

Backup key generation model for one-time password security protocol

Jeyanthi N and Sourav Kundu

School of Information Technology and Engineering, VIT University, Vellore-632014, Tamil Nadu, India.

E-mail: njeyanthi@vit.ac.in

Abstract. The use of one-time password (OTP) has ushered new life into the existing authentication protocols used by the software industry. It introduced a second layer of security to the traditional username-password authentication, thus coining the term, two-factor authentication. One of the drawbacks of this protocol is the unreliability of the hardware token at the time of authentication. This paper proposes a simple backup key model that can be associated with the real world applications' user database, which would allow a user to circumvent the second authentication stage, in the event of unavailability of the hardware token.

1. Introduction

In the ever-increasing world of information technology, security is one of the principal concerns of every organization - government, military, corporate and academic institutions. In the traditional user authentication model, each user is given a username and a password, which is to be entered when prompted, in the authentication stage.

This model, however, is vulnerable to "replay attacks". In this form of attack, the intruder intercepts the communication channel and obtains the username and password. Later he reproduces it in the authentication stage of the product/service.

Relay attacks can be detected in several ways, e.g. using the low-cost RFID protocol, or the timing based protocol [4].

1.1 One-time Password (OTP)

Lamport [1] suggested a mechanism which prevents replay attacks. It involved generating a unique one-time password (OTP) during the authentication stage of a user. The password would be valid for only that particular login session. Thus, the login procedure was divided into two stages:

1. In the first stage, the user would enter the username and password.
2. In the second stage, the user must enter the one-time password delivered via a hardware token.

For every login session of a user, a different OTP would be sent. This procedure is also called *two-factor authentication*.



1.2 OTP Generation

A seed value was planted in a hash function, and the hash function was run N times. After each successful authentication event, the number of secure hash function trials (N), is reduced by one. Thus, in the first trial, the hash function runs N times, in the second trial, $N-1$ times, and so on. This produces a unique OTP for every user in the authentication stage [2]. The output of the hash function is a 64-bit OTP [2] and is encoded in a form, which is easy to manually “copy and paste” from one device (hardware token) to another (authentication interface).

1.3 Application of OTP in the Consumer Industry

The sophisticated increase in network attacks signalled the need for better security, which was in turn, easy to operate by the end-user. OTPs were slowly rolled out to consumers. One of the most noticeable applications of OTP started with Google when it urged its users to enable two-factor authentication in their free email service – Gmail. Later Microsoft, Dropbox and other tech giants joined the bandwagon.

2. Two-Factor Authentication

The process involves two stages or “layers” of security. The first stage is the traditional username-password pair. Once the user is able to successfully validate the credentials, an OTP is generated by the server and sent to the hardware token of the user. The hardware token is the user’s registered cellular phone number. Once the user receives the OTP, he has to manually copy-paste the password from the hardware token to the authentication interface.

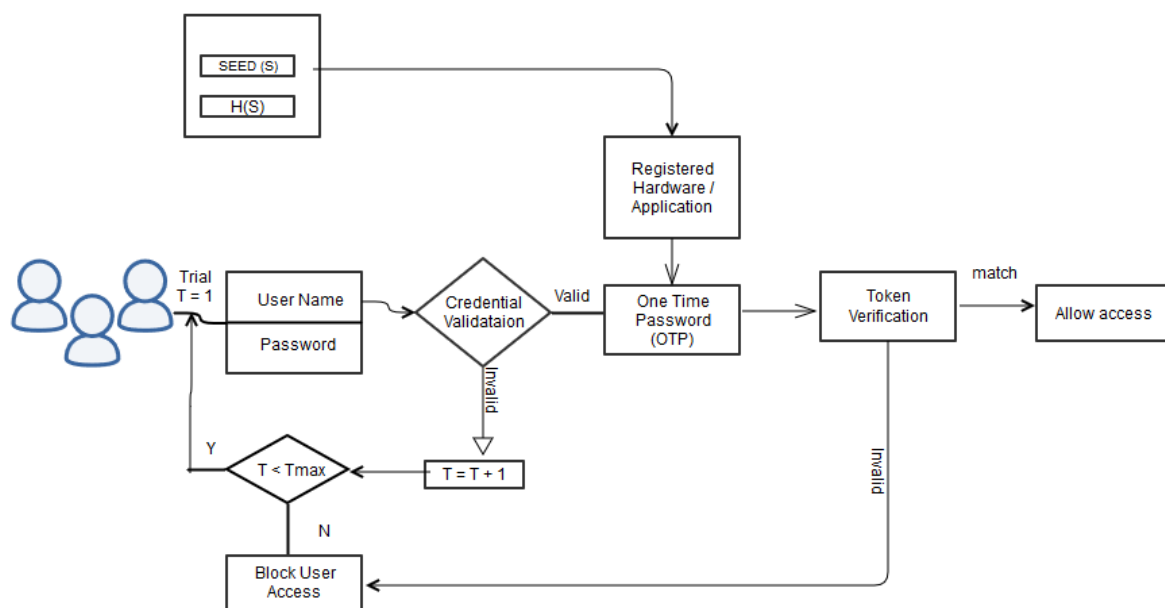


Figure.1. Overall Architecture of Backup Key Generation

This method prevented relay attacks – even if the intruder obtains the username and password, he still will not have access to the OTP, and thus will be barred at the second authentication stage.

2.1 Limitations of Two-Factor Authentication

One of the primary limitations of this authentication model is the hardware token – specifically the cellular phone.

If this device were to be unavailable or unresponsive at the time of authentication, then the second stage would not be passed, and the login attempt would fail. There are several other factors that would affect the hardware token such as:

- *Working cellular network plan:* In order for the one-time password to be delivered via the Short Messaging Service (SMS), the cellular phone must have network connectivity. This method might fail to authenticate a legitimate user, based on his physical location.
- *Availability of hardware token:* Electronic devices are not 100% reliable. If the device is lost, stolen, damaged or simply unresponsive, then it would impair the second stage of the authentication process.

This paper suggests a simple prototype model which would allow a user to circumvent the second authentication stage by using a random, truly unique backup key.

3. Working Principle of Backup Key Model

The backup key is a 16 digit alphanumeric code with both uppercase and lowercase symbols. Each entry in the user database, i.e. every username-password pair, is associated with a unique backup key. That key would not be associated with any other pair within the user database.

At the time of a user's registration, the backup key would be supplied to him and associated with his account. It is assumed that the user has kept it safe. In the event when he is barred at the second authentication stage (due to unavailability of the hardware token), he could enter this backup key and regain access to his account.

Immediately, a new backup key would be generated and associated with his account. The user will also be notified of the new backup key, through an appropriate communication medium, e.g. via email or SMS.

4. Implementation

The key requirements to implement the backup key model are as follows:

- A relational database management system (RDBMS) to store the backup key associated with each user-password pair.
- A platform independent programming language.

In our implementation, we have used the open-source MySQL database and Java as the programming language.

Following are the three entities that we're going to implement in this paper:

1. A *database schema* that holds the backup keys. This database schema has been designed keeping the real world applications in mind and can be directly modified or adapted, to fit the needs of particular application.
2. The *backup key generation algorithm*, which generates a unique backup key.
3. The *event triggered algorithm*, which describes the steps to follow when the user has used the currently assigned backup key and a new key is to be assigned to his account.

4.1 Database Schema

We consider a real world example where the software uses a database to store the user account information. Let us consider a MySQL database named SAMPLEDB. The schema is defined as follows.

Table 1: ACC_MAIN: This table holds the list of the accounts associated with the software. It contains 3 (or more) columns depending on the requirements of the application.

1. **ACC_ID:** This is the primary key. It is used as a unique identifier since two usernames might be similar.

2. USERNAME: This field may or may not be unique depending on the needs of the software.
3. PASSWORD: Respective password for that username.

Table 1: Database Schema for Acc_Main Database

ACC_MAIN		
ACC_ID	USERNAME	PASSWORD
13871	Arcot	****
13872	brown	****
13873	cathrine	****
13874	devrat	****
13875	Evans	****

Table 2: BACKUP_KEY: This table holds the backup keys associated with each account. It references the primary key of ACC_MAIN table, (ACC_ID) as the foreign key. Its columns are:

1. BKPKEY: This is the 16 digit alphanumeric backup key.
2. ACC_NO: This is the foreign key that references ACC_ID from ACC_MAIN.

Table 2: Database Schema for Backup_Key Database

BACKUP_KEY	
ACC_NO	BKPKEY
13871	zRCPuiXIwgbs57bU
13872	kXlrDfyo2bCrbLmn
13873	ZJdMKynJTZsyoOI6
13874	gEyPy99ajgqqr1QS
13875	Tfpkn4DtNh0WcqZl

Table 3: DISCARDED_KEY: This table holds those backup keys that have been already used in application. The only way a key is sent to this table is when a user uses the backup key model to gain access in the second authentication stage. Thus, the number of entries in this table is the exact number of times the backup key model was used in the application. This table contains only one column – DIS_BKPKEY.

Table 3: Database Schema for Discarded_Key Database

DISCARDED_KEY
G5Ub2LMy8n8UDmcR
s08lahQoc2Le3l9j
KOTOk9rQs0ZlhN6a
kPH6H0tPILPC8VXA
SQi6F4bCsDg4Uquv

4.2 Algorithm1: Backup Key Generation

This algorithm returns a unique backup key K.

- A 16 digit backup key is generated using the random() function of the programming language.
- The key is then checked with the DISCARDED_KEY table
 - If a match is found, then the key is discarded and another key is generated
 - This step is repeated until one such key is found which did not exist earlier
- The key is then returned

The probability of two, 16 digit, randomly generated alphanumeric keys, being the same, is extremely low and can be avoided when calculating the complexity. This check *guarantees the uniqueness* of the generated backup key. For more information on the uniqueness criteria. Fig. 2 outlines the flowchart for this algorithm.

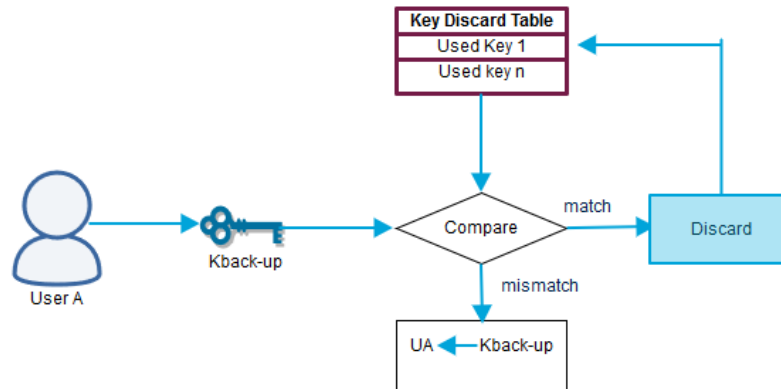


Figure 2:Backup Key Generation Algorithm

ALGORITHM: generate_backup_key()

INPUT: None

OUTPUT: Key K

Generate backup key K using rand() function

CHECK (K exists in DISCARDED_KEY table)?

IF TRUE go to STEP 2

If FALSE go to STEP 4

RETURN K

4.3 Algorithm 2: Event Triggered Algorithm

In the event that a user wants to use the backup key in order to gain access in the second authentication stage, the current backup key (that is associated with his account), is matched with the user-supplied key. If they match, then the event_triggered () function is called. The flowchart for this algorithm can be found in Fig. 2.

- The current backup key C is moved to the DISCARDED_KEY table
- A new key K is produced from the generate_backup_key() function
- K is then associated with the user's account

ALGORITHM: event_occured()

INPUT: User account U (who trigger the event)

OUTPUT: Associate new backup key to U

Key C = current key associated with user U

Add key C to table DISCARDED_KEY

Key K = generate_backup_key ()

Associate key K with user U

4.4 Backup Key Uniqueness Criteria

There are two scenarios in which the uniqueness criteria of the backup key must be fulfilled.

When the backup key is associated for the first time to a new user, i.e. at the time of registration

When a user has used the currently assigned backup key and a new one must be added

In the first scenario, Algorithm 1 is used. The username and account entry is made in the database and the corresponding backup key is generated and assigned.

In the second scenario, Algorithm 2 is used. First, the old backup key is moved to the DISCARDED_KEY table. A new key is then generated using Algorithm 1 and assigned to the user's account.

It can be inferred from Algorithm 1 that the backup key generated is always unique. This guarantees the uniqueness criteria of the prototype. At the same time, it increases the time complexity of the backup key generation algorithm to $O(n \log n)$.

The number of backup keys available is infinitesimally large. It is in the order of $62P16 = 5.71 \times 10^{27}$, which makes it suitable for large scale applications.

5. Results

Java and MySQL were used to implement the algorithms in a test environment. It was found that the algorithms had produced the expected output. Results for each of the algorithms have been depicted in Fig. 3 and Fig. 4.



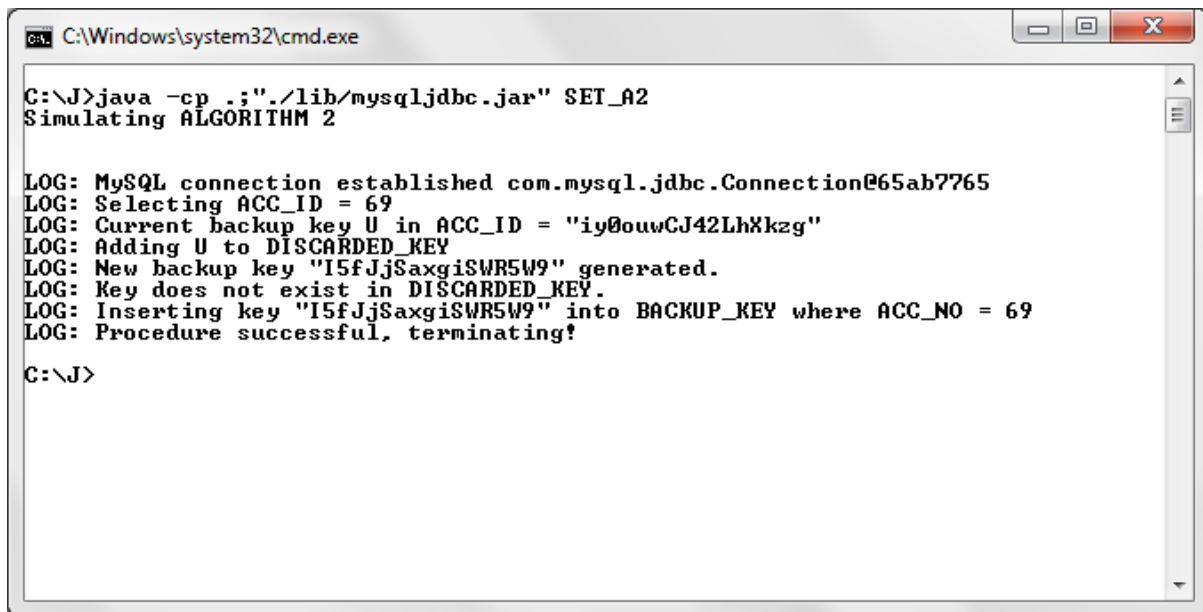
```
C:\Windows\system32\cmd.exe

C:\>java -cp .;\"./lib/mysqljdbc.jar\" SET_A1
Simulating ALGORITHM 1

LOG: MySQL connection established com.mysql.jdbc.Connection@65ab7765
LOG: Backup key "iy0ouwCJ42LhXkzg" generated.
LOG: Key does not exist in DISCARDED_KEY.
LOG: Inserting key "iy0ouwCJ42LhXkzg" into BACKUP_KEY where ACC_NO = 69
LOG: Procedure successful, terminating!

C:\>_
```

Figure 3: Program Output for Algorithm 1



```

C:\Windows\system32\cmd.exe

C:\J>java -cp .;\"./lib/mysqljdbc.jar\" SET_A2
Simulating ALGORITHM 2

LOG: MySQL connection established com.mysql.jdbc.Connection@65ab7765
LOG: Selecting ACC_ID = 69
LOG: Current backup key U in ACC_ID = "iy0ouwCJ42LhXkzg"
LOG: Adding U to DISCARDED_KEY
LOG: New backup key "I5fJjSaxgiSWR5W9" generated.
LOG: Key does not exist in DISCARDED_KEY.
LOG: Inserting key "I5fJjSaxgiSWR5W9" into BACKUP_KEY where ACC_NO = 69
LOG: Procedure successful, terminating!

C:\J>

```

Figure 4: Program Output for Algorithm 2

Real World Example

Consider an employee database of two distinct companies – A and B. It is possible to have a common backup key, for two (distinct or similar) username-password entries, in the two companies. However, the probability of that event would be exceedingly low. Yet it is possible because companies A and B are distinct, and hence are treated as different authentication models. However, at the same time, it is impossible to have the same backup key within company A, as it would violate the uniqueness criteria.

6. Conclusion

Using the backup key generation model, the second stage of the two-factor authentication model can be circumvented. However, this technique assumes that the user has kept the backup key in a safe and readily available location. The time complexity of both the algorithms is $O(n \log n) + c$, where c is a constant. The logarithmic complexity occurs since the algorithm searches the DISCARDED_KEY database for a possible match of the newly generated backup key.

As time goes by, more entries are made to the DISCARDED_KEY database. As a result, searching takes slightly longer in the larger database and there is a slow increase in the time complexity of both the algorithms.

References

- [1] Leslie Lamport 1981 Password Authentication with Insecure Communication *Communications of the ACM* **24**(11) 770-772
- [2] Haller N, Metz C, Nesser P and Straw M A One-Time Password System RFC 2289 3
- [3] Jorge Munilla and Alberto Peinado 2010 Enhanced low-cost RFID protocol to detect relay attacks *Wireless Communications & Mobile Computing* **10**(3) 361-371
- [4] Jason Reid, Juan M. Gonzalez Nieto, Tee Tang and Bouchra Senadji 2007 Detecting Relay Attacks with Timing-Based Protocols *ACM Symposium on Information, Computer and Communications Security Singapore*

- [5] Jeyanthi N, Thandeeswaran R and Vinithra J 2014 RQA Based Approach to Detect and Prevent DDoS Attacks in VoIP Networks *Cybernetics and Information Technologies* **14(1)** 11-24
- [6] Mahantesh Gawannavar, Payal Mandulkar, Thandeeswaran R and Jeyanthi N 2015 Office in cloud: Approach to Authentication and Authorization *Recent Advances in Communications and Networking Technology Bentham sciences* **4(1)** 49-55
- [7] Vijayan R and Jeyanthi N 2017 Enhancing trust in mobile Adhoc network using trust and Dynamic with energy efficient multipath routing protocol *International Journal of Signal and Imaging Systems Engineering Inderscience Publishers Ltd* **10(1-2)** 31-38