Mobile Netw Appl (2011) 16:304–336
DOI 10.1007/s11036-011-0303-9

Security Attack Safe Mobile and Cloud-based One-time Password Tokens Using Rubbing Encryption Algorithm

Fred Cheng

Published online: 12 April 2011

Abstract One-time Password (OTP) Token has become one of the main stream security products during the past few years. OTP Token can automatically generate a random password. It is especially popular to be used with the Two-factor Authentication (2FA) system. OTP Token has proliferated into many different form factors such as standalone token, PC, PDA, cellular phone and Cloud-based token. But most of the implementation has their short comings with high token cost, not easy to carry and high supporting or deployment cost. Certain implementations may also compromise the network security when the token is lost or stolen. Moreover, most of the tokens can be broken-in by Man-in-the-Middle Seed-tracing and Shoulder-surfing security attacks. To overcome such aforementioned issues, we propose a secure encryption algorithm – Rubbing Encryption Algorithm (REAL). We use REAL to implement a Mobile-based and a Cloud-based OTP Token as design examples. Both of them are of high security level, lower total token cost and can resist the aforementioned security attacks as well.

Keywords authentication · security · OTP token · one-time password · OATH · security attack

1 Introduction

How to securely send and receive secret and confidential information to and from a remote site has been an old problem for thousands of year. In the ancient time, the emperor would give his General an official seal and an encryption/decryption pad before the army marched away to the remote front. The secrecies would be encrypted with this pre-assigned encryption PAD[1]. The encrypted message or document would be wax-sealed by the official seal. It was then carried by the brave soldier riding on a horse and speeding toward the destination. There might be hostile enemies lingering along the way. They would try to intercept the carrier to discover the secrecies of the message.

In the high tech twenty first century, it is interesting to find that a similar scenario is still in play. We use cipher systems to encrypt secrecies before sending it to a remote site. The encrypted message is digitized and sealed in a packet format. It is sent through an invisible and pervasive electronic network (the Cloud and or other special network). However, there are many malicious people (enemies) lingering in the middle of this network. They are trying to snatch away the secret message for illegal financial gain or other harmful acts. It is not a surprise for us to frequently hear news about network break-in and financial loss. Indeed, the history is replaying itself but in a new format.

1.1 Motivation of the research

The Cloud has become a popular business transaction platform nowadays. Unfortunately, this powerful and pervasive network somehow is overshadowed by the growing security threat emerging from the various attacks. The Two-factor Authentication (2FA) technology, combin-

F. Cheng (✉)
International Technological University & FPC Consultancy,
Los Altos Hills, CA 94022, USA
e-mail: fredtcheng@yahoo.com

[1] One famous example is Caesar's Cipher. It was named after Rome emperor Julius Caesar. He used this cipher to securely communicate with his generals more than 2000 years ago.

ing a One-time Password (OTP), has emerged as a popular protection system. The 2FA system employs two user specific factors for authentication. It can significantly enhance the network security. As a result, the US government strongly recommended its financial industry to conform to such authentication system by end of 2006 [43].

Many solutions were proposed to implement the OTP function [2, 34, 45]. Unfortunately, they lead to no interoperable capability among different tokens. It also results in poor or no compliant with various 2FA systems. And the cost of a token is relatively high. Moreover, many OTP tokens compromise the security when they are lost or stolen. Eventually, these factors results in the slow adoption of the OTP token and 2FA system in the market [49].

In responding to the slow OTP market adoption, a group of companies formed OATH – Initiative for Open AuTHentication [17]. Its goal is to develop an open One-time Password (OTP) reference architecture. They use open standards with free license to accelerate the adoption of strong authentication such as the 2FA system with a fixed standard OTP Token.

OATH addresses the standardization, compliance and interoperability issues associated with OTP algorithm. But it does not address the issues on implementation method, cost, system deployment plus maintenance, secret key protection and user experiences of the OTP token. These are other key subjects to be addressed when implementing an OTP token. They are also the important parts affecting the market adoption on the OTP Token. Our motivation is to find some viable solutions to fill the gap where OATH leaves out.

1.2 Proposed solutions

To address the aforementioned issues, we propose a new cipher system – Rubbing Encryption Algorithm (REAL) in this paper. REAL is an improved secure scramble cipher with a complex encryption key. A user does not need to memorize or enter the key when decrypts a REAL ciphertext. The encryption key is embedded on the hardware token. By laying the hardware token directly over the REAL ciphertext image on a device's screen, a user can electronically "rub" (decrypt) the OTP code. This is why this new cipher system gets its name – "Rubbing Encryption ALgorithm" (REAL). Having a feature of decrypting without entering the key, it allows REAL to use a long and complex key for encryption. So REAL can securely encrypt a short word length OTP code with high security.

When REAL encrypting the OTP code (plaintext), it generates a REAL Matrix (REAL Image). REAL Matrix is an equiprobable matrix. Equiprobable matrix has the highest Shannon entropy (uncertainty) among the same size of matrices. This feature ensures the desired security protection of the ciphertext. The OTP code is scrambled (hided) inside the REAL Matrix according to the REAL encryption key. REAL key is the locations where OTP code symbols hided inside the REAL Image. The key can be in the forms of a code pointer or transparent window on the hardware token surface.

REAL does not use complicate mathematic operation. It does not require high powerful computing platform. So REAL is suitable to operate on a devices with less powerful CPU, smaller memory and limited battery power capacity.

Two implementation examples are detailed in this paper. One is a Cloud-based OTP Token [4]. The other is a Mobile OTP Token [5]. Both tokens are complaint to OATH standard. REAL Cloud-based OTP token obtains OTP code from any PC that can access the OTP sever through Internet connection. It is good for a secure login authentication even when using a public PC or Internet kiosk. REAL Mobile OTP token is a standalone token. It works with a designated cellular phone. A user uses it just like a regular standalone token to generate an OTP code for 2FA authentication need.

Implementation and operating procedure of a REAL Cloud-based OTP Token is as follows. Time-based OATH OTP algorithm (TOTP) is used for OTP generation. We first design an image display form factor - also called REAL Image (RI). The OTP code generated by an OATH TOTP server is hided inside the REAL Image using REAL. To further protect the REAL Image, the RI is then embedded in a much larger size of Ciphertext Image (CI). The elements of this CI and RI are made of the same symbol set as TOTP code. Both RI and CI are also made to be the equiprobable matrices. So the OTP server can send the ciphertext image securely throughout Internet without worrying the security of OTP code. To obtain the OTP code, a user initiates the OTP generation through OTP server. Once receiving the Cipheretxt Image, the user can simply place her REAL hardware token on the PC to decrypt the OTP code.

REAL Mobile OTP token is a standalone token. Mobile OTP token uses OATH Event-based OTP algorithm (HOTP). Once a user finishes her registration, the OTP server generates a series of OTP codes. These OTP codes are REAL encrypted into REAL Images (RI). An Offset value is generated from the Hashed Index (HI). RI is logically bitwise exclusive-OR'ed with the Offset value to generate the Delta. Series of codes become a series of Delta values. Sequential order of Delta values are then rearranged according to the value of the HI associated with each delta value. This re-arranged Delta values form the Delta Table (DT). Both DT and HI are further encrypted by a key generated from user's credential data. The encrypted DT and HI are then stored in the cell phone. Since we only

store the difference data of the ciphertext, it protects the code and OTP key. To use the token, a user activates the token with her credential data. The phone generates RI and displays it on the screen. The user places her hardware token on the display to decrypt the OTP code.

One REAL hardware token can carry two or more embedded keys. It means one RAEL hardware token can operate as two or more OTP tokens. By mixing the OTP codes from different seeds, REAL OTP token can send out codes that do not conform to any pseudo random sequence. A hacker cannot reverse the OTP seed by intercepting such REAL encrypted OTP codes. So we can have the "Security Attack Safe" REAL OTP tokens.

We organize this paper as follows. The background and related work is summarized in Section 2 and followed by the introduction of Rubbing Encryption Algorithm (REAL) in Section 3. In Section 4, we implement and analyze the security of an OATH compliant REAL Cloud-based OTP Token. We implement and analyze the Mobile OTP Token in Section 5. We further use REAL to implement and analyze two OTP tokens that can resist MITM Seed-tracing and Shoulder-surfing attacks in Section 6. A discussion on 2FA, OTP Token and REAL's limitations is illustrated in Section 7. We then conclude this paper and future improvement work in Section 8.

2 Background and related works

2.1 Password and network security

Remote user authentication has been in use for greater than 30 some years. In the early day, a userID and password were the only requirement to effectively fend off any illegal access. As network break-in happens from time to time, password has evolved from a simple scheme to a complex form to enhance the network security protection.

Table 1 lists various types of password and its strength. But even with a complex and good strength password, the network is still broken-in. The problem is that a user's password usually stays unchanged (static password) for a period of time. So upon intercepting the password, an intruder can impersonate the user to login when the user signs off the network. This is the so called Man-in-the-Middle (MITM) Replay attack. So a complex static password alone cannot fully protect the network security.

2.2 Lamport's dynamic password

Leslie Lamport in his landmark 1981 ACM paper [21] proposed a new password system to counter the MITM Replay attack. Leslie used a one way hash function *(F)* to generate a series of dynamic passwords. The non-reversible hash feature ensures the desired protection on the seed (x). Since only the user has the correct seed (x) to generate the distinct password *F(x),* a user can surely prove her by such password and be securely authenticated by the server.

Lamport's dynamic n passwords are shown as follows.

$$\text{User}: F^{n-1}(x), F^{n-2}(x), \ldots, F^{n-i}(x), \ldots, F(F(x)), F(x), x. \quad (1)$$

Before the authentication process, the user chooses an initial seed (x) and computes a series of password as shown in (1). To start the authentication process, the user delivers the $F^n(x)$ through a secure method to server. During the login process, the user then sends the password $F^{n-1}(x)$ to server. Server hashes the password, $F(F^{n-1}(x))$, and compare it to the prior-stored password $F^n(x)$. If both values are the same, the user is authenticated and allowed to access the network.

For future login, the user will send the password $F^{n-i}(x)$ to server. Server will authenticate the user by hashing the received password, $F(F^{n-i}(x))$, and compare the result with the last password $F^{n-i+1}(x)$ from the last login session. Using such scheme, server will only have to store the last password for next authentication need.

Since F is an ideal hash function, each $F^i(x)$ will be of different value in the series. It guarantees the freshness of the password for each login session. Each password is only good for one session and has no direct visible relationship with next password. The password intercepted by MITM attack is no longer useful for next login. The intruder cannot reuse the intercepted password. Lamport's dynamic password successfully prevents the MITM Replay attack.

Table 1 Password and its security strength

Password Type	Example	Strength	Resist Brute Force or Dictionary attacks
Simple and basic	1234, dog	Poor	No
Long and random number	910482563957	Poor	No
Long phrase	ilikemyschool	Better	May be
Case sensitive	Steve, F1CarRace	Better	May be
Alpha-numeric	a12f5g, Jeff1234	Higher	Yes
Alpha-numeric with symbol	joe@itu168$	Higher	Yes

2.3 Two-factor Authentication (2FA)

Lamport's dynamic password concept facilitated the birth of many new authentication systems. In order to protect the security of remote network access, a new authentication protocol was developed besides the dynamic password. It is the so called Two-factor Authentication (2FA) protocol. It employs two different factors that are chosen from any two of the following three groups [43].

- What you know – such as password, PIN, phrase, …
- What you have – such as Token, Smart Card, ATM Card, Certificate, …
- What you are – such as retina, finger print, voice, face, …

2FA's operation is as follows. A user sends her UserID and password (what you know) to the network authentication sever for authentication. The server verifies the user by her UserID and password in server's database. If it is matched, the server then asks the user to send the correct prior registered second factor for final authentication. Once being fully authenticated, the user is allowed to access the network. Figure 1 shows the 2FA system.

Such secret second factor can be any one factor from "what you have" or "what you are". Only the user has her pre-registered secret second factor. An intruder cannot have or obtain this secret easily. So a 2FA system can accurately authenticate a user. It provides a better network security than the basic UserID and password system. This is one of the key reasons that the Federal Financial Institutions Examination Council of the US government endorsed and recommended 2FA as the key authentication technology for its financial industry [43]. In fact, the US government asked the industry to implement such system by end of year 2006.

2.4 OTP and OTP token

Among all the second factors, token's particular capability of automatically generating a dynamic session password has helped it to gain a dominant position in the 2FA system. Each password is only used once for each login session. This dynamic session password is called One-time Password (OTP). Many solutions were proposed to implement such OTP Token [2, 34, 44]. But they do not gain the wide market adoption. It is mainly due to several deficiencies such as poor token interoperability, poor inter-system compliance, high hardware token cost, poor portability, and high deployment and support cost. In particular, many implementations compromise the security when the token is lost, stolen or secretly copied [46].

OTP is generated from two secrets – a seed (K) and a value (C). The OTP generating algorithm has two types. For event-based system, the value (C) is the sequential event count number. For time-based system, the value (C) is the Internet time data. Such Internet time data is usually synchronized to a highly accurate and reliable time source such as National Institute of Standards and Technology (NIST) Internet Time Service NTP servers [30].

The shared secret seed (K) can be of two types – symmetric and asymmetric. Symmetric secret algorithm shares and stores the same key (seed) at OTP token (client) and authentication server. Since token contains both secret key and counter values, security can be compromised if the token is lost or stolen. Asymmetric secret algorithm uses PKI (Public Key Infrastructure) Digital Certificate. Such OTP token does not store encryption key. It has the public digital certificate instead. So token's loss or stolen will not result in compromising the security. But implementing a PKI requires a complicated and high cost system. It limits the mass adoption from the market.

OTP token's form factor can be of two types – either hardware or software form. Figure 2 shows different variety of OTP Tokens available in the market. Hardware token has the portability advantage and can be independently used anywhere. But it cannot be provisioned easily through Internet for immediate activation. So the deployment and support are expensive and difficult. It bears with high total maintenance cost as well.

Software token, on the other hand, can be easily provisioned through the Internet. Its deployment and service is faster and easier than the hardware token. But the key and counter secrets are stored on a designated PC or mobile device. They can be secretly copied by a malicious person. It may compromise network security. Such drawback offsets the other advantages over the hardware token. Software OTP token thusly does not gain as much popularity as hardware token.

Traditional OTP token works with only one specific authentication server. For different 2FA servers, the user has to carry as many different tokens. It is inconvenient and also costs the user extra money.

One frequently seen problem associated with OTP token is the token-server de-synchronization issue [23]. For event-

Fig. 1 Two-factor authentication operation

Fig. 2 Various form of One-time Password tokens

based token, it happens when the token is accidentally activated without performing a login process. An unsuccessful login process may also trigger a counter value increment on the token while server still has the previous value. For time-based token, it usually has a high precision crystal clock. But the aging and temperature variation on electronic circuit and its components will cause the time source to drift faster or slower. It then results in the de-synchronization issue.

And when de-synchronization happens, the OTP code generated by the token is different from what the server thinks it should be. Authentication process cannot be complete then. Server usually sets a de-synchronization deviation threshold (e.g. within 2 or 3 events from the last login for event-based token) to allow a user to complete the authentication. Such threshold setting also prevents the malicious people from keeping on attacking the server with many different OTP codes.

2.5 OATH and OATH OTP token

Seeing the slow market adoption of the OTP and 2FA, a group of companies formed the Initiative for Open AuTHentication [17] in the early decade of year 2000. This initiative under the collaboration of industry is usually called "OATH". OATH's mission is to develop an open reference architecture by leveraging the existing open security standards for the universal adoption of the strong authentication system.

OATH has published its OTP generation algorithms. Event-based OTP (HOTP) is presented in the RFC4226 [27] on December 2005. Time-based OTP (TOTP) was proposed in an Internet Engineering Task Force memo [28] dated September 8, 2010. Another challenge-response authentication method (OCRA) was also proposed in an Internet Engineering Task Force memo [29] dated September 9, 2010.

HOTP uses Keyed-Hash for Message Authentication (HMAC) method [16] to generate the OTP code. The chosen hash algorithm is SHA-1 [7]. SHA-1 [15] was published by National Institute of Standards and Technology (NIST) in year 1995 as the official federal information processing standard.

HOTP uses a 64 bit counter (C) with a 160 bit key (also called Seed, K). SHA-1 hashes the two secrets to generate a 160 bit digest. HOTP then uses a dynamic truncation program to reduce the bit length to the range of 48 bit to 64 bit code or a 6 to 8 digit OTP code. The SHA-1 hash function guarantees the unique and irreversible of the digest. The truncation procedure further produces a random and irreversible OTP code. So we can have a unique and secure OTP code from the HOTP generation.

Figure 3 shows HOTP code generation logic block diagram. It can be further represented in (2).

$$\mathrm{HOTP} = \mathrm{Truncate}(\mathrm{HMAC} - \mathrm{SHA} - 1(\mathrm{K}, \mathrm{C})). \tag{2}$$

2.6 MITM Seed-tracing and Shoulder-surfing security attack

Given a fixed seed (K) and algorithm, the generated OTP codes will always be the same no matter where or when we run the code generation program. In other word, the generated codes will show a fixed pseudo random sequence pattern. An intruder may have the chance to trace out the seed (K) if he captures enough series of OTP codes from the same token. This is the so called Man-in-the-Middle Seed-tracing Attack (MITM Seed-tracing attack).

Shoulder-surfing attack in network security refers to secretly observe a user while she is using the OTP token or entering the password during the login session. An intruder can obtain the user's OTP code or other confidential information through such secret attack without the user's

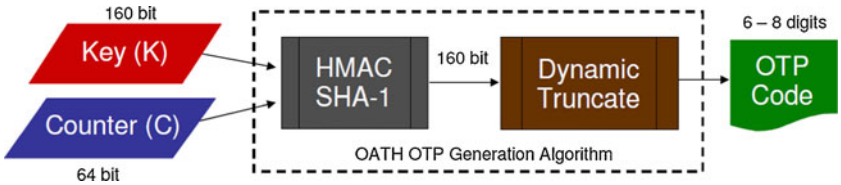


Fig. 3 OATH HOTP generation block diagram

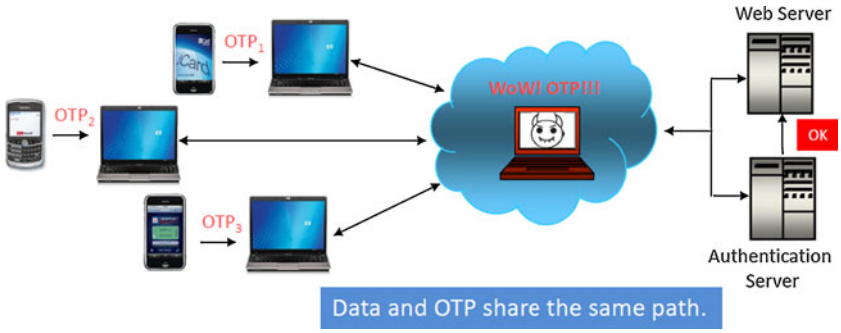


Fig. 4 Standalone mobile OTP tokens

awareness. The intruder can then pursue Seed-tracing if he obtains enough series of OTP codes as we discussed in the previous paragraph.

2.7 Related work on mobile OTP tokens

2.7.1 Standalone mobile OTP token

Several implementations have emerged as the key methods to use a cellular phone in remote network authentication. One such solution focuses on using cellular phone as a standalone OTP token [6, 33, 45]. The phone is a computational platform to generate the OTP code. The OTP generating software, user's secret seed (K) and counter value (C) are stored in the cellular phone. In operating the token, the user activates the software. The phone generates the OTP code. The user reads the OTP code and enters it into a PC or Internet device for 2FA need. Once authenticated, the user is allowed to access the network. Figure 4 shows the full operation of such system.

Standalone mobile OTP token has the same pros and cons as the traditional token. It can be used anywhere without the geographical limitation or dependence on communication protocol and equipment. But these simple mobile tokens usually do not have any capability to resist the OTP seed (K) tracing by MITM interception or Shoulder-surfing attacks. Moreover, it stores the secret seed (K) and counter value (C); these secrecies can be exposed if the phone is lost or stolen. The network security is comprised then.

2.7.2 Out-of-band transceiver mobile OTP token

An alternative proposal focuses on using the cellular network as a secure out-of-band channel to transmit or receive the OTP code to and/or from the authentication server [3, 22, 26]. Figure 5 shows such a system. The OTP code is generated by an OTP server. The OTP code is transmitted as an image data or text message via cellular network's Short Message Service (SMS) to the user's cell phone. The user then enters such OTP code to the authentication server to complete the 2FA procedure. Once authenticated, the user is allowed to access the network. Theoretically, the OTP can also be sent to remote user on her PC. The user then sends the OP code to the authentication server through the cellular network.

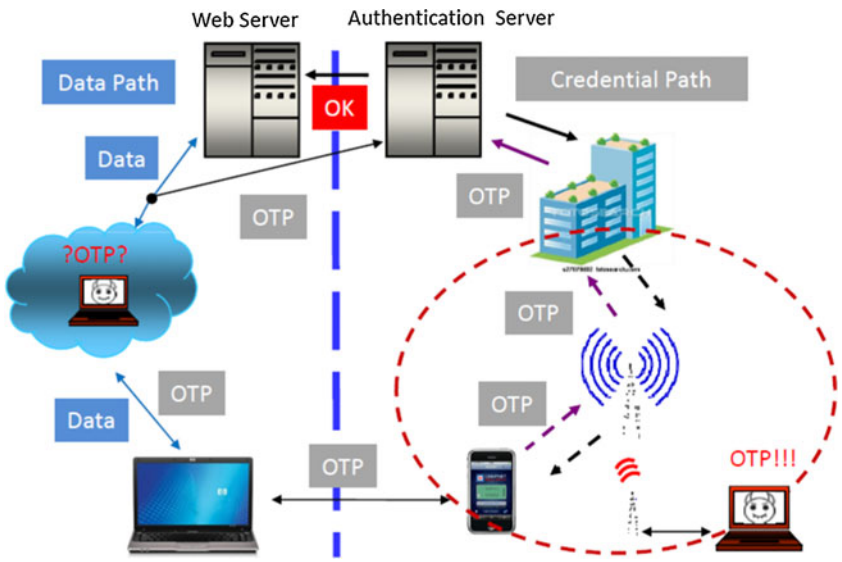


Fig. 5 Mobile OTP token using Out-of-the-band for OTP code

The OTP code will not goes back to the server through the original transmitting path. It provides the desired security protection. Since the OTP is generated centrally, the server can use a highly randomized algorithm. It can further reduce the MITM Seed-tracing attack risk.

An improved version [42] uses the mobile phone as the OTP receiver. The OTP code will be sent back to sever through mobile phone as well. So the OTP path and data path are completely separated. The intruder can no longer obtain the OTP code in the Cloud. It completely prevents the traditional MITM attack in the network. But the Shoulder-surfing attack is still an issue.

There are limitations of such mobile token. Cellular's QoS (quality of service) will affect the reliability of OTP generation when using cellular SMS. SMS is a best effort delivery service. The cellular service providers cannot guarantee a real-time or in-time delivery. Moreover, when a user is out of the cellular service coverage area, such as in a basement of the building or in the rural area, using SMS sometimes is not even possible. These factors limit the use of such OTP token.

Besides, new software and hardware are needed to allow the authentication server to interface to SMS system. It increases cost on system set up besides the service and maintenance. An intruder can also use a special cellular receiver to intercept the OTP code if the server or user's cellular numbers are known. So such out-of-band channel is no longer as secure as it originally plans for.

2.7.3 Mobile authenticator

The most recent proposal [1, 13, 14, 20, 40, 41, 45] can be called a Mobile Authenticator. It involves using the Subscriber Identity Module (SIM) and other newly proposed communication protocols such as the Liberty Alliance Federation Standard [24] or The Free Auth Project [11]. Figure 6 shows such scheme. The mobile device contains the user's credential information in its SIM card. The authentication is carried out directly through the phone, cellular network and the cellular service provider's server. Following the protocol procedure, once authenticated, the cellular service provider notifies the web server to allow the user's access.

Two separate paths are used in this scheme. The user's credential data is only sent through the cellular network. The path separation prevents the MITM attack in the network. But the effectiveness of such scheme is limited by cellular service coverage as we discussed in the Subsection of 2.7.2. Moreover, using different authentication protocols and OTP algorithms usually leads to a totally new authentication system. The authentication is done by a cellular provider. Cellular provider needs to show its capability and reliability in providing such an important service. It also needs to earn the trust from the user. The service charge and business model are yet to be defined and agreed among the cellular providers and each individual company involved. New software and hardware are required at server to implement such scheme. This proposed solution may not be fully compliant with the existing authentication servers either. And the Shoulder-surfing attack is still an issue.

2.7.4 Short summary on mobile OTP token

Table 2 summarizes the pros and cons of the various solutions of the mobile OTP tokens. It is interesting to find that a standalone cellular phone-based Mobile OTP Token outshines the other two newer approaches on performance versus cost ratio. The key issue is that we need to resolve the security issues associated with this standalone mobile

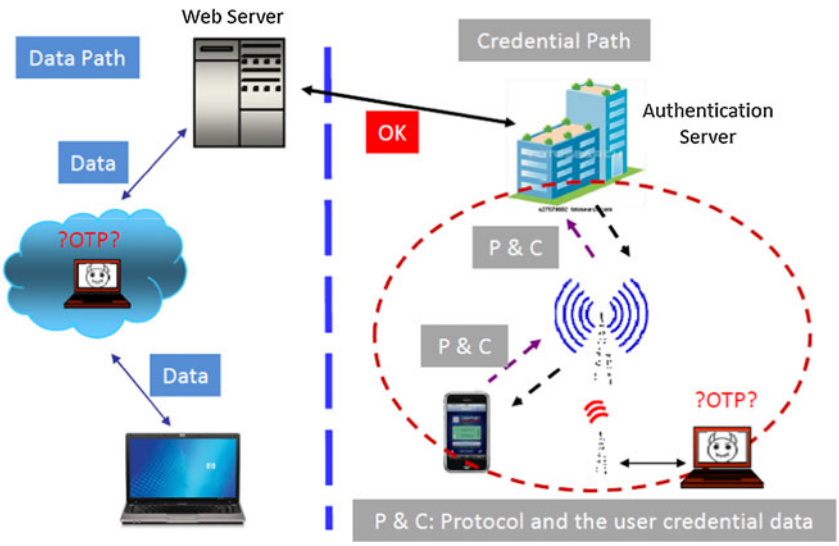


Fig. 6 The operation of a mobile authenticator

Table 2 Comparison among various types of Mobile OTP token

Category	Stand Alone Token	Out of Band TXR	Mobile Authenticator
Role of phone	Computational platform	Transceiver of OTP code	Part of the authenticator
OTP generated by	Phone	Server	Phone and Server
OTP submission	Through PC	Phone, SIM & SMS	Phone, SIM & Protocol
Type of phone	2.5G and up	2.5G and up	3G and up
Simple usability	Yes	No	No
Low phone $/month	Yes (no SMS/data plan)	No (SMS plan)	No (3G Data Plan)
No cellular limitation	Yes	No	No
Compatibility & Interoperability	Yes	No	No
No system change	Yes	No (additional H/W,S/W)	No (Complex system)
Protect secrets	No	Yes	Yes/No
MITM attact safe	No	No	Yes (as of Oct/2010)
Shoulder-surfing attact safe	No	No	?

OTP token – such as the protection of local stored confidential data. This is exactly what the REAL Mobile OTP Token sets out to do.

2.8 Related work on Cloud-based OTP tokens

Using a central server to generate OTP code has many advantages. One is to prevent the token-server de-synchronization issue. We can also benefit from using a highly randomized algorithm for OTP generation. It can greatly enhance the network security level. And the user does not have to carry a physical electronic token. It saves money for the user. Moreover, there is no more risk of security breach due to the loss of an electronic hardware token.

Ideal Cloud-based OTP token has few key features. One is on-demand OTP code generation. The code is generated by the server any time when a user requests. It prevents the accidental exposure of the pre-generated OTP codes during any network break-in. The OTP code is securely encrypted and sent through Internet. The ciphertext is displayed in a web browser window on the requesting user's PC or mobile device. The user then securely obtains the correct OTP code and enters it to complete the 2FA procedure. The OTP code should be fully compatible with the existing authentication infrastructure. Moreover, the Cloud-based OTP Token should successfully resist MITM intercepting attack. Figure 7 lists the basic features of an ideal Cloud-based OTP Token.

2.8.1 Out-of-band transceiver type Cloud-based OTP token

A few solutions were proposed to associate OTP generation with the Cloud. One such method can be called Out-of-band Transceiver type Cloud-based OTP Token. In [26], a user requests the OTP generation to the web server (WS). The WS sends the specific data in a bar code format to the web browser window on the user's PC (PC). The user uses the mobile phone (M) to take picture of the bar code and decode the embedded data with preinstalled software. These data is sent to authentication server (AS) through the phone using the cellular's short message service (SMS). Authentication server verifies the URL data representing the AS and passes the rest of data to web server. Web server verifies the session ID in the data. When the session ID and URL data match the original, a full authentication is obtained. The password's traveling path is WS-PC-M-AS-WS. The data can travel in a reverse order as well. Figure 8 shows operation of such system.

This approach sends the secret data through cellular network. Such application is limited by the cellular network service coverage. The SMS system also cannot guarantee to have in-time or real-time delivery of the secret data. The data can still be intercepted by a malicious people who equips with the proper cellular receiver. Moreover, web server may send personal confidential cookies to a user. Unfortunately, this Out-of-Band Transceiver scheme offers no protection on this. An intruder can easily obtain these cookies by way of MITM attack.

Reference [48] adds a security proxy server between the remote PC and the destined server as shown in Fig. 9. It

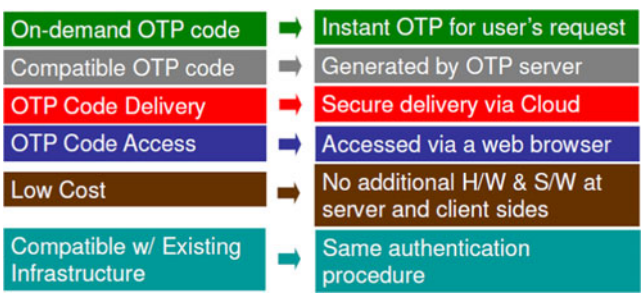


Fig. 7 Key features of an ideal Cloud-based OTP token

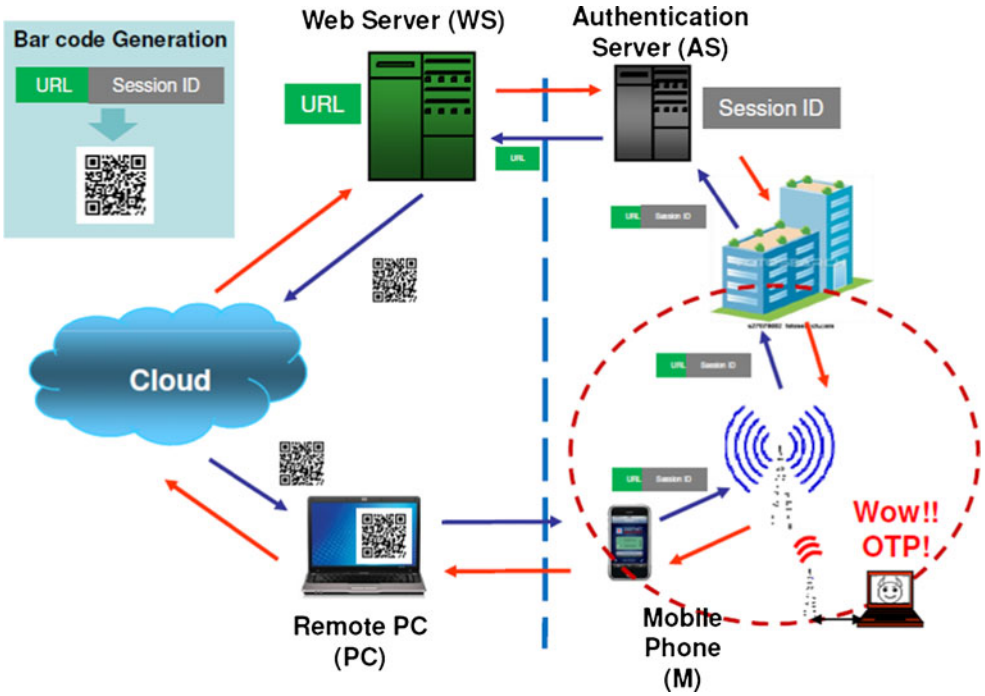


Fig. 8 Out-of-band transceiver type Cloud-based OTP token

uses the Out-of-Band Transceiver to send OTP code through SMS. The proxy serves as a temporary cookie holder on a user's behalf. So the cookies will not be exposed even in a public insecure environment such as public PC or Internet Kiosk. But since it utilizes the same SMS to send secret data, it suffers from the same cellular coverage issue.

2.8.2 Indirect-code Cloud-based OTP token

Some recent proposals [12, 18] use the Internet to send the user an intermediate code to find the actual OTP from a pre-printed PAD. We call it an Indirect-code type of Cloud-based OTP Token. Figure 10 shows the set up of such token. During the registration, a pre-printed PAD is

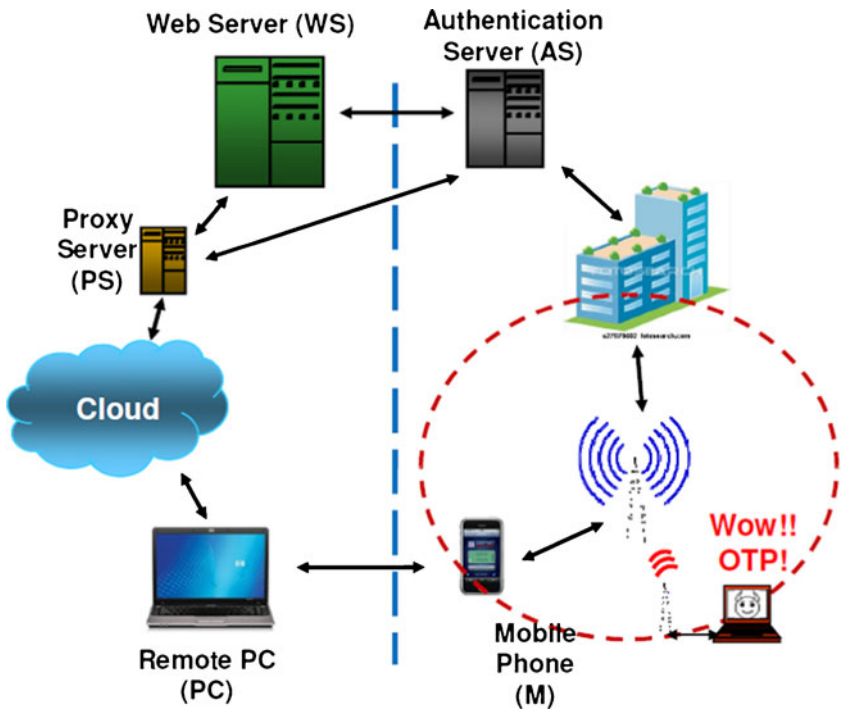


Fig. 9 Proxy with Out-of-band transceiver Cloud-based OTP token

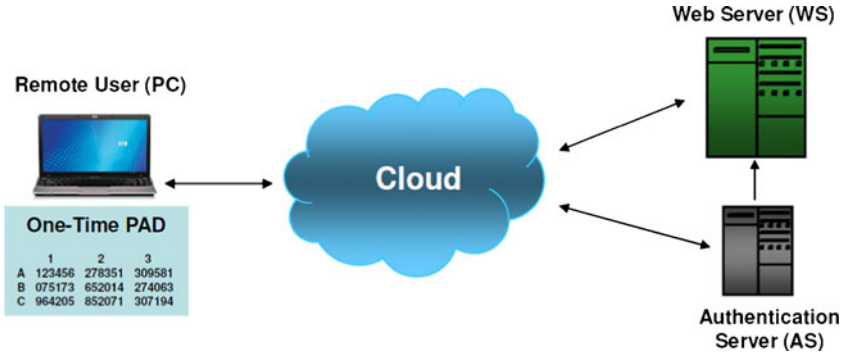


Fig. 10 Set up of an Indirect-code Cloud-based OTP token

generated by server. It is sent to the user through another channel such as regular mail, e-mail or hand delivery.

Figure 11 shows one form of the indirect codes and pre-printed PADs. The server sends an indirect code of 100 and 50 to the user. The user looks up the pre-printed PAD and obtains the OTP code as A3DZ3V. She enters such code and submits it to complete the full authentication.

In this scheme, the OTP code is pre-printed and is not generated as on-demand basis. For different 2FA servers, a user needs to carry different pre-printed PADs. It can be very troublesome when working with many different 2FA systems. The network security is compromised if the PAD is lost, stolen or secretly copied. Furthermore, the exposure of confidential cookies is another security risk.

2.8.3 MITM proxy-based token

In KYPS [31] and KLASSP [9], they use intermediate code to implement the Cloud-based OTP solutions. A MITM Proxy server is used. This method is thusly called MITM Proxy-based Token. A user pre-registers herself with the Proxy Server. Full user credential (user name and password for each destined web server) are stored at the Proxy Server. The user is assigned a password or one form of passcode by the Proxy. The user uses such password to login the Proxy. Once verified by proxy, her full credential is sent to the destined server for final authentication. Figure 12 shows such scheme.

In KYPS (Keep Your Password Secure) scheme, a pre-printed random indirect code is used as passcode for proxy server login. In operation, the user contacts Proxy by sending in her UserID and destined web site name. The password is recovered once the user sends in the correct indirect code after server's challenges. Proxy server then sends both the UserID and recovered password to the destined server for authentication. Proxy sits in the middle between the destined server and the user. So the web server's URL or security cookies are not exposed.

In KLASSP (KeyLogger Avoidance using Shared Secret Proxy) scheme, the user pre-registers at the Proxy with her userID (such as janec123@hotmail.com). The Proxy login password is a shared secret that is pre-determined by the Proxy. The shared secrets can be a symbol and certain location in a symbol table that the user will looks for it. During the login, the Proxy sends a symbol table for each password character entry. If a correct symbol is shown in the right location, the user keys in the correct password character otherwise a random one. This action repeats until the full password is completely entered. The Proxy then sends the UserID and recovered password to the destined server for final authentication.

Both schemes do provide certain protection on multiple sites authentication with an intermediate random code for each site. In KYPS, the recovered password by the Proxy is always static and cannot meet the 2FA system's OTP requirement. For KLASSP, it is possible to send a dynamic OTP code through its encrypted password entry scheme. But the user will need to have certain means to generate the OTP code. KLASSP then becomes a redundant procedure of the OTP submission step. Overall, the "MITM Proxy Server" scheme cannot be compatible with the existing 2FA systems.

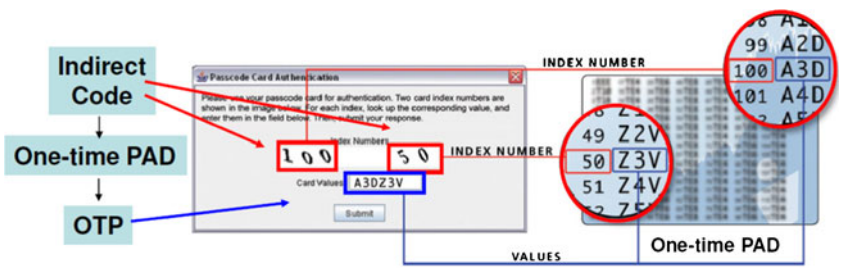


Fig. 11 Indirect code sent by server

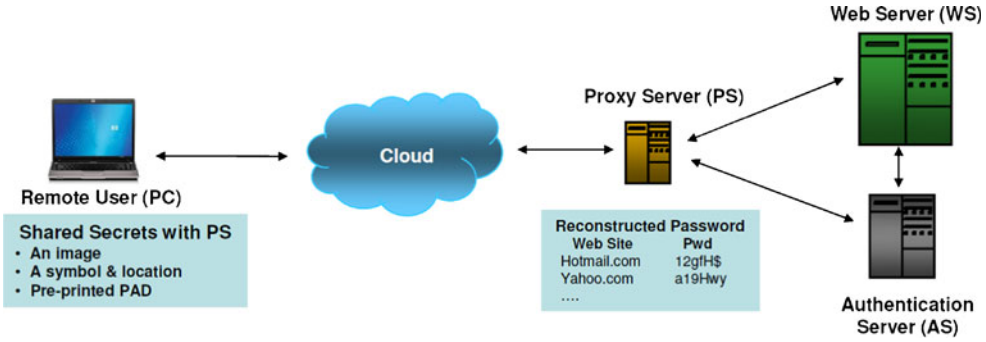


Fig. 12 MITM Proxy-based OTP token

2.8.4 Short summary on Cloud-based OTP token

Table 3 summarizes the analysis of various Cloud-based OTP tokens discussed. Let's try to compare them to a few key features of an ideal Cloud-based OTP token.

- On-demand OTP code
  Only Out-of-band Transceiver type of token has capability meets this requirement. But it cannot guarantee the on-demand delivery due to the use of SMS.
- Compatible with existing 2FA infrastructures
  Out-of-band Transceiver and Indirect code type can meet this item. But they all fail to meet the on-demand OTP code generation requirement.

It clearly shows that a good and reliable Cloud-based OTP token is still yet to be invented. It is a good challenge for us to work on.

3 Rubbing encryption algorithm

We have seen pros and cons among many different schemes in Section 2 on how to transmit and protect the OTP code. We propose a new and secure OTP token to overcome those deficiencies. To achieve that, we use a new encryption algorithm to protect the secrecies that generate the OTP code. This new algorithm is called Rubbing Encryption ALgorithm (REAL). We will review related background and encryption algorithms prior to the introduction of REAL. We begin with the review of short message encryption method.

In a cipher system, the ciphertext's security level usually depends on its encryption key (K) length. The longer length the encryption key is of, the higher security level its ciphertext has. For most symmetric ciphers, the chosen key length usually is smaller than that of the plaintext. An extra long key may not be fully utilized in a short plaintext encryption. Encrypting a short message such as a 6 to 8 digits OTP code does present itself a paradox between the security level and the chosen key length.

Two classical cipher systems Transposition Cipher and Substitution Cipher, seem to provide a secure and easy encryption scheme for short plaintext. They can be the good candidates for OTP code encryption.

Transposition cipher uses a key (K) as the symbol sequence pattern to encrypt a plaintext. Dummy characters can be padded onto the plaintext. So the plaintext can be evenly divided into many small blocks that have the same length (L) as the chosen key. Each block will then follow

Table 3 Various types of Cloud-based OTP schemes

Category	Out of band Channel	Indirect Code w/reprinted PAD	MITM Proxy Server
Proxy server	N/A	N/A	MITM secrecies holder
OTP generated by	OTP Server (activated thru web)	OTP Server (generated & deliverd in advance)	Proxy server (not for 2FA application)
OTP submission	Through PC	Phone, SIM & SMS	Phone, SIM & Protocol
Authentication by	Real-time OTP code	Pre-printed OTP	Static password
2FA compatibility	Yes	Yes	No
Safe using public PC	No (loss privacy cookie)	No (loss privacy cookie)	Secure
Limitation	Cellular service area	Carrying Multi-PADs	Trusted Proxy set up
Compatibility & Interoperability	No	No (Proprietary)	No for 2FA (Yes for simple pwd system)
No system change	Yes	No (additional H/W,S/W)	No (Complex system)
System change	Additional H/W & S/W	Additional S/W	Additional Proxy H/W
Protect secrets	No (when loss phone)	No (when loss reprinted PAD)	Yes (if proxy Trustworthy)
Seed Tracing Safe	No	Yes/No	Yes/No

the key pattern (K) to swap its symbol sequence. For ease of memorizing the key pattern, a key-word is used. The symbol's alphabet order sequence in the key-word represents the key pattern of this encryption. The new symbol stream that makes up from each of these transposed blocks is the ciphertext.

The above scheme has "L! – 1"[2] keys (one is the original plaintext). Given an OTP code of "437891" and a key-word of "362514", we will have a ciphertext of "713948". The key length is 6. We have total of 719 set of key (= 6! - 1). The probability of finding the correct key or plaintext OTP code is 1 out of 719. This is considered a very poor security level. So it is very difficult to protect the OTP code security using such simple scheme.

The transposition system is an old cipher. It only involves changing the symbol's sequence. So it does not demand high computing power or large memory buffer. This cipher consumes little energy. It is an ideal cipher system for low cost, mobile or portable applications where CPU speed, memory size and power supply are of certain limitations. But for a plaintext with small word length (say 6 to 8 symbols long), it is very difficult to use a larger key to ensure the desired security on ciphertext.

A substitution cipher, on the other hand, replaces the plaintext's symbols with other symbols. It does not alter the symbol sequence of the original plaintext. It is very easy to encrypt the plaintext as long as the substitution table is available. One of the famous substitution ciphers is the Caesar's Cipher which was very popular 2,000 years ago.

For a 6-digit OTP code using zero to nine numerals, we can have total of 1 million sets of different keys if each symbol is allowed to reappear. The key size is large enough to protect the ciphertext. So a substitution cipher may be a good encryption algorithm for short OTP code. But finding an easy way to memorize the key and securely protection of the encryption key are the important factors for us to work on.

In [25], it provides a good reference on how to effectively use the mixture of two different ciphers. The authors combine a high quality scrambling system (transposition cipher) with a strong encryption mechanism. The message is scrambled first by a Smart Card with less powerful CPU. A small block (e.g. 128 bit block) is chosen from the scrambled message to be encrypted with a chosen encryption algorithm. The algorithm is based on a shared hash function and key (k) that transferred by a handshaking between host computer and Smart Card.

The authors of [25] prove that the mixed scheme can deliver a fast and secure cipher system. They further prove that a simple good quality transposition cipher system (scramble cipher) can achieve the desired security as long as the scrambling key is well kept.

3.1 REAL mathematic background

Rubbing Encryption Algorithm (REAL) was proposed in both [4] and [5]. The goal was 1) to securely encrypt the short OTP codes, 2) to safely deliver the ciphertext to a remote device through the Cloud, and 3) to securely store the ciphertext in a designated remote device. The remote device can be a personal computer, a notebook PC, a cellular phone, a mobile Internet device or other similar function products. Furthermore, there is no need to enter the encryption key when decrypts a REAL ciphertext. REAL also does not demand high computing power.

REAL bases on a secure scramble cipher. The OTP code is scrambled into a lengthy symbol stream or a large size matrix but still maintaining the OTP's original symbol sequence. REAL then maximizes the uncertainties of the stream or matrix to ensure a secure ciphertext.

Given an M x N matrix (X) made up from Y variety of symbols (S), the probability (P_i) of a symbol (S_i) being displayed in the matrix X is

$$P_i = a_i/MN, \tag{3}$$

where a_i is the total number of occurrences of a symbol S_i displayed in X. The matrix size (T) can also be shown as follows.

$$T = MN = \sum_{i=1}^{Y} a_i. \tag{4}$$

The probability to accurately predict each element's symbol in matrix X is related to X's uncertainty H(X). The higher a matrix uncertainty is, the tougher to correctly guess its element's symbols. The matrix X's uncertainty can be described by the entropy of P_i's probability distribution. Using Shannon's definition of the entropy for discrete information source [37, 39], we can show matrix X's uncertainty H(X) as follows.

$$H(X) = -\sum_{i=1}^{T} P_i(\mathrm{Log}_2 P_i). \tag{5}$$

Consider a special probability distribution where every symbol has an equal chance of occurrence and equal number (a) of symbol is displayed in the matrix X. Its probability distribution reaches an equiprobable state. Such a matrix X is called an Equiprobable Matrix. The size (T) of an equiprobable matrix is then represented as follows.

$$T = \sum_{i=1}^{Y} a_i = \sum_{i=1}^{Y} a = aY, \tag{6}$$

[2] L! is L factorial. $L! = L \times (L-1) \times \ldots \times 2 \times 1$.

where a is a positive integer. Each symbol's occurrence probability (P_i) then becomes

$$P_i = a_i/MN = a/T = a/aY = 1/Y. \quad (7)$$

From (7) we can find that for a fixed set of symbol used in the elements of an equiprobable matrix, its occurrence probability (P_i) is equal to the inverse of its variety (Y). It has been shown in [38] that H(X) will reach its maximum value when X is an equiprobable matrix. By substituting (6) and (7) into (55), we can reduce the equiprobable matrix's uncertainty H(X) to a much simpler form.

$$H(X) = T((Log_2 Y)/Y) = a(Log_2 Y). \quad (8)$$

Equation 8 and Fig. 13 show that H(X) is directly proportional to the equiprobable matrix size (T). The higher the matrix size is, the higher uncertainty the matrix X has. So it is much tougher to correctly guess the symbol displayed in a larger size of equiprobable matrix using a fixed set of variety of symbol in its elements.

We can further derive the uncertainty H(S) of a Y variety equal-occurrence symbol (S) which will be chosen to display in a matrix X. The probability (P_{Si}) of a symbol (S_i) being chosen as the element of matrix X is as follows.

$$P_{Si} = 1/Y. \quad (9)$$

The uncertainty H(S) of symbol S can be derived from the probability distribution P_{Si}. Again, using the definition of Shannon entropy [39], we have

$$H(S) = -\sum_{i=1}^{Y} P_{si}.(Log_2 P_{Si}). \quad (10)$$

Substitute (9) into (10), H(S) can be reduced to

$$H(S) = Log_2 Y. \quad (11)$$

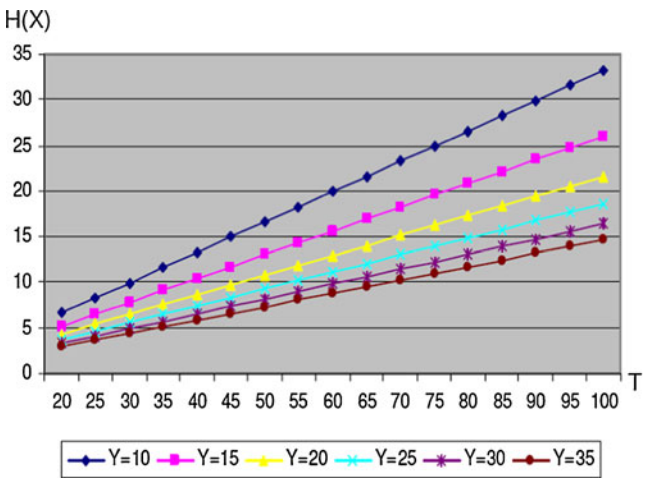


Fig. 13 Matrix's uncertainty versus different matrix size

It is interesting to find that for an equiprobable set of symbol its uncertainty is simply equal to its own variety (Log_2 Y).

From the equations listed above, we can reach the following conclusions on the Equiprobable Matrix's (X) properties and its relationship with a chosen Y variety symbol (S) used in its elements.

- An equiprobable matrix has the highest uncertainty H (X) among all the similar size of matrices.
- The uncertainty increases proportionally with its own matrix size (T) when a fixed variety (Y) symbol set is used in its elements.
- The uncertainty becomes smaller when the symbol variety grows larger while matrix size is kept fixed.
- H(X) reduces drastically and levels off as symbol's variety (Y) increases.
- Symbol S' uncertainty H(S) is equal to its own variety. But due to the logarithmic effect, the uncertainty will increase at a slow pace and level off very fast as the variety (Y) increases.

Based on the conclusions, we can securely hide a short plaintext or OTP code (O) in an equiprobable matrix (X) as long as X uses O's symbol set in its elements. Such an equiprobable matrix ensures the desired security of the OTP code in the X. We can also further enhance the OTP code security by optimizing the matrix size (T) and symbol variety (Y) according to Eqs. 8 and 11.

3.2 REAL encryption procedure

During encrypting, REAL places the original OTP code symbols as part of the elements of the matrix (X). Such matrix is also called REAL Image (RI). A RI can be in the form of a data string (one dimension), a two dimensional matrix or can be of other multiple-dimension spatial forms. By maintaining an equiprobable image in each dimension, these multiple dimensional REAL Images still retain the high uncertainties. They ensure the desired security of the ciphertext. The Rubbing Encryption Algorithm has such intrinsic feature. It enables a new spatial encryption method to protect the ciphertext with desired security. The corresponding REAL encryption key is then of the same multiple dimensions as well.

For ease of discussion, an M x N matrix X (REAL Image) and numerals of 0 to 9 as the set of symbol are chosen to illustrate the proposed REAL. A 6-digit (D=6) OTP code of "381269" is the plaintext for encryption. REAL encryption procedure is as follows.

3.2.1 Drafting a REAL image

REAL encryption starts out by designing a RI to be displayed on the remote device's screen. Screen size, image

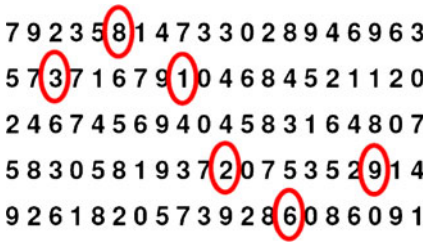


Fig. 14 Example of a basic REAL Image (REAL matrix)

symbol font size, symbol set and total image element count (matrix size) are the major factors to consider when designing a RI. More importantly, RI is designed for easy viewing on the remote device screen. Figure 14 shows an example of REAL Image with size of 5×20.

3.2.2 Generating REAL key

REAL encryption key is the specific spatial locations (W_i) where the plaintext's symbols are placed inside the RI. Figure 14 shows those red circled locations where plaintext OTP code is hided.

Given a 5×20 RI, we can use a credit card like of plastic card as REAL hardware token. Figure 15 shows one such example. Transparent windows are randomly placed on the token surface. For D symbols of plaintext, we have "D" number of transparent windows on the hardware token. Each plaintext's symbol (S_i) will be placed in one of the opening window locations (W_i) according to its original sequence order. So the locations of these opening windows are also the key for REAL encryption and decryption.

3.2.3 Generating an actual REAL image

REAL places the plaintext symbols into the corresponding W_i locations in matrix X according to each symbol's occurring sequence. Since D=6, plaintext code=381269, we then have D_5=3, D_4=8, D_3=1, D_2=2, D_1=6 and D_0=9 for the plaintext code. Row I and II of Table 4 show how each plaintext OTP symbol is assigned to the corresponding window location.

In a RI with size T, each W_i has its own specific element a_{ij} in the matrix X. These elements are filled with the corresponding symbols shown in row II of Table 4. The rest of "T - D" locations are filled with randomly chosen symbols so that matrix X is equiprobable. Such matrix X is the REAL Image.

Figure 16 shows a 5×20 RI. It has total of 100 (T) elements. We have 10 symbols (0 to 9 numerals). Each symbol occurs 10 times. Each symbol has equal chance of occurrence (1/10). So such a RI is an equiprobable matrix. It has the highest uncertainty among the size 100 matrices.

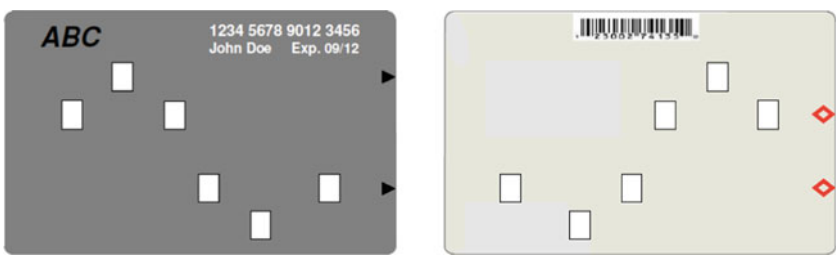
An alignment marker is included in the RI. This is the vertical line with the black solid triangle plus red diamond symbols. Alignment markers are printed on the token's right edge on both the front and rear sides as well. The alignment marker will help a user to accurately align the hardware token over the RI. The alignment markers also indicate which side of token should be used when rubbing (decrypting) the RI (ciphertext).

3.3 REAL decryption procedure

To decrypt the ciphertext, a user simply overlays the REAL hardware token over the RI. After properly align the token, the plaintext (OTP code) will clearly appear through the transparent windows on the token. Figure 17 shows such an example. The original OTP code is rubbed (decrypted) by the token and is ready for 2FA application. Notice that the user does not need to enter the REAL key to decrypt the REAL Image (ciphertext).

4 Implementing a REAL Cloud-based OTP token

In this Section, we will implement a Cloud-based OTP token using REAL as the base cipher. We will show the implementation first. We then use this OTP token to demonstrate and analyze REAL's features. We will also analyze the token's security level. This token is compliant to OATH OTP standard. The OTP passcode is set to have 6-digit. The Cloud-based OTP Token uses OATH time-based OTP algorithm (TOTP) [24]. It helps to prevent the Counter De-synchronization problem between an OTP



a. Front side b. Rear side

Fig. 15 Basic REAL hardware token. **a**. Front side, **b**. Rear side

Table 4 Generation of a REAL matrix (REAL Image)

	W_5	W_4	W_3	W_2	W_1	W_0
I	D_5	D_4	D_3	D_2	D_1	D_0
II	3	8	1	2	6	9

token and authentication server. The design goal is to meet as close to the ideal Cloud-based OTP token's requirements (discussed in Section 2) as possible.

4.1 Designing a REAL Cloud-based OTP hardware token

4.1.1 Token's form factor and material

According to a Federal Reserve Bank of Boston survey report, 609.8 million credit cards were held by U.S. consumers in 2008 [10]. A Visa USA internal statistics published in 4th quarter of 2006 indicated that consumers carry more than 1 billion Visa cards worldwide back then [47]. These reports tell us that the credit card's form factor is well accepted by the consumer besides its convenience in financial transaction. So we choose the REAL hardware token to be with the same form factor like a credit card (e.g. Fig. 15). Its dimension is 3.375"×2.125"×0.03" (L x W x T). The material of this token also conforms to credit card's ISO/IEC FDIS 7810 standard [19]. The user can easily carry this REAL hardware token in her wallet or pocket.

The token manufacturing process is very similar to a regular credit card. With same material and similar manufacturing process, the cost of a REAL hardware token is comparable to a credit card.

4.1.2 Basic REAL image and ciphertext image generation

In our example, the REAL Image uses a symbol font size of 12 and a numeric array of 5×20. We have total of 100 (T)

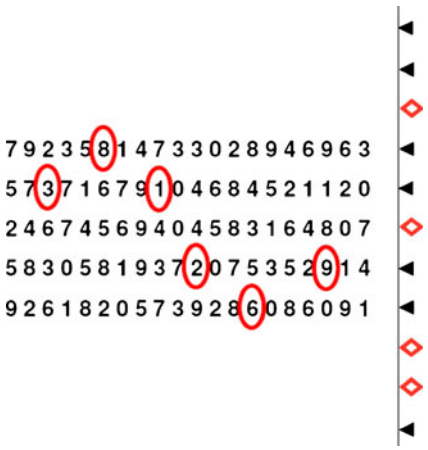


Fig. 16 REAL Image with the side alignment marker

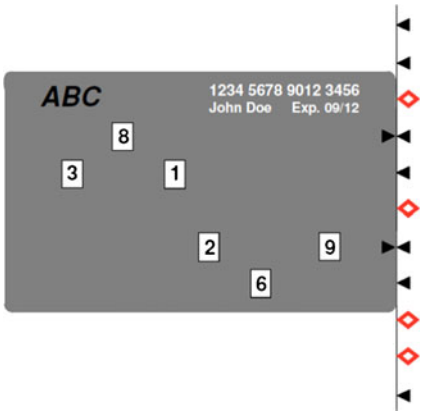


Fig. 17 Rubbing (decrypting) OTP code from the REAL Image

elements in an equiprobable RI. Figure 18a shows such a basic RI example.

To further improve the OTP code protection, we design an 11×20 Ciphertext Image (CI). Entire RI as a whole is randomly embedded inside this CI by OTP server. In the CI, each numeric symbol has 22 occurrences. The CI is an equiprobable matrix as well. It has the highest matrix uncertainty among all the 11×20 matrices. It offers additional level of security protection to the RI and OTP code. Figure 18b shows the details of such Ciphertext Image.

4.1.3 Dynamic REAL key scheme

In Fig. 18a, to find the OTP code without a REAL encryption key (the hardware token), the probability (P_1) is

$$P_1 = 1/C(100, 6)) = 8.4E - 10. \quad (12)$$

This is quite a secure level of protection when all 6 digits are all of different symbols. But considering an extreme case: 1). all 6-digit OTP symbols are the same numeral, and 2). both the RI and OTP code were captured by the intruder, the probability (P_{1E}) of finding the REAL key is then

$$P_{1E} = 1/C(10, 6) = 2.3E - 04. \quad (13)$$

This is an alarming situation. The REAL key is literally without much protection.

To increase the RI's security, a dynamic REAL key scheme is implemented on this Cloud-based OTP Token. Such Dynamic REAL Key scheme is implemented in three steps. First, 4 additional transparent windows (R) are added on the hardware token besides the original 6 windows (D). The original 6 windows arrangement will have a REAL key count of

$$\text{REAL Key Count } (6\_\text{Windows}) = 2 \times C(100, 6) = 2.4E + 09. \quad (14)$$

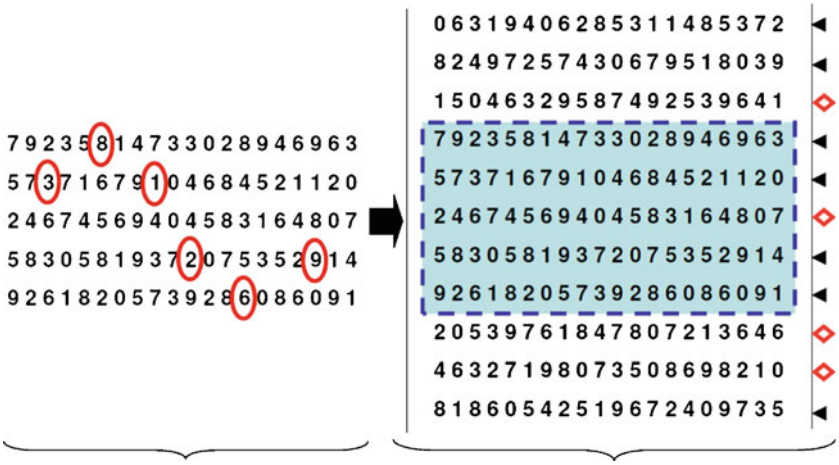


Fig. 18 Basic REAL Image and Ciphertext Image. **a**. REAL Image, **b**. Ciphertext Image

The addition of another 4 windows will increase the total possible REAL key count to

$$\text{REAL Key Count } (10\_\text{Windows}) = 2 \times C(100, 10)) = 3.4\text{E} + 13. \quad (15)$$

It is an increment of 14.5 thousand times. And only one of the 3.4E+13 REAL keys is the correct one. Moreover, among the 10 windows only 6 of them are the correct ones to show the valid OTP code. This increases the key count by an additional factor of

$$\text{Incremental factor} = C(10, 6)) = 210. \quad (16)$$

In other word, for a given 10-window REAL hardware token, OTP server has 210 ways to randomly place the 6-digit OTP code. It helps to prevent the Shoulder-surfing attack that obtains the window locations of the REAL key.

Secondly, a randomly chosen set of 7 windows are circled with a thick pink color band on their peripheries (see Fig. 19). Such windows are called "Tagged Window". Tagged Window interacts with the "Tagged Symbol" to complete the third step of Dynamic REAL key scheme.

A "Tagged Symbol" is the symbol that has a thick pink color bar on its background such as shown in Fig. 20. The OTP server places the color bar tag by referencing to the hardware token's tagged window locations before finishing the final Ciphertext Image. A tagged symbol seen through a tagged window is considered as a redundant symbol and will be ignored. A true OTP symbol can be a tagged symbol as long as it is not shown through a tagged window. So both the tagged symbol and tagged window add more randomness into the ciphertext. It thusly provides significantly better security protection to the OTP code hided inside the CI.

Going back to the extreme case in (13), by using the tagged symbols and 7 tagged windows, the probability (P_{t1}) of finding the tagged windows in the 10 openings is

$$P_{t1} = 1/C(10, 7) = 0.008. \quad (17)$$

The probability (P_{t2}) of finding the 4 non-valid tagged windows is

$$P_{t2} = 1/C(7, 4) = 0.029. \quad (18)$$

So overall, the probability (P_1) of finding the REAL Key is

$$P_1 = P_{1E} \times P_{t1} \times P_{t2} \times (1/2) = 2.7\text{E} - 08. \quad (19)$$

This probability is much smaller than a direct guess of the 6-digit OTP code which has probability of 1E-06. So the security level is greatly enhanced by this Dynamic REAL Key scheme.

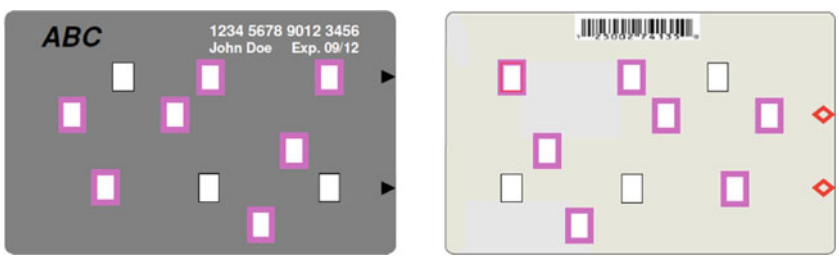


Fig. 19 Credit card like REAL hardware token with tagged windows. **a**. Front side, **b**. Rear side

Fig. 20 A REAL and Ciphertext Images with tagged symbols. **a**. Tagged REAL Image, **b**. Tagged Ciphertext Image

4.1.4 Token database and other

Using different front and rear alignment markers (as shown on the right edge of the token in Fig. 19) allows each token to embed two set of windows (W_i) or two sets of encryption keys (K_i). Aligning and matching the RI's markers with the token's, we can determine which key to use (or to use which side of token) during REAL decrypting.

Each hardware token has a bar code serial number that links to its confidential file. The file contains the token's window location data (REAL encryption key) and the alignment marker type plus location data. Each user is assigned a unique hardware token. The token's serial number is linked to the user's credential information (UserID, password and/or PIN) and stored securely in a database server for future use.

4.2 REAL Cloud-based OTP software

REAL Cloud-based OTP Token has two major parts of software. They are Program File and Data File. Both files are securely stored at server. The Program File contains the executable files to generate OTP, REAL Image, Ciphertext Image and other housekeeping tasks. The list of Program File includes the following programs.

- OATH TOTP Code Generation program
- HMAC-SHA-1 program
- Ciphertext Image Generation program
- End of session housekeeping program
- Help Menu with Demo program and Version Number

The above OATH TOTP Code Generation program uses the exact OATH Time-based algorithm (TOTP) published in [28]. The algorithm is as follows.

$$\mathrm{TOTP} = \mathrm{Truncate}(\mathrm{HMAC} - \mathrm{SHA} - 1(\mathrm{K}, \mathrm{T})), \qquad (20)$$

where K is a user specific 160 bit key, T is a value associated with the actual Internet time.

Data File consists of two major components – User Credential File (UC File) and OTP File. UC File stores the user's credential data (UC) and the user's accumulated OTP server access count (q). A user's credential data includes UserID and Cloud-based OTP access password (COP). The two data are hashed before securely storing at a user verification database server. Its generation is shown in the following pseudo code (4.9).

$$\mathrm{UC} = \mathrm{HMAC} - \mathrm{SHA} - 1(\mathrm{UserID}, \mathrm{COP}). \qquad (21)$$

The OTP File includes the hardware token's serial number, OTP code generation keys (K), token's window locations and alignment marker type plus location data. To further protect each user's secret OTP File, a 160 bit encryption key (CK) is generated from the user's credential information (UC) and her accumulated Cloud-based OTP server access count (q). Each user's OTP File is encrypted with her personal dynamic key (CK). The CK is generated as follows.

$$\mathrm{CK} = \mathrm{HMAC} - \mathrm{SHA} - 1(\mathrm{UC},\ \mathrm{AND}(\mathrm{UC}, \mathrm{q})). \quad [3] \qquad (22)$$

The encrypted OTP file is stored in another secure database server by following the value significance order of each user's UC. In other word, each user's UC is the memory locator of the encrypted OTP File in a separated database server. Such scheme protects the user's secret OTP File even if the server is broken-in by either any inside or outside intruder.

[3] "AND" means a logical AND operation function.

4.3 Secure, scalable and reliable servers and robust network

Since the OTP is generated on the fly, a high scalability server is used to meet the potentially large volume and simultaneous OTP requests from many users. It is the key factor to ensure a successful Cloud-based OTP service. High speed server is used to ensure faster delivery of the Ciphertext Image.

The server will synchronize its clock to an accurate Internet time source such as NIST's Internet Time Service NTP servers [30]. NTP server is very accurate and with extreme fine resolution (200 pico-seconds per increment). Such fine resolution, stable and accurate time source automatically ensures the desired uniqueness and randomness for each generated OTP code.

REAL Cloud-based TOTP program can be running as a standalone program in a separate web site. The user will be redirected to the OTP server site by her destined server during a 2FA login process. The OTP server then generates and securely sends the code to both the user (through REAL encryption) and her destined server. After obtaining the correct OTP code, the user then enters the OTP code to her destined site to complete the 2FA process. Ideally, it can also be integrated into a 2FA system and works as an add-on software module to the authentication server.

A reliable Cloud-based OTP token operation relies on the secure network set up and its protection mechanism. It should effectively resist various malicious attacks – such as denial of service and many other attacks from the Cloud. The Hypertext Transfer Protocol Secure (HTTPS) [32] is engaged when a user starts to login the web site. HTTPS' extra protection will further reduce the level of malicious attacks existing in the Internet.

4.4 Operating a REAL Cloud-based OTP token

To operate a REAL Cloud-based OTP Token, a user links to a destined web site. She activates the REAL Cloud-based OTP Token by sending in her UserID and COP through HTTPS. Server will take the received UserID and COP using (21) to generate a hash digest. The hashed digest is then compared to the UC data stored in the verification database server. If there is a match, the user is allowed to access the Cloud-based OTP server. Obtaining both the UC and server access count (q) from verification server, the user's personal encryption key (CK) is then generated following (22). With UC and personal encryption key, TOTP generation key (K) and the user hardware token's window, tag and alignment marker data can be obtained. By using (20) and TOTP key (K), the OTP code is generated and ready for REAL Image and Ciphertext Image generation.

Figure 21 shows the details of the Cloud-based OTP operating procedure. The pseudo code to generate a Ciphertext Image is listed as follows.

CIPHERTEXT_IMAGE_GEN (K, T)

0 UC = User Credential //input by a user when activates OTP generating process.

1 Using UC, server retrieves the user's OTP secret key (K) and W_i location data on the REAL Image.

2 OTP = Truncate(HMAC-SHA-1 (K, T)).

3 OTP = Concatenate(D_5|D_4| D_3|D_2| D_1|D_0).

4 Sequentially and randomly placing D_5 through D_0 into the 10 W_i locations inside the 5 x 20 matrix. Let other unfilled W_i be open and without any numeral.

5 Fill the rest of matrix elements including the unfilled W_i with randomly chosen symbols. So that each symbol has 10 occurrences. This is REAL Image.

6 Place REAL Image inside the 11 x 20 matrix. Fill the rest of matrix elements with randomly chosen symbols. So that each symbol has 22 occurrences. Place tag bar on the elements so that only REAL Image will display the correct OTP code through user's hardware token. This is the Ciphertext Matrix.

7 Place alignment lines on each side of the Ciphertext matrix that matches to the actual hardware token's width. Place the correct alignment marker on the right hand side of the alignment line. So when properly aligned the correct OTP code will be shown through window openings. This is the Ciphertext_Image.

8 End of program.

Ciphertext Image (CI) is sent through Internet using HTTPS protocol to display on the user's PC as shown in Fig. 22a. The user overlays and aligns her hardware token on the screen to decrypt (rub) the correct OTP code from the Ciphertext Image as shown in Fig. 22b. A double-click on the "OTP Now" button will initiate another new round of OTP code generation. The user can then rub additional OTP code for another 2FA need. The OTP code rubbing sequence is as follows.

- Starting from the top of the far left column,
- Reading vertically downward till reaching the end of the column,
- Moving to the next column and reading through the end of this column, and
- Following such sequence until reaching the bottom of the far right column on the token.

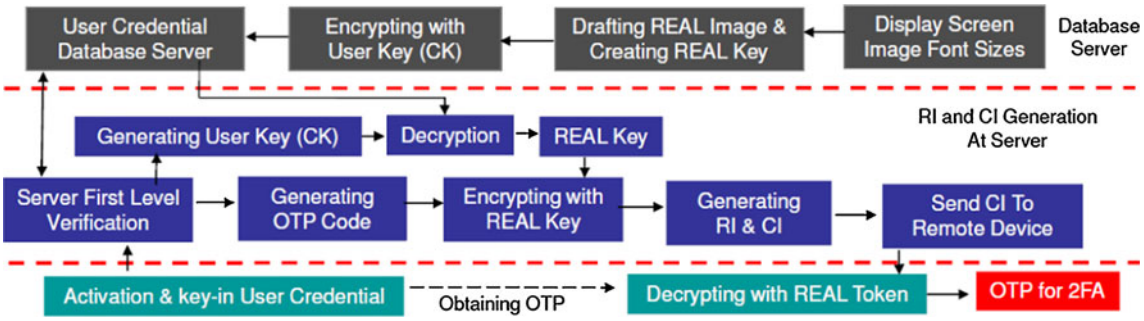


Fig. 21 Cloud-based OTP token operational procedure

Following the above sequence and ignoring the tagged symbols (5, 3, 6 and 9) that shown through the tagged windows, the first six numerals are the OTP code. Tagged symbols 8 and 9 are displayed in the none-tagged windows (thin black window boundary). They are legitimate symbols and are part of the OTP code. In Fig. 22, the OTP code is 381269. The user then enters this OTP code into the login window to complete the 2FA process.

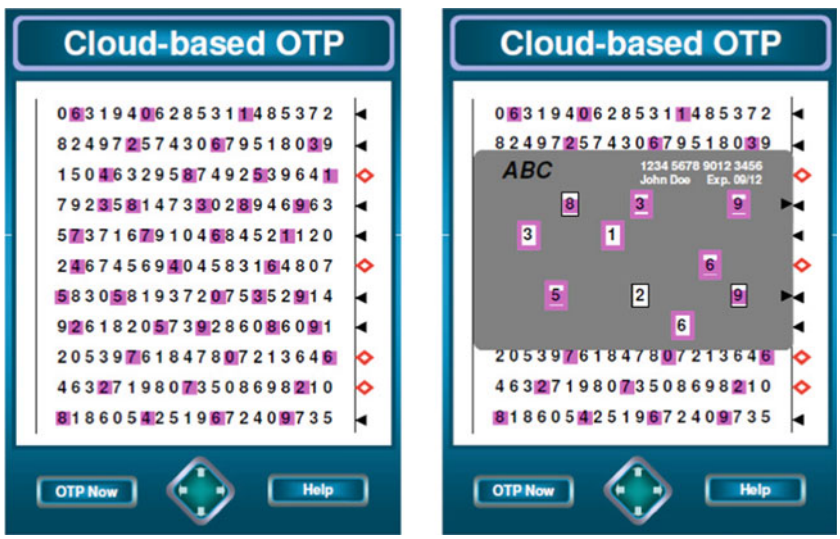
4.5 Design goals review

From market survey as of Q2, 2010, a regular hardware OTP token was priced around $5 [8] to $40 or even higher. A software OTP token was priced about $3 to $10 [36] or higher. The REAL Cloud-based OTP Token requires neither any electronic hardware nor installing any proprietary software on the user side. All the user need is to carry a credit like REAL hardware token to rub the OTP code through a web browser. Besides easy to carry, the REAL hardware token is very inexpensive with a cost comparable to a popular credit card (less than $0.50 each).

REAL Cloud-based OTP Token uses the exact same OATH TOTP generation algorithm to generate the OTP code. The OTP code is 100% compliant to any OATH authentication server. It is also 100% interoperable with any other OATH TOTP token that uses the same key (K) and time value (T).

The hardware token can be easily mailed to a user following the similar logistic of a credit card. After receiving the hardware token and using HTTPS, a user logins into a designated web site (specified by the server) and submits her token's serial number to activate the token. The activation can also be done through a home phone which follows a similar secure procedure that a regular credit card does. These simple and proven secure activation procedures ensure a low deployment cost as well.

A user activates the OTP generation program entirely through a standard web browser on a PC or Internet devices. No proprietary software installation is required on the user side. Technical support can be done easily through Internet to the remote user. It can significantly reduce the



a. Ciphertext Image b. Rubbing OTP

Fig. 22 Cloud-based OTP token. **a**. Ciphertext Image, **b**. Rubbing OTP

supporting cost of such token and its operation. The user does not need to learn any new software operation. It ensures a quick adoption by a new user.

4.6 Security analysis

4.6.1 OTP code, REAL image and Ciphertext image security

REAL Cloud-based OTP Token uses the exact OATH TOTP algorithm to generate an OTP code. It does not replace or alter the OTP symbol during the encryption and decryption processes. So the OTP code is of the same security level as that generated by a regular OATH OTP token. In other word, REAL Cloud-based OTP Token does not degrade the OTP code security level. A detailed OATH OTP code security analysis can be found in Appendix A of RFC4226 [27] for further reference.

For a REAL Image of 5×20, the total number of REAL key is equal to the number of 10-window pattern times the dynamic session key. The detail is as follows.

$$\text{Total } 10-\text{window Pattern per side(TWP)} = \mathrm{C}(100,10) = 1.7\mathrm{E}+13. \tag{23}$$

$$\text{Total Dynamic Session Key per side(TDK)} = \mathrm{C}(7,4) = 35. \tag{24}$$

$$\begin{aligned}\text{Total REAL Key per side} &= \text{TWP} \times \text{TDK} \\ &= 6.1\mathrm{E}+14.\end{aligned} \tag{25}$$

$$\text{Total REAL Key per token} = \text{TWP} \times \text{TDK} \times 2 \text{ sides} = 1.2\mathrm{E}+15. \tag{26}$$

Having a REAL Image but without the hardware token, to correctly guess a window pattern (REAL key and OTP code), the probability (P_a) is

$$P_a = 1/\text{Total REAL Key per token} = 8.3\mathrm{E}-16. \tag{27}$$

In our implementation, the REAL Image is embedded inside Ciphertext Image. The probability (P_b) of finding the correct location of the actual REAL Image from a known Ciphertext Image is $1/7(=1/(11-5+1))$. So given a known Ciphertext Image, the probability (P_1) of correctly finding the REAL key and OTP code is

$$P_1 = P_a \times P_b = 1.2\mathrm{E}-16. \tag{28}$$

So it is significantly tougher on cracking either image than a brute force guess of the 6-digit OTP code which has a probability of 1E-06.

Moreover, finding one session of REAL key with a known REAL Image does not automatically guarantee the intruder to find the next REAL key. The dynamic session key scheme enables the server to throw in another random complexity at the final Ciphertext generation step. Number of the dynamic session key increases when tagged window or tagged symbol increases. This dynamic session key increases the complexity of REAL key. It ensures the freshness of the REAL encryption key for every session. It also increases the total possible REAL key count. So it guarantees the desirable security level when sending and displaying Ciphertext Image in public kiosk or PC.

Furthermore, all the symbols used in the elements of both REAL and Ciphertext Images are all in pure image form. They are not in a regular text data format. So these images cannot be read directly by a malicious program such as certain Trojan. It helps to protect the security on both REAL Image and Ciphertext Image.

4.6.2 When hardware token is lost, stolen or secretly photo copied

A regular OTP token can automatically generate an OTP code with the push of a button on the token. Whoever gets hold of the token has the full capability to generate all the future OTP codes. The security is compromised until such token is removed from the active service list in the server database.

When a REAL hardware token is lost or stolen and falls into an intruder's hand, its protection mechanism is the UserID and COP password. The security level will be as strong as how the user chooses the UserID and COP password. The token issuer should enforce a higher strength password rule, such as using alpha-numeric character with greater than 6 to 8 digit in length.

The Cloud-based OTP server is set to shut off any unsuccessful login after a limited number (e) of trials. The number e is the threshold of login trial that is allowed of an authentication system. It is a tradeoff between a desired network security protection level and its user friendliness policy. Such threshold should be determined by each individual organization to fit its specific application needs.

Certain options can be implemented to prevent hardware token being secretly photo copied. By adopting optical technologies into REAL Cloud-based OTP hardware token making process, we can greatly enhance token's security protection. This can be our future work for further exploration.

4.6.3 Using an insecure public PC

Internet Kiosk or public PC is conveniently available in brokerage firm, bank, hotel, airport, school or conference. Using such public PC or kiosk is becoming popular. When using such a PC to access Cloud-based OTP server, the user's credential, Ciphertext Image and the decrypted OTP code may be captured by such insecure machine. But to replay the login, the malicious people will not be able to decrypt the new Ciphertext Image without the REAL hardware token. The probability (P_3) to successfully find the window pattern (REAL key and OTP code) is 6.9E-19. The detail calculation is as follows. Let's define a few terms before the calculation.

- P_a=Probability of finding RI from the Ciphertext Image,
- P_b=Probability of finding the 6 windows in RI by using the 6-digit OTP code,
- P_c=Probability of finding the other 4 windows in RI by using the 6-digit OTP code,
- P_d=Probability of finding the 7 tagged windows, and
- P_3=Probability of finding the REAL Key by using the 6-digit OTP code.

We then have the probability (P_3) of finding the REAL Key by using the 6-digit OTP code as

$$P_3 = P_a \times P_b \times P_c \times P_d. \tag{29}$$

CI has 11 rows and RI has 5 rows. As the entire RI is embedded in the CI as a whole unit, there is only $7(= 11 - 5 + 1)$ ways of placement. So we have $P_a = 1/7 = 0.14$.

We assume that the symbols are evenly distributed in the REAL Image. Each symbol's distribution is as shown in Table 5. We divide the image into 6 intermingled sections. Each section has certain number of symbol that match to the particular OTP code digit (D_i) as listed in the first row of the Table 5. For the symbol of D_5, its possible location can be determined from the 3 symbols in the first section of the RI. So the probability to find its correct location is 1/3. We then move onto the next code symbol following the assumptions listed in Table 5. P_b can be found by multiplexing each symbol's probability. We then have

$$P_b = (1/3)^2 \times (1/5)^2 \times (1/7)^2 = 9.1E-05. \tag{30}$$

The rest of the 4 locations are all tagged windows as the symbols are ignored during the decryption steps. We have around 22 to 26 (average 24) tagged symbols in a 5-row image. We assume that we need to find the 4 tagged windows from the 24 tagged symbol locations. We further assume that 2 of the tagged symbols were identified as the OTP code symbol. So the probability (P_c) of finding the other 4 windows in rest of the REAL Image is

$$P_c = C((24-2), 4) = 1.4E-04. \tag{31}$$

Since these 4 windows do not show the valid OTP symbols, such 4 windows are "Tagged Window". We still need to indentify the rest of 3 tagged windows from the 6 windows indentified from P_b. The probability (P_d) of finding the other 3 tagged windows from those windows that were found in P_b is related to Dynamic REAL Key scheme that OTP server assigns. Such probability will be affected by the captured RI (with the tagged symbols) and the possible repetition of the symbol in the captured OTP code. To ease our discussion, we can assume a worst case that all the 3 tagged windows can be found directly without any guessing. In other word, P_d is equal to 1. So we then have the probability (P_3) as

$$P_3 = (P_a \times P_b \times P_c \times P_d)/2 = 8.9E-10. \tag{32}$$

Again, the P_3 is much smaller than 1E-06 which is the probability to correctly brute force guess the 6-digit OTOP code. So even in a tough situation of losing the OTP code and CI, REAL Cloud-based OTP token still has far better security protection than a regular token.

To further protect the confidential cookies that retains a user's confidential data, we can adopt the solution in [48] by adding a proxy server as shown in Fig. 23. Authentication service provider should add a check button to allow the user to notify the server during the login procedure that she is using a public PC or kiosk. So the proxy server will hold web cookies on the user's behalf. And the confidential cookies will not be exposed to the insecure PC.

Table 5 Finding the REAL key with a known OTP code and REAL Image

	D_5	D_4	D_3	D_2	D_1	D_0	Total
Possible # of symbols within one divided section	3	5	7	7	5	3	
Probability of finding the correct symbol location	1/3	1/5	1/7	1/7	1/5	1/3	9.1E-05

4.6.4 Security level on certain special codes when under a Trojan attack

Certain Trojan may have the capability of obtaining a user's OTP code and its corresponding Ciphertext Image. This is especially true when using an insecure public PC or Internet kiosk. An intruder may use such information to trace hardware token window pattern (REAL key). We had discussed a similar situation in Subsection 4.6.3. If a known

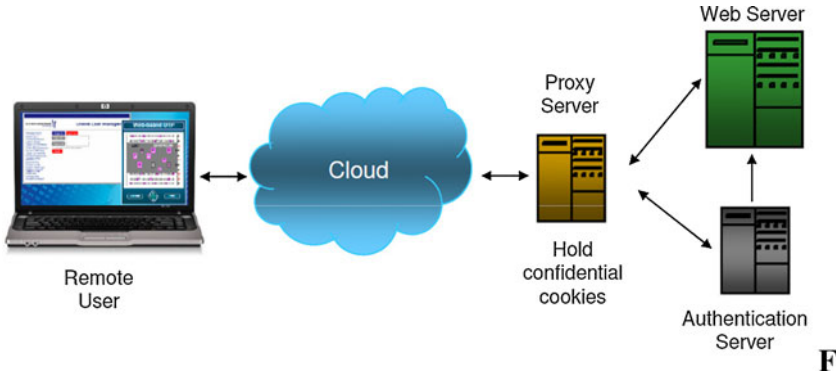


Fig. 23 Cloud-based OTP token using Proxy server

OTP code is made up with all 6 different symbols, the probability (P_3) to correctly trace out the REAL 10-window pattern is about 8.9E-10 as proved in (32). It is much more difficult than a direct brute force guess of the 6 digit OTP code (=1E-06).

Even if the intruder finds the correct 6 windows for one specific OTP code, he still cannot apply this finding for other Ciphertext Image to find a next new OTP code. This is due to that we have total of 10 windows plus the use of Dynamic REAL Key scheme. The server can always opt to use one of the next 69 dynamic session keys by placing the tags on different symbols. Both the tagged window and tagged symbol makes the window tracing much difficult than a straight forward brute force guess of the OTP code.

In an extreme case such as all 6 digits are of the same symbol. The probability to correctly trace out the 6-window pattern from such a captured OTP code plus a known related REAL Image is 2.3E-04. Though there are still 4 more windows and a few tagged windows that the intruder has to find before he can obtain the full REAL key. But easily leaking out even a small portion of the REAL key information (on 6 valid windows) may reduce the Cloud-based OTP Token's security level. It degrades the security strength of the Cloud-based OTP Token and may eventually become an issue. One way to solve such a problem is by not sending out such poor uncertainty and low security level OTP code. The criterion of an acceptable and secure code is not to have more than 3 repetitive symbols in an OTP code. Since the code is generated by OTP server, the server can easily examine and screen such poor code before starting the REAL Image and Ciphertext Image generation.

4.6.5 MITM replay and REAL key tracing attack

REAL Cloud-based OTP Token can easily prevent "MITM Replay" attack with the dynamic OTP code. Another security attack is to intercept many OTP codes and then try to trace the window location (REAL Key). As each REAL hardware token contains two sets of REAL Key, the server can use the two keys in a random manner. So the OTP codes collected by the intruder will not show any formidable information for key tracing. The likelihood of showing the window pattern through the intercepted OTP codes will be broken by the Dynamic REAL Key scheme set by the server. It also increases the security level of REAL Cloud-based OTP Token security level accordingly.

5 Implementing a REAL mobile OTP token

The REAL Mobile OTP Token is a mobile phone based standalone token using REAL as the base cipher. The token is OATH compatible with 6-digit OTP passcode. It uses event-based OTP algorithm (HOTP) [27]. It generates OTP code automatically without the aid of any PC or other Internet device.

Using a mobile phone as the OTP generation platform presents itself certain difficulties. Mobile phone runs on a small battery. Mobile phone CPU is not as powerful as that used in the desktop PC. We need to reduce or eliminate the heavy number crunching operation as much as possible. Fully compatible and interoperable with the existing authentication infrastructure are one of the must have items. The token contains certain secrecies inside the phone. As phone can easily get lost or stolen, securely protecting the secrecies becomes very important as well. So balancing a security level to trade off among the power, CPU and conformance is one of the most critical subjects when implementing a mobile OTP token.

To do so, we do not compute OTP code directly on the phone set. All the OTP codes are pre-generated and encrypted using the low power encryption cipher such as Rubbing Encryption Algorithm (REAL) by OTP server. The encrypted codes is then stored in the designated phone. The user can then rub (decrypt) the OTP code using a REAL hardware token. To address the conformity issue, we use the same OTP algorithm as the existing authentication system.

5.1 Design the REAL mobile OTP hardware token

We choose a plastic card with the size of 1"×2" as hardware token (as shown in Fig. 24). This is the same size of a mini membership card issued by supermarket or franchised book store chain. The material of such token conforms to credit card's ISO/IEC FDIS 7810 standard [19]. The token is thin and small. It can be easily put on key chain or kept in a wallet. The token making process is very similar to a regular credit card. So the cost of this hardware token is comparable to a credit card.

The OTP code pointer is the solid black triangle printed on the token periphery. The REAL encryption key is embedded on the token by the code pointer's locations. Token's periphery is transparent. Overlaid symbols can be clearly read through the token. A barcode serial number is printed on the rear side to correlate the token with a specific REAL encryption keys. Every token carries two sets of REAL key with one on each side. The REAL Image will indicate which set of key to be used during the rubbing (decryption) process. An image alignment line (light color line with code pointer's bases attached) is printed on the token. It helps the alignment between the token and the REAL Image for quick decryption.

5.2 Encryption procedure

For ease of discussion, a REAL Image (X) of size 40 (T) and a set of 10 numerals of 0 to 9 are chosen to illustrate the proposed implementation method.

5.2.1 Key generation

The REAL key is embedded on its hardware token as well. To generate a key, we first design a desired REAL Image (RI) form factor. RI's size should fit well on the phone's screen. The symbol font size should allow for easy reading. In our design (as shown in Fig. 25), the phone screen size is 1.375"×1.875". The RI's size is 40 (T). We use a size 11 bold Arial font for each numeric symbol. For 6-digit (D) of plaintext, we use 7 (= D + 1) code pointers as REAL key (see Fig. 24). The additional one pointer is used as the indicator for choosing the front or back side REAL key.

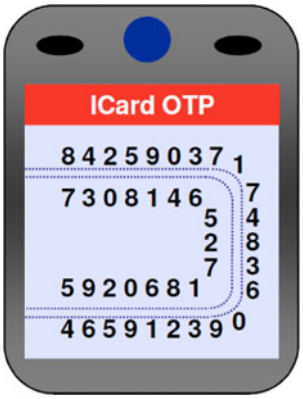


Fig. 25 REAL Image of a Mobile OTP token

5.2.2 Generating REAL image

In the actual provisioning process after a user activates her token, the server generates a series of OTP codes. These codes are encrypted using the user's REAL key (embedded on her hardware token) to generate a series of corresponding REAL Images. We use the following example to illustrate the REAL Image's generation.

Assuming OTP code=807235, we then have D_5=8, D_4=0, D_3=7, D_2=2, D_1=3 and D_0=5 as the plaintext symbols. REAL places the plaintext's symbols into the corresponding W_i locations in REAL Image according to each symbol's

REAL_Image_GEN(K, i)

0 OTP(i) = Truncate(HMAC-SHA-1(K, i));

1 OTP(i) = Concatenate (D_5|D_4| D_3|D_2| D_1|D_0);

2 Sequentially placing D_5 through D_0 into the corresponding W_i locations of the REAL Image;

3 Fill in an odd random number (3 in our example) in W_6 to indicate using key on the front side of the token. If rear key is preferred, an even number will be filled in at W_6 location;

4 Fill the rest of REAL Image elements with randomly chosen symbols so that the Image reaches equiprobable state. //This is REAL_Image(i);

5 Data(i) = Concatenate(elements of REAL_Image(i));

6 End of program.

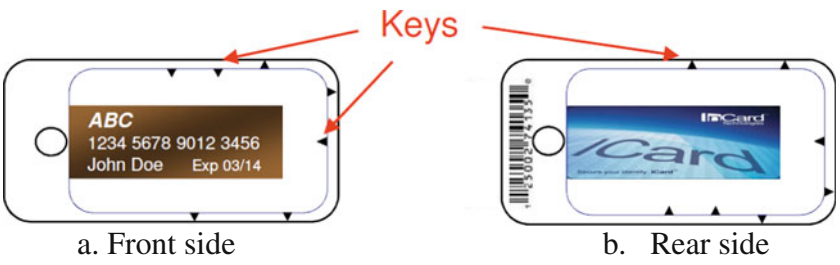


Fig. 24 Mobile OTP token's REAL hardware token. **a**. Front side, **b**. Rear side

Table 6 Encrypting OTP code using REAL

Placement of OTP Code	Key Locations	W_6	W_5	W_4	W_3	W_2	W_1	W_0
	I	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	II	3	8	0	7	2	3	5

occurring sequence. Row I and Row II of Table 6 show how the plaintext symbols are sequentially assigned to W_5 to W_0 locations. Number in W_6 (D_6) is an indicator for choosing front or rear side key during the rubbing (decryption) process. An odd number means to use the front key. An even number will redirect the user to use the rear side key. Rest of the elements in RI is then randomly filled with the numerals so that each numeral appears 4 times in the image. Such a REAL Image is then an equiprobable matrix. It has the highest matrix uncertainty among the size 40 matrices. Figure 25 shows one of such RI examples. The pseudo code to generate a series of REAL Images is shown as follows.

In this pseudo code, K is a 160-bit randomly chosen OTP seed and i is an event counter value. OTP seed K is a user specific random number. The event counter value (i) keeps the sequential number count of the OTP generation. It automatically increases by one count after each OTP generation. The REAL_Image(i) is the encrypted OTP code generated from OATH HOTP formula [28] as shown in step 0. Data(i) is the concatenation of all the elements of the REAL_Image(i) such as shown in Fig. 26. To make up a 5,000 units of OTP codes and REAL Images, we simply run the event counter value (i) from 1 through 5,000 in the above "REAL_Image_GEN" program. We then have 5,000 copies of Data(i) of the REAL Images for further processing before stored into the designated mobile phone. The flow chart of REAL OTP operation if shown in Fig. 27.

5.2.3 Offset generation

To further protect the ciphertext data, we use a random value (Offset) to generate a logic operation difference (Delta) between Data(i) and Offset(i). Offset(i) is generated by a one-way-hashing operation from the (i-1)th index value. This hashed index value is called HI. HI is generated by using the OATH OTP generation formula. It is shown as follows.

$$\mathrm{HI(i)} = \mathrm{HMAC-SHA-1(HI(i-1), 1)}, \tag{33}$$

where i is equal to or greater than 1. And "i" is the event count value. The initial hashed index HI(0) is determined by the following equation.

$$\mathrm{HI(0)} = \mathrm{HMAC-SHA-1(K\_1,\ 1)}, \tag{34}$$

where K_1 is a 160 bit random number chosen by the server.

Once HI(i) is available, Offset(i) can be generated by taking the HMAC-SHA-1 operation to generate two 160 bit data (Fig. 28). The two 160 bit data are concatenated to form the Offset(i). The pseudo code to generate the entire series of Offset(i) is shown as follows.

Offset_GEN

0 i = 0, //initialize program loop counter;

1 HI(0)= Truncate(HMAC-SHA-1(K_1, 1)), //K_1 is a 160 bit random number;

2 Bit_159 to Bit_0 = HMAC-SHA-1(HI(i), 1);

3 Bit_319 to Bit_160 = HMAC-SHA-1(HI(i), 19);

4 Offset(i) = Concatenate(Bit_319 to Bit_0);

5 i = i + 1;

6 HI(i) = Truncate(HMAC-SHA-1(HI(i-1), 1));

7 If i > Max_Count, go to Step 9, //Max_Count = total counts of DATA(i);

8 Go to Step 2

9 End of program.

5.2.4 Delta table and secure storage

Delta(i) is the bit-Exclusive-OR (B-XOR) of the corresponding Data(i) and Offset(i) (Fig. 29). Their relationship is as

$$\mathrm{Delta(i)} = \mathrm{B-XOR(Data(i),\ Offset(i))}. \tag{35}$$

Delta Table (DT) is the compilation of the entire Delta(i) in a special relationship to the corresponding HI (i). That is, Delta(i) is not stored according to the original sequence of Data(i). Delta(i) is rearranged to follow the value significance of its corresponding HI(i). The newly tabulated Delta(i) forms the DT. DT and the last HI data are then further encrypted using the algorithm similar to [26] "Scramble All and Encrypt Small" with the user's personal encryption key (CK). The encrypted data are securely stored in the designated mobile phone (Fig. 30). Personal encryption key (CK) is

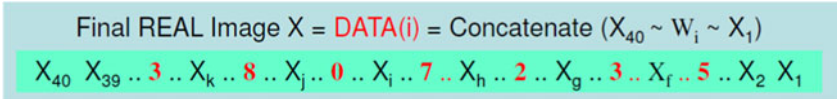


Fig. 26 REAL_Image(i) and Data(i) generation

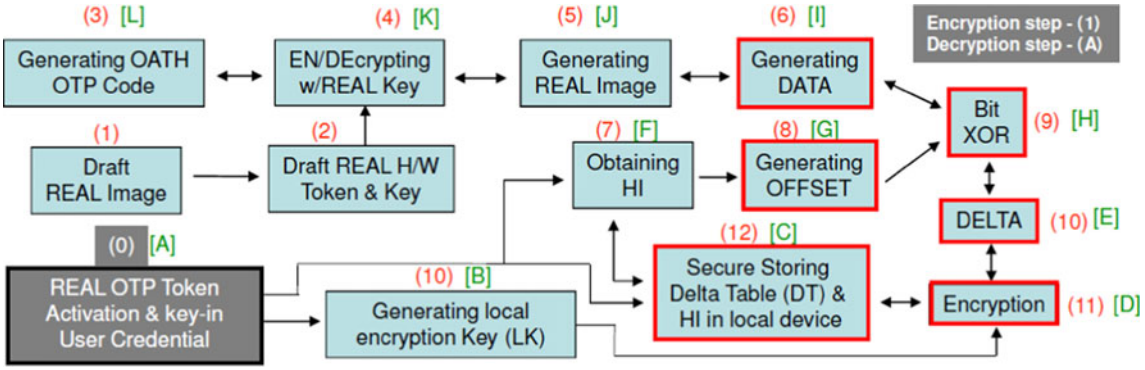


Fig. 27 Operating procedure of REAL mobile OTP token

the concatenation of the userID and Mobile OTP Token password (MOTP).

$$CK = \text{Concatenate}(\text{userID}, \text{MOTP}). \tag{36}$$

5.3 REAL decryption procedure

To decrypt a REAL ciphertext, we mainly follow the procedures of Section 5.2 but in a reversed order. The user activates OTP generation program. She then keys in her userID and password (MOTP). Personal encryption key (CK) is generated to decrypt DT and HI(i-1) stored in the mobile phone. After obtaining the last HI(i-1) and Delta(i) value, a new HI(i) value is generated using (33). Once HI(i) is available, Delta(i) can be found by sorting through Delta Table (DT). Offset(i) is also generated from HI(i). Data(i) is then obtained following (35). Subsequently, REAL_Image (i) is reconfigured from Data(i) and displayed on the mobile phone's screen (Fig. 31).

By properly aligning the unique REAL hardware token on top of the REAL Image(i), the plaintext data is (rubbed out and) indicated by the pointers on the token (as shown in Fig. 31). During the decryption process, the user always starts with front key (front side of the hardware token). The OTP code rubbing sequence starts from the most top left symbol on the outer ring of the REAL Image (RI). The first symbol (N_{1f}) pointed by the front first code pointer determines which key to use.

In Fig. 31, the first symbol (N_{1f}) is 3. It is an odd number. It means that this RI should be decrypted using front side key. The actual OTP code reading begins from the second pointer (pointing to the symbol N_5). Ignoring the first symbol (3) indicated by the first code pointer, we obtain the OTP code as "807235".

In Fig. 32, the first symbol (N_{1f}) through front side of token is 8. It is an even number. It means this RI should be decrypted using rear side key. Ignoring the first symbol (6) indicated by the first code pointer, we obtain the OTP code as "478818" using the rear key (Fig. 32).

5.4 REAL mobile OTP client software and provisioning

REAL Mobile OTP Token has two major parts of software. They are Program File and Data File. These two groups of software will be installed inside a user designated mobile phone. The Program File contains a set of executable

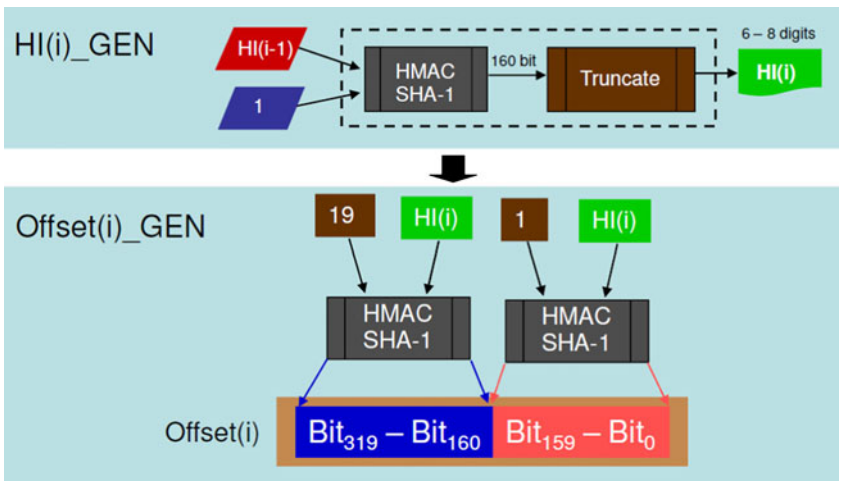
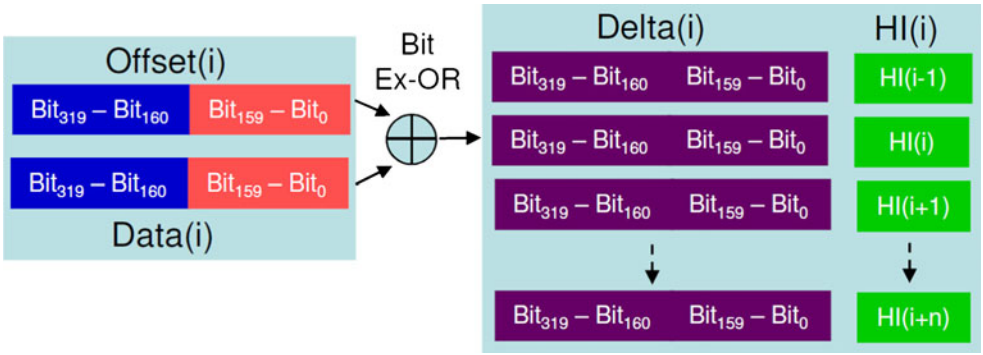


Fig. 28 Generating hashed index and offset

Fig. 29 Delta(i) generating



programs to generate the REAL Image and other housekeeping tasks. The list of Program File includes the following programs.

- OATH HOTP Code Generation program
- HMAC-SHA-1 program
- REAL Image Generation program
- End of session housekeeping program
- Help Menu with Demo program and Version Number

The provisioning work is done as follows. After receiving the assigned REAL hardware token, the user calls the server to initiate the OTP provisioning work. Server asks for the user's credential and token's serial number. It then associates the token's REAL key with the user's credential. The OTP server generates a series of 5,000 OTP codes and randomly uses the user's REAL keys (2 keys per token) to encrypt the codes. Subsequently, the server generates the DT and HI and encrypted them with user's personal key (CK) to form the Data File. The encrypted Data File and Program File are zipped together at the end of the provisioning work. The zipped file can then be downloaded into the user's mobile phone through a secured Internet connection. After the auto-installation, both the Program File and the encrypted confidential Data File are securely stored in the mobile phone and ready for use.

5.5 Security analysis of the REAL mobile OTP token

5.5.1 Design goals review

We use same low cost material like credit card for hardware token. Their manufacturing process is very similar also. No additional electronic component is used. The hardware token manufacturing cost is comparable to the credit card (less than US$0.50 each).

For a 5,000 units of OTP code stored in a REAL Delta Table, its code size is about 200 KBytes. It is sufficient for a consecutive 2.7 years use with an average daily usage of 5 OTP codes. The entire software code size of REAL Mobile OTP Token is about 3.5 MBytes. The entire program and OTP codes can be easily stored in a mobile phone or micro-SD (Secure Digit) card that fits in phone's memory slot.

The token uses the same OATH OTP algorithm to generate the OTP codes. It ensures 100% compliance to any OATH authentication server. Using the same key (K) and counter value (C), REAL Mobile OTP Token can maintain full interoperability with any OATH compatible token. The token can directly replace any existing or expired OATH OTP token for cost saving.

The deployment and technical support can be easily done through Internet. The token's distribution and activation methods can follow the secure logistic practice that the credit card company is doing. This help to save the deployment and support cost. There is no need to change or add any hardware or software on the existing OATH authentication server. Overall, it maintains an easy deployment and low cost operation.

REAL encryption is a fast and simple secure scramble cipher. It does not use complicated mathematical algorithm. It does not demand very fast CPU power either. So it can work easily with the less powerful CPU used in the mobile phone. In fact, this is one of the important features that REAL posses.

5.5.2 Security of OTP code and REAL image

REAL does not alter or replace any of the OTP code during the encryption or decryption process. So the REAL decrypted OTP code is as secure as the one generated by

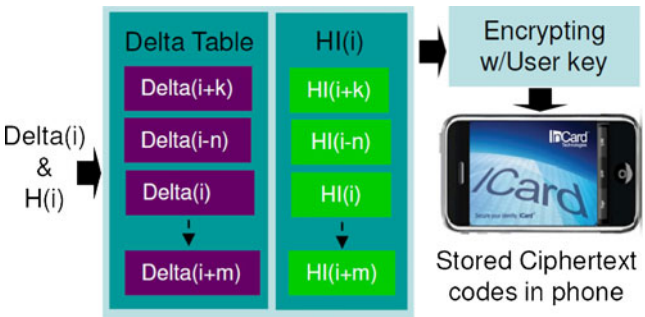


Fig. 30 Generating Delta_Table (DT) and storing the DT inside mobile phone

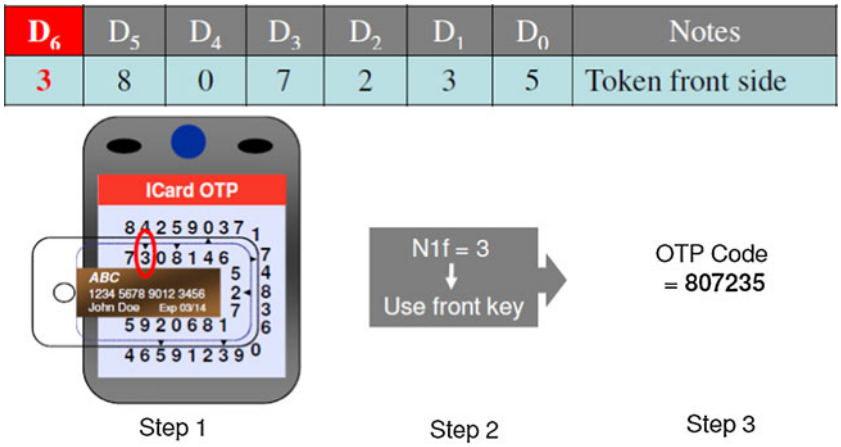


Fig. 31 Rubbing (decrypting) Mobile OTP token using front key

an OATH compatible token. In mobile OTP token, the REAL encrypted OTP codes are further encrypted using a personal encryption key (CK). It prevents these secret data from being easily tampered. They provide the desired level of integrity and security protection.

REAL Image (RI) contains the OTP code. Its security level affects how secure the OTP code is protected. Given an image size of 40 (T) and a 6-digit (D) OTP code,

$$\text{Total number of REAL key} = C(40,(6+1)) \times 2 = 3.72E+07. \quad (37)$$

Without the aid of either a hardware token or a known OTP code, to correctly guess the REAL key from a RI, the probability (P_1) is

$$\text{Probability}(P_1) = 1/(C(40,(6+1)) \times 2) = 2.7E-08. \quad (38)$$

It is tougher than a straight brute force guess of the 6-digit OTP code (the probability is 1E-06).

Given a known REAL Image, to directly guess the correct 6-digit OTP code the probability (P_2) is

$$\text{Probability}(P_2) = 1/C(40,6) = 2.6E-07. \quad (39)$$

This probability is again smaller than a straight brute force guess of the 6-digit OTP code. So even when a REAL Image is known to a malicious person, without the hardware token, a Mobile OTP token still stands very secure in such adverse situation.

On the other hand, a traditional OATH token display the full OTP code on the screen without any encryption. The probability to obtain the correct code from the image on the LCD screen is 1 out of 1.

5.5.3 Cellular phone is lost, stolen or confidential file is secretly copied

Cellular phone is small and prone to get lost, stolen or its confidential file is secretly copied. If such phone or file is at a malicious person hand, he has to crack the user's credential first before activating the OTP function. Even if the cellular phone is activated, trying to correctly guess the OTP code with a known REAL Image but without the specific hardware token, the probability (P_2) is 2.6E-07. To find the REAL key on this phone, the probability (P_1) is 2.7E-08. Both jobs are tougher than a brute force guess of the 6-digit OTP code (probability=1E-06).

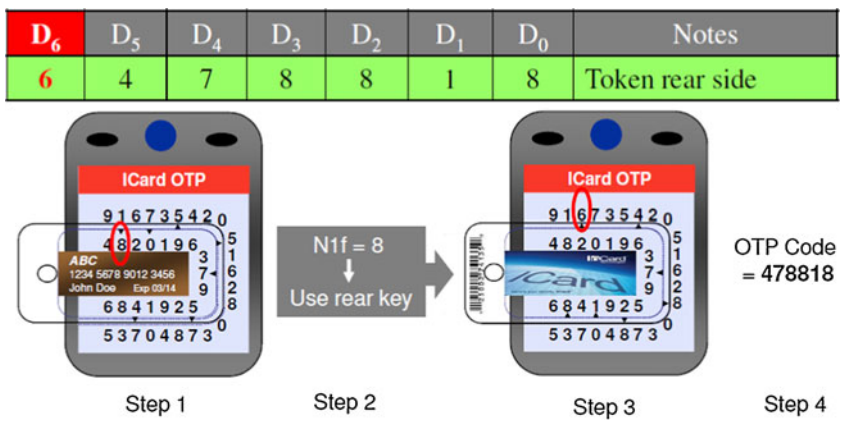


Fig. 32 Rubbing (decrypting) Mobile OTP token using rear key

In some extreme cases when the OTP code has many repetitive symbols, the probability (P_1) can be high. The REAL Image can be readjusted to make the repetitive symbols' uncertainty be as high as possible (i.e. increase such symbol's occurrence while reduce other symbol occurrence). For example, an OTP code of "111111" can be securely hided inside a REAL Image with all the elements using numeral 1.

5.5.4 Hardware token is lost, stolen or secretly copied

REAL hardware token does not contain any electronic to generate OTP code. The token will only work with the specific cell phone that has the user's Data File for OTP generation. The loss of a REAL hardware token or the pointer pattern being photo copied, the security will not be compromised. Just having a hardware token, the probability (P_3) to correctly guess the OTP code is same as to brute force guess the 6 full digits. The probability (P_3) is 1 out of 10^6.

5.5.5 Both confidential data and hardware token are secretly copied

A malicious person may secretly copy the confidential Data File and hardware token without the user's knowledge. Such person can set up a phone with the user's confidential Data File. The last protection that a Mobile OTP Token has is the user's credential. This is the one that includes userID and password to activate the Mobile OTP function on the phone. In general, server should ask the user to use a complex password. So there will be enough security protection when such adversary situation happens.

6 Security attack safe REAL OTP tokens

In this Section, we discuss and analyze certain adversary security attacks beyond the traditional MITM Replay attack. Two such attacks – MITM Seed-tracing Attack and Shoulder-surfing Attack, are discussed and analyzed. We then implement certain OTP tokens that can resist such unusual attacks using REAL as the base cipher.

6.1 OTP token that resists MITM seed-tracing attack

Seed-tracing is one form of the direct attack on an encryption algorithm (Fig. 33). It can be performed in many different ways. For OTP token, the encryption algorithm and its implementation scheme must be well disclosed to the public. The attacker can take such disclosures and use certain mathematical mean to reduce the complexity of the algorithm. He then uses certain captured ciphertext to reversely trace the encryption key (seed). Some may even take certain pre-generated ciphertext (using the same algorithm and a known key) and compare to the captured ciphertext. Through these efforts, he then tries to find the pseudo random sequence of the captured samples. The secret encryption key (seed) may become reversible and eventually be found. A traditional OTP token does not have any built-in capability to effectively resist such attack.

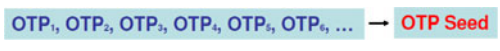


Fig. 33 Tracing the seed by using OTP codes' pseudo random sequence

An improved REAL OTP Token can provide extra protection when MITM Seed-tracing attack happens. We name this version a Multi-seeding OTP Token (also called "Ms. OTP Token"). Figure 34 through Fig. 36 show one such example implemented in a REAL Mobile OTP Token. Same technique can be easily implemented in REAL Cloud-based OTP Token as well.

In our examples shown in Section 4 and Section 5, each REAL hardware token has two sets of REAL encryption keys. If a REAL key is used for one token, each REAL hardware token actually can operate as two OTP tokens. The user can use one of the tokens in a mixed randomly order determined by the OTP provisioning server.

Figure 34 shows such mixed random sequence generated by provisioning server. OTP_{Ai} and OTP_{Bi} mean the i'th OTP code generated by using seed A & seed B respectively. The OTP codes generated by seed A (K_A) is encrypted by the REAL key (RK_A) on front side of token. OTP code from Seed B (K_B) uses the REAL key (RK_B) on the rear side to encrypt and decrypt. So even though the user carries only one REAL OTP hardware token, she actually has two tokens (Token A and Token B) with herself all the time.

The first symbol (N_{1f}) pointed by the first pointer on the front side is used as the indicator to select token A or token B (front or rear REAL key). In Fig. 35, the N_{1f} is an odd number 3. So we use token A (front key - RK_A) to decrypt the OTP code. By following the normal REAL rubbing sequence, the OTP code is "807235". In Fig. 36, the N_{1f} is an even number 8. So we use token B (rear key - RK_B) to decrypt the OTP code. And the OTP code is "478818".

The authentication server knows the mixed OTP code sequence as the code provisioning work is done by OTP server itself. The mixing of two pseudo random sequence OTP code makes a better randomness on OTP codes. The

Fig. 34 Operation of Multiple-seeding OTP token

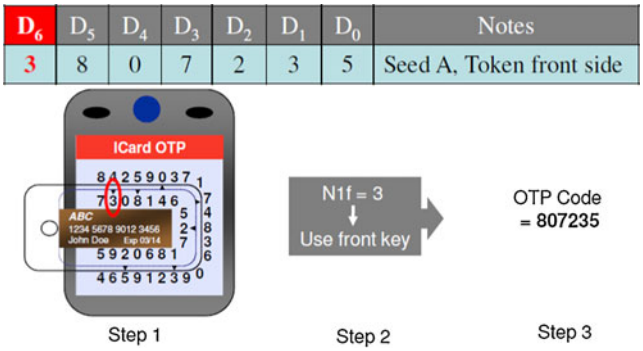


Fig. 35 Multiple-seeding OTP token (Ms. OTP Token) using front key to rub the seed-A OTP code

MITM intercepted OTP codes cannot show any viable pseudo random sequence data. It makes random number seed tracing very difficult. Such MITM Seed-tracing attack is then prevented.

6.2 OTP token that resists shoulder-surfing attack

Shoulder-surfing attack happens when a malicious person secretly observes the action while a user logins a destined server. The malicious person may then obtain the user credential information to trace the login secrecies or OTP code. Such attack does not involve with any mathematical cryptanalysis at the beginning. It is more like a social theft and can be carried out in a secret manner without the user's attention or knowledge. Once the secret information is collected, a cryptanalysis can be launched to find the next OTP code or trace the seed if many OTP codes are captured.

Such attack does not happen in the electronic network. It does not make any noticeable noise either. So it is very difficult for a user to prevent such attack. Almost all of the prevailing OTP tokens do not have any Shoulder-surfing prevention capability. It does not even have any built-in attack resistance mechanism other than asking the user to take cautious step when operating the OTP token.

To fend off Shoulder-surfing attack, we use a random offset concept in our REAL OTP token. The idea is to decouple the direct relationship between the code pointer location (REAL key) and the OTP code symbol. So the intruder can no longer find any meaningful data from the information captured through the Shoulder-surfing attack. The token with such feature is called Multi-random OTP Token (also called "Mr. OTP Token"). Figures 37 and 38 show an implementation example using a REAL Mobile OTP Token. Again, same technique can be used with the REAL Cloud-based OTP Token discussed in Section 4 as well.

The first symbol (N_{1f}) pointed by the first front pointer is still used as the indicator to select front or rear side's key. But both N_{1f} and N_{1r} will be used as a value added to each symbol's numeric value pointed by the rest of the six code pointers. The ten's digit will be dropped if the added value is greater than or equal to 10. The general equation is as follows.

$$D_{if} = \left(\text{Value of } N_{1f} + \text{Value of } N_{(7-i)f}\right) \mod 10, \quad (40)$$

$$D_{if} = \left(\text{Value of } N_{1r} + \text{Value of } N_{(7-i)r}\right) \mod 10, \quad (41)$$

where D_{if} is the ith digit of OTP code using front REAL key, $N_{(7\text{-}i)\ f}$ is the symbol pointed by the (7-i)th pointer using front REAL key and "mod 10" is modulo 10 operation. When making the REAL Image, each $N_{(7\text{-}i)\ f}$ or $N_{(7\text{-}i)\ r}$ value should be considered with the original OTP code according to (40) and (41).

In Fig. 37, the first symbol (N_{1f}) pointed by the first front code pointer is equal to 3. The N_{1f} is an odd number. We use the key on front side of hardware token to rub the

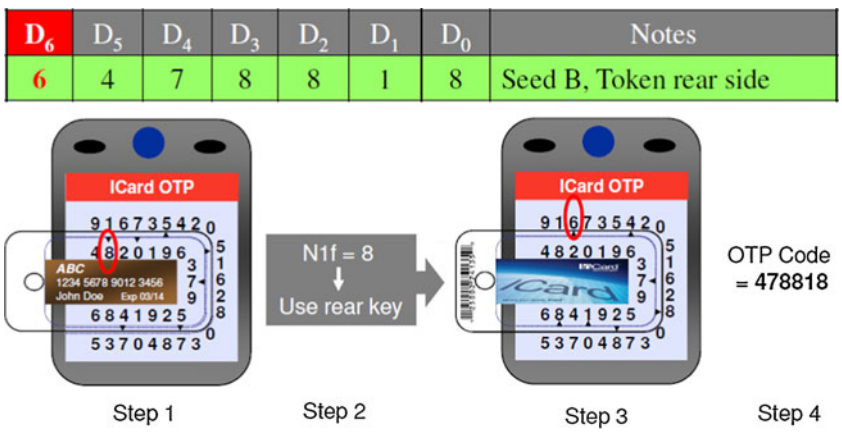


Fig. 36 Multiple-seeding OTP token (Ms. OTP Token) using rear key to rub the seed-B OTP code

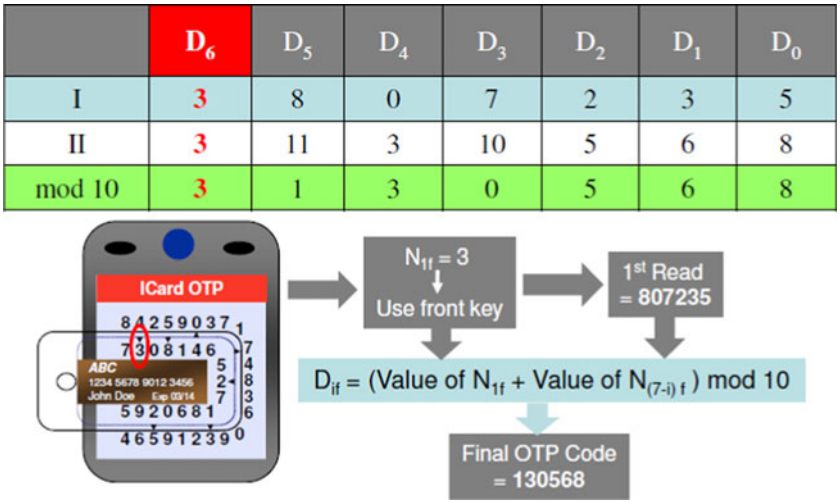


Fig. 37 Multiple-random OTP token (Mr. OTP Token)

OTP code. The second front code pointer indicates N_{2f}=8. From (40), we find

$$D_{5f} = (3 + 8) \mod 10 = 1. \quad (42)$$

The table in Fig. 37 shows the entire OTP code decryption procedure. Symbols pointed by the code pointers are read first. It is "807235" with D_6 equals 3. They are recorded in row I of this table. After adding D_6's value (3), the result is shown in the II row. We then drop the ten's digit (taking the mod 10 operation). The III row shows the final result which is the valid OTP code. Following this new decryption procedure, we have the full OTP code as "130568".

The REAL Image in Fig. 38 shows N_{1f}=8, an even number. We use REAL key on the rear side to decrypt the OTP code. The first pointer on the rear side shows a number N_{1r}=6. We find N_{2r}=4. We can find D_{5r} through (43).

$$D_{5r} = (6 + 4) \mod 10 = 0. \quad (43)$$

Following such procedure, we obtain the full OTP Code as "034474".

Both N_{1f} and N_{1r} are randomly chosen by provisioning server. The modular operation also adds additional randomness to the codes. So even if a malicious person sees the code pointer location by Shoulder-surfing attack, the pointed symbols and the actual OTP codes do not show any direct one-to-one relationship. In fact, they do not even relate to each other in the next run of OTP code. Such token then protects the code security and resists the attack from the troublesome Shoulder-surfing technique.

6.3 Security attack safe REAL OTP token

An OTP token that resists both MITM Seed-tracing and Shoulder-surfing attacks can now be realized by combining both Multi-seeding and Multi-random techniques. By using Multi-seeding technique, we can defend the MITM Seed-tracing attack. Using Multi-random technique, we have the capability to fend off Shoulder-surfing attack. Both techniques require no assistance from the remote PC or additional electronic on hardware token. In fact, the same basic hardware token is used without any further change.

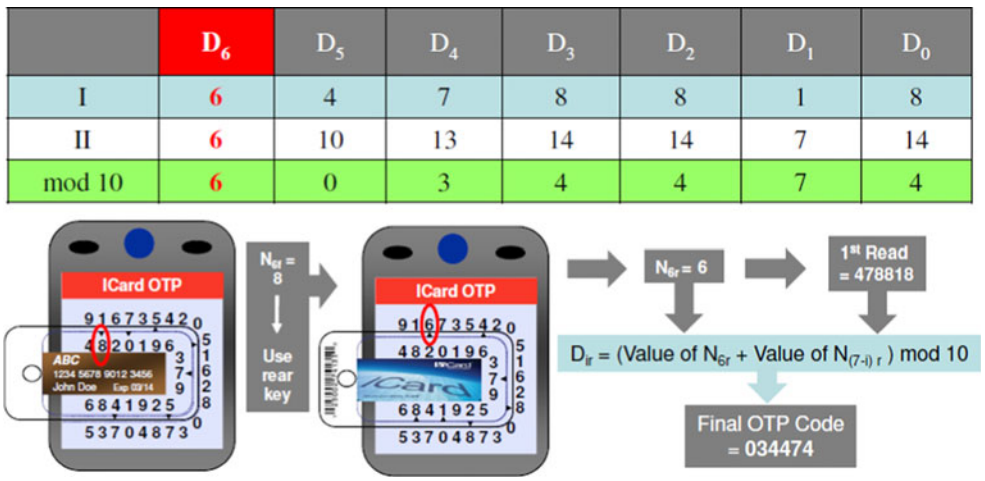
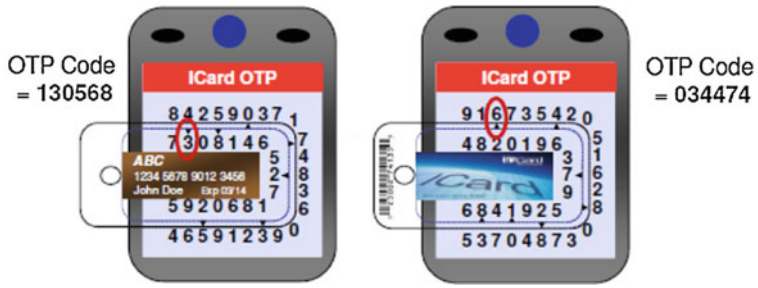


Fig. 38 Multiple-random OTP token (Mr.OTP Token using rear key)

REAL Secure Mobile OTP Token = Ms.OTP + Mr.OTP

	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Notes
A1	3	8	0	7	2	3	5	Seed A, front side, Ms. OTP
A2	3	1	3	0	5	6	8	OTP = Ms. OTP + Mr. OTP
B1	6	4	7	8	8	1	8	Seed B, rear side, Ms. OTP
B2	6	0	3	4	4	7	4	OTP = Ms. OTP + Mr. OTP



a. Using front key b. Using rear key

Fig. 39 A security attack safe REAL OTP token. **a**. Using front key, **b**. Using rear key

Again, though the example shown is with a Mobile OTP token, the technique can actually be implemented with other type of REAL OTP tokens. This is one of the versatile examples on using REAL's multiple dimensional keys for better security protection. This feature enables us to have better protection on network security as well as OTP token itself. Figure 39 shows the combined operation of the Security Attack Safe REAL OTP Token.

7 Limitations of the OTP token and two-factor authentication

MITM attack has constantly advanced its own capability with new techniques. One of most recent developments is that it can work with some Trojan program and silently resides in a PC. Such program has the capability to execute malicious actions without any user's knowledge. In some case, it may redirect the user to a fraudulent web site to collect a user's data, and then redirects the network connection back to the genuine web site after the data collection. In some extreme case, it may imitate a user from user's own PC to interact with a genuine web site. Such MITM and Trojan attacks present new challenges to entire network and data security research institutes and industry. The 2FA system and OTP token alone cannot solve these issues [35]. It will require layered security schemes, more security tools and strict disciplines to make up a better protecting environment. Variety of REAL OTP Tokens can then become one of the key components to make up such a secure system solution.

8 Conclusion

One-time Password (OTP) token has the capability to automatically generate a dynamic session password. It is a leading password technology in today's Two-factor

Table 7 Notations

Symbol	Notation
AND	Logic AND operation
C(n, k)	Calculation of the k-combination in a set with size n
L!	$L! = L \times (L-1) \times \ldots \times 2 \times 1$
mod X	Modulo operation based on X
2FA	Two-factor authentication
Ciphertext Image	A larger size ciphertext contains REAL Image
Data	Bit representation of REAL or Ciphertext Image
Delta	Delta=bit-exclusive-OR (Offset, Data)
DT	Delta Table – a compilation of the series of Delta(i)
HI	Hashed Index
HOTP	OATH Event-based OTP algorithm
MITM attack	Man-in-the-Middle security attack
OATH	Initiative for Open AuTHentication
OCRA	OATH Challenge-response-based OTP algorithm
Offset	An intermediate value generated from HI
OTP	One-time password
REAL	Rubbing Encryption ALgorithm
REAL Image	REAL encrypted ciphertext
TOTP	OATH Time-based OTP algorithm

Authentication (2FA) System. But it has certain deficiencies such as high token cost, difficult for carrying, high deployment and support expenses. In particular, many of the implementations may comprise the security when token is lost, stolen or secretly copied.

We present a novel Rubbing Encryption Algorithm (REAL) as the base cipher for a new token. REAL has its unique strength in securely encrypting the short word length data such as an OTP code. REAL uses a secure and fast scramble algorithm to hide the OTP numeric code in a large equiprobable matrix. Such matrix has the highest uncertainty among the same size of matrices. REAL encrypted data (also called REAL Image or Ciphertext Image) can be securely transmitted to and displayed in a remote PC or even a public Internet Kiosk. Furthermore, REAL decryption does not require the entering of an encryption key. So the length and complexity of a REAL encryption key is no longer limited by a user's poor memory capability. These REAL key's niche features allow the use of a much higher complex key.

Credit card like REAL hardware token is inexpensive and easy to carry. The REAL encryption key is embedded on the hardware token. A user uses such inexpensive non-electronic hardware token to "rub" (decrypt) the OTP code from a Ciphertext Image shown on PC screen. Two OATH compliant REAL OTP tokens were implemented and analyzed. One is a Cloud-based OTP Token. The other is a Mobile OTP token. Both tokens have better security level than the traditional hardware or software tokens. Moreover, they can also resist MITM Seed-tracing and Shoulder-surfing security attacks. This is the special feature that cannot be found in the traditional tokens.

REAL can be implemented in applications with multiple dimensional form factor. The two examples shown use multiple encryption keys embedded in a two dimensional hardware token. This feature provides the needed enabling technique to make the OTP token more secure and can resist certain security attack. Such feature certainly has potential for other applications. REAL can also be further enhanced on its security strength. These are the work for our further exploration.

Table 7 lists certain notations and symbols used in this paper. Please refer to this table for the meaning of each symbol, notation and term used in this paper.

Acknowledgement We will like to thank Innovative Card Technologies, Inc., Dr. Paul Wu of FPC Consultancy, Dr. Carl Chang, Dr. Ying Cai of Computer Science Department of Iowa State University and Dr. CK Yuan for their help and guidance on algorithm development and OTP tokens implementation.

References

1. Abe T, Itosh H, Takahashi K (2007) Implementing Identity Provider on Mobile Phone. In *Proceedings of the DIM 2007 Conference.* Nov. 2, 2007
2. Aladdin (2009) eToken One-time Password (OTP). Nov. 2009. http://www.aladdin.com.etoken/otp.aspx
3. Aloul F, Zahidi S, El-Hajj W (2009) Two Factor Authentication Using Mobile Phones. In *Proceedings of the 2009 IEEE/ACS International Conference on Computer Systems and Applications*. 2009
4. Cheng F (2010) A Novel Rubbing Encryption Algorithm and The Implementation of a Web-based One-time Password Token. In *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference*. July 22, 2010, Seoul, S. Korea. doi:10.1109/COMPSAC.2010.21
5. Cheng F (2010) A secure mobile OTP Token. In *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2010. Volume 48, Part 1, 3 – 16.* doi:10.1007/978-3-642-17758-3_1
6. Deepnet Security (2010) MobileID - A Mobile, Two-way and Two-factor Authentication. http://www.deepnetsecurity.com/products2/MobileID.asp
7. Eastlake D 3rd, Jones P (2001) RFC3174 –US Secure Hash Algorithm 1 (SHA1). *Network Working Group. The Internet Society (2001).* September 2001
8. Entrust (2010) Entrust IdentityGuard Token Quantity One: $5. April 29, 2010. http://www.entrust.com/strong-authentication/identityguard/tokens/
9. Florencio D, Herley C (2006) KLASSP: Entering Passwords on a Spyware Infected Machine Using a Shared Secret Proxy. In *Proceedings of ACSAC 2006*. July, 2009
10. Foster K, Meijer E, Schuh S, Zabek M. The 2008 Survey of Consumer Payment Choice. 2010. *Federal Reserve Bank of Boston, Public Policy Discussion Paper No. 09-10.* April, 2010
11. FreeAuthProject (2010) The FreeAuth Project, http://www.freeauth.org/site
12. GRC (2009) Flexible One-time Password MetaSystem, Perfect Paper Passwords. Nov. 2009. http://www.grc.com/ppp.htm
13. Hallsteinsen S, Jorstad I, Thanh D (2007) Using the mobile phone as s security token for unified authentication, In *Proceedings of the ICSNC 2007 Conference. IEEE Computer Society.* 2007
14. Haverinen H, Asokan N, Maattanen T (2001) Authentication and Key Generation for Mobile IP Using GSM Authentication and Roaming, In *Proceedings of the ICC 2001 Conference*. 2001
15. Information Technology Laboratory, National Institute of Standards and Technology. Secure Hash Standard. 1995. *Federal Information Processing Standards Publication, FIPS PUB 180-1.* April 17, 1995
16. Information Technology Laboratory, National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). 2002. *Federal Information Processing Standards Publication, FIPS PUB 198.* March 6, 2002
17. Initiative for Open AuTHentication (2009) Oath Vision (2009) http://www.openauthentication.org/about
18. Interactive Brokers (2009) Login with security Code (Bingo) Card. Nov. 2009. http://ibkb.interactivebrokers.com/node/1042
19. International Organization for Standardization (2003) *ISO/IEC 7810:2003*. http://iso.org/iso/catalogue_detail?csnumber=31432
20. Kostiainen K, Ekberg JE, Asokan N (2009) On-board Credentials with Open Provisioning. *ASIACCS'09*, Mar. 2009
21. Lamport L (1981) Password authentication with insecure communication. Commun ACM 24(11):770–772
22. Liao K, Sung M, Lee W, Lin T (2009) A one-time password scheme with QR-Code based on mobile phone. In *Proceedings of*

the INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference. 2009. doi:10.1109/NCM.2009.324
23. Liao S, Zhang Q, Chen C, Dai Y (2009) A unidirectional one-time password authentication scheme without counter desynchronization. In *Proceedings of the CCCM 2009 Conference*, September 29, 2009
24. Liberty Alliance (2010) Liberty alliance project. http://www.projectliberty.org/
25. Markus J, Stern J, Yung M (1999) Scramble All, Encrypt Small. In *Lecture Notes in Computer Sciences, 1999, Volume 1636/1999, 95-111.* doi:1007/3-540-48519-8.8
26. Mizuno S, Yamada K, Takahashi K (2005) Authentication Using Multiple Communication Channels. In *Proceedings of the DIM 2005 Conference.* Nov. 11, 2005
27. M'Raihi D, Bellare M, Hoornaert F, Naccache D, Ranen O (2005) *HOTP: An HMAC-Based One-time Password Algorithm. The Internet Society, Network Working Group. RFC4226*. Dec. 2005
28. M'Raihi D, Machani S, Pei M, Rydell J (2010) *TOTP: Time-based One-time Password Algorithm draft-mraihi-totp-timebased-o5.txt. The Internet Society, Network Working Group,* April, 2010
29. M'Raihi J, Rydell M, Naccache D, Machani S, Bajaj S (2010) *OCRA: OATH Challenge-Response Algorithm. Variants 12. The Internet Society, Network Working Group. Internet Draft*. Sep. 2010. http://tools.ietf.org/pdf/draft-mraihi-mutual-oath-hotp-variants-12.pdf
30. National Institute of Standards and Technology, Physics Laboratory, Time & Frequency Division (2009) Set Your Computer Clock Via the Internet – NIST Internet Time Service (ITS). Nov. 18, 2009. http://tf.nist.gov/service/its.htm
31. Pashalidis A (2009) Accessing password-protected resources without the password. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 04, 66–70.* doi:10.1109/CSIE.2009.910
32. Rescola E (2000) *HTTP Over TLS. The Internet Society, Network Working Group. RFC2818*, May, 2000
33. RSA (2009) RSA SecureID. Nov. 2009. http://www.node.aspx?id=1156
34. RSA. RSA SecureID, Software Authenticator. http://www.rsa.com/node.aspx?id=1313
35. Schneier B (2005) The failure of two-factor authentication. *Communications of the ACM*, Apr. 2005
36. SECUREHQ (2010) SecurID SID820 Soft Tokens, April 29, 2010. http://www.securehq.com/group.wml&groupid=708&deptid=80
37. Shannon C (1948) A mathematical theory of communication. The Bell System Technical Journal 27:379–423, 623-659
38. Shastri A, Govil R (2001) Optimal discrete entropy. Appl Math E-Notes 1:73–76
39. Stinson D (1995) Cryptography – Theory and Practice. CRC Press, Inc., Boca Raton, pp 44–67
40. Thanh D, Jonvik T, Thuan D, Jorstad I (2006) Enhancing internet service security using GSM SIM Authentication, In *Proceedings of the IEEE GLOBECOM 2006 Conference*
41. Thanh D, Jonvik T, Thuan D, Jorstad I (2009) Strong authentication with mobile phone as security token. In *Proceedings of the IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. doi:10.1109/MOBHOC.2009.5336918
42. Thanh D, Jonvik T, Feng B, Thuan D, Jorstad I (2008) Simple strong authentication for internet applications using mobile phones. In *Proceedings of the IEEE GLOBECOM 2008 Conference.* 2008
43. U.S. Federal Financial Institutions Examination Councils. (2005) *FFIEC Release Guidance on Authentication in an Internet Banking Environment*
44. Verisign (2010) *Compare Two-Factor Authentication Credentials*, Dec. 30, 2010. http://www.verisign.com/authentication/two-factor-authentication/compare-two-factor-authentication/index.html
45. Wangensteen A, Lunde L, Jorstad I, Thanh D (2006) A generic authentication system based on SIM, *The International Conference on Internet Surveillance and Protection (ICISP'06) 2006*
46. Wikipedia. Two-factor Authentication. In *the Challenges section. October 10, 2010.* http://en.wikipidia.org/wiki/two-factor_authentication
47. Woolsey B, Schulz M. Credit Card Statistics, Industry Facts, Debt Statistics. 2010. *CreditCards.com.* November 2, 2010. http://www.creditcards.com/credit-card-news/credit-card-industry-facts-personal-debt-statistics-1276.php#Circulation-debit
48. Wu M, Garfinkel S, Miller R (2004) Secure web authentication with mobile phones. *DIMAC Workshop on Usable Privacy and Security Software*