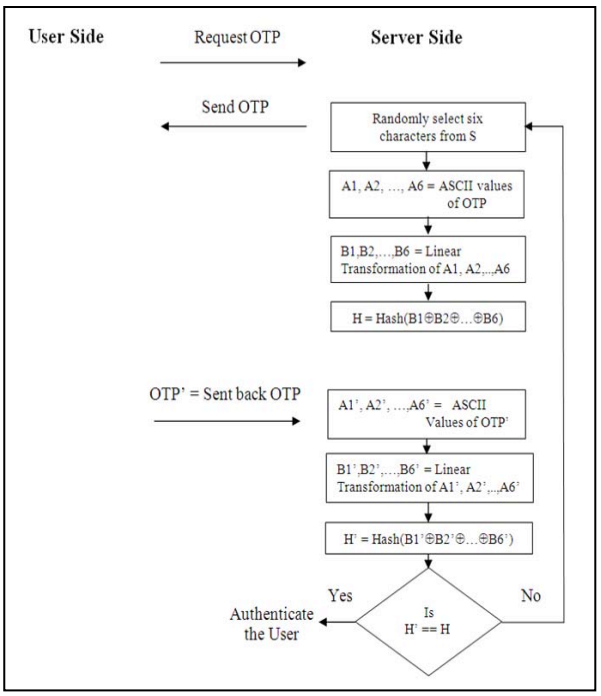

Fig. 3. OTP Sending and Verification Phase

In this paper, we have proposed a way of generating OTPs with alphanumeric characters. The possible ways of implementing it with practical circuits will be analyzed. Also variants of the proposed model with a concentration on improving the randomness will be undertaken.

References

[1] C.I. Fan, C.N. Wu et al., "Active One-Time Password Mechanism for User Authentication" , LNCS Vol. 7861, pp. 464–471, Springer, 2013.

[2] Y.W. Kao, G.H. Luo, H.-T. Lin et al., "Physical access control based on QR code", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 285–288, IEEE, 2011.

[3] W.Jang, S.Cho et al., "User-Oriented Pseudo Biometric Image Based One-Time Password Mechanism on Smart Phone", CCIS, Vol. 200, pp. 49–58, Springer, 2011.

[4] L. Lamport, "Password authentication with insecure communication," Communications of the ACM, Vol.24, No.11, pp.770-772,1981.

[5] M.H.Eldefrawy, M.K.Khan, K.Alghathbar, "One-Time Password System with Infinite Nested Hash Chains", CCIS, Vol. 122, pp.161–170, Springer, 2010.

[6] Peter Linz, "An Introduction to Formal Languages and Automata", Jones and Bartlett Publishers, Inc., 2011.

[7] Howard Anton and Chris Rorres, "Elementary Linear Algebra", Wiley, 2005.

[8] Alfred, J., Van Menezes Paul, C., Oorschot, S., Vanstone, A. "Handbook of Applied Cryptography" , CRC Press LCC, 1996.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, "Introduction to Algorithms", MIT Press, 2001