University of New Orleans

ScholarWorks@UNO

University of New Orleans Theses and Dissertations

Dissertations and Theses

5-20-2011

One Time Password Scheme Via Secret Sharing Techniques

Christopher Miceli
University of New Orleans

Follow this and additional works at: https://scholarworks.uno.edu/td

Recommended Citation

Miceli, Christopher, "One Time Password Scheme Via Secret Sharing Techniques" (2011). *University of New Orleans Theses and Dissertations*. 1330.
https://scholarworks.uno.edu/td/1330

This Thesis is protected by copyright and/or related rights. It has been brought to you by ScholarWorks@UNO with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in University of New Orleans Theses and Dissertations by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.

One Time Password Scheme Via Secret Sharing Techniques

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Information Assurance

by

Christopher Miceli

B.S. Louisiana State University, 2009

May, 2011

© 2011, Christopher Miceli

Acknowledgments

This work would not have been possible without the guidance of Dr. Golden Richard and Dr. Kenneth Holladay who promoted my ideas and pushed my curiosity. I would also like to thank the members of my committee for their proofreading and other contributions: Dr. Daniel Bilar and Dr. Jaime Niño. The computer science departments at both The University of New Orleans and Louisiana State University provided a solid foundation of education that promoted this research. Specifically, I would like to thank Dr. Hartmut Kaiser, for his profound mentoring abilities and Dr. Shantenu Jha, for his ability to push me to research.

Table of Contents

Abstract

Many organizations today are seeking to improve security by implementing multi-factor authentication, i.e. authentication requiring more than one independent mechanism to prove one's identity. One-time passwords in the form of hardware tokens in combination with conventional passwords have emerged as the predominant means in high security environments to satisfy the independent identification criteria for strong authentication. However, current popular public one-time passwords solutions such as HOTP, mOTP, TOTP, and S/Key depend on the computational complexity of breaking encryption or hash functions for security. This thesis will present an efficient and information-theoretically secure one-time password system called Shamir-OTP that is based upon secret sharing techniques.

Shamir secret sharing, Proactive secret sharing, Verifiable secret sharing, one-time password, multi-factor authentication, password

Introduction

This thesis's goal is to introduce a one-time password system that has been derived from techniques initially developed to implement secret sharing. The advantages and disadvantages of such a system will be described and compared to other current password systems. An appendix is included that describes multiple implementations and how to use them.

Background

Before being presented, a background of prerequisite topics will be given independently and then combined into a functional password system.

One Time Passwords

One-time passwords are passwords that are only valid for a single or small number of transactions. This contrasts with conventional passwords which are valid for many transactions as users are reluctant to voluntarily change passwords frequently. Since OTPs are only valid for a limited number of uses, an attacker has a smaller window of time to gain access to resources guarded by such a password because any previously stolen passwords will likely have become invalid. As with traditional passwords, one-time passwords are vulnerable to man-in-the-middle attacks [1]. By observing the OTP before it is successfully received by the authenticator, an attacker has a valid password. Because of this undesirable property, both OTPs and conventional passwords must travel securely.

Typically, the one-time password is generated by a hardware device that the person desiring to be authenticated carries to promote use across many physically distant domains. The hardware implements an algorithm that generates passwords in a specific manner that the authenticator knows. The hardware device will often display the password on a small screen for a user to type into the authenticator.

In this hardware based approach, if the hardware or computer that generates the passwords were stolen, the thief would be able to authenticate himself just by reading the numbers on the display.

Because of this reason, one-time passwords are often one part of a multi-factor authentication (MFA) system where two or more independent factors of authentication are used to identify a user. The three factors consistently considered possible for authentication are something the user knows, something the user has, and something the user is. Others methods such as someone you know have also been suggested [2]. The static password satisfies the something you know factor. Being able to enter a valid OTP implies that you have the password generator; the something you uniquely have factor is satisfied. Implementing this strategy alleviates the fear of a lost password generator falsely authenticating a malicious entity.

With the increase in the popularity of mobile cellular phones and personal digital assistants (PDAs), there have been some attempts to use these devices as a physical OTP generating device. In this case, the phone or PDA will satisfy the something you uniquely have factor of authentication and the password it generates shows this. Initially, when these devices were limited in capabilities, the OTP would be sent to the device as a text-message that the user would then be able input into the system for authentication. Electronically authenticating for the bank JPMorgan Chase uses this approach along with a traditional password [3]. As more capable mobile operating systems and devices began to appear, most modern mobile phones and PDAs are able to generate the OTP themselves and can completely replace a dedicated hardware device or a text-messaging solution. Recently Google has been utilizing this approach to enable MFA for all accounts that opt-in [4].

Algorithms generating temporary passwords can be time-based or mathematical-based. Time-based algorithms generate passwords that are valid for a set period of time before automatically updated by the algorithm (often a hardware device). Technically, a one-time is a misnomer as a password can be used multiple times as long as it is within one time period. A hardware device of this type typically always displays a password, and the password is constantly changing. The length of time that an OTP is valid is an important security parameter in these schemes because one password is valid until the time period expires and then updated. If a password is infrequently updated, an attacker has a longer window for exploitation. As the period length grows, the security of OTPs approaches that of conventional passwords. For example, an eavesdropper could capture the OTP that has just been generated as it travels across a network. Once captured, the attacker has the entire lifetime of the password for unauthorized access.

One implementation difficulty of time-based passwords is that the clock of the password generator must be synchronized with the clock of the authenticator or else a user may be wrongfully denied authentication. Another concern when utilizing a time-based OTP is clock drift. Clock drift is the phenomena where different clocks do not run at the same speed and after some time, will not have the same time value at the same point in time, even if they did previously. Often, the way to reduce problems with clock drift is to have periodic clock resynchronization or have the authenticator accept a range of passwords, say two time periods, reducing the security of the system. SecurID is a proprietary commercial system by RSA Security that uses hardware devices to generate passwords that change every thirty or sixty seconds [5].

For a concrete example of one such time-based one-time password algorithm, the public algorithm Time-based One-Time Password (TOTP) is described.

Setup

- Establish a shared secret integer k between the authenticator and the user
- Agree upon a time step X and an initial time T_0

Authentication

- The user calculates $T = \left\lfloor \frac{(current\ time - T_0)}{X} \right\rfloor$

- The user sends the authenticator $HOTP(K, T)$ where HOTP is a OTP generation algorithm based upon cryptographic hash functions described below

The authenticator is able to verify that the password is valid as any password generated outside of the current time step since T_0 will differ. Also, since the shared key is present, the authenticator can be sure the password was generated by someone in possession of the shared key [6].

Mathematical-based generate passwords that do not have values based on time; instead they are generated algorithmically each time a password is desired. The authenticator and client's algorithms are in step with each other. In many cases, this involves a counter that is incremented each round. Every time a client is successfully authenticated, the authenticator will only accept the next OTP generated by the algorithm. Unlike time-based OTPs, mathematically-based OTPs are valid for only one use and are not vulnerable to the attack described above in which a password is used twice within one period. Also, clock synchronization and drift are not a concern. One area of concern is if a mathematical-based password is stolen, it can be valid indefinitely, as long as the real user does not attempt to authenticate, thus advancing the password the authenticator is expecting.

Implementation issues for these algorithms include keeping the authenticator and client's algorithms synchronized. For example, suppose a client's hardware device presented a password and the user of this device is unable to correctly enter that number to the authenticator. If the user indicated to the hardware prematurely that the password was accepted, the user has no way of retrieving the old password that the authenticator is waiting for. To combat this, often implementations have the authenticator accept a window of passwords and adjust this window based on the last valid password entered. S/Key and hash-based OTP (HOTP) are two popular forms of mathematical-based password generators.

For a concrete example, S/Key, sometimes known as Lamport's scheme, is a popular OTP solution developed to authenticate dumb terminals on Unix-like operating systems [7]. The password generation is based off a cryptographic hash function.

Setup

- Establish a secret key s
- Apply a cryptographic hash function $H(x)$ to s n times
- Discard s
- The authenticator stores $H^n(s)$ while the clients stores $H^i(s)\ for\ i\ \in\ \{2, \dots,\ \ n\}$

Authentication

- To be authenticated the i^{th} time, i the user provides $H^{n-i}(s)\ for\ i\ \in\ \{1, \dots, n-1\}$
- The authenticator computes $H\left(H^{n-i}(s)\right)$ and compares it the stored value on the server: $H^{n-i+1}(s)$
- If the result matches the stored value, discard the stored value and store $H^{n-i}(s)$

For simplicity, the output of the hash function can be mapped to simple English words to facilitate input into a terminal. This is an example of a solution that has a finite number of uses before re-initialization of the system must take place. Most implementations forbid this re-initialization from being done remotely.

Other more modern schemes such as the HOTP do not have this re-initialization limitation. HOTP is built upon hashed-based message authentication codes (HMAC) and often uses the Secure Hash Algorithm (SHA-1) for cryptographic hashing [8]. HOTP is published by the Initiative for Open Authentication (OATH). Conceptually, HOTP calculates the SHA-1 based HMAC keyed with a shared secret on a counter. The steps to compute a given password with HOTP are:

- Establish a shared secret s
- Initialize a counter c to zero
- Define H as an HMAC calculated with SHA-1
- Let $Truncate$ be a function that selects 4 bytes in a standard manner
- A client wishing to be authenticated sends $Truncate\big(H(s,c)\big)\ \&\ 0\text{x7FFFFFFF}$
- If the authenticator's value matches the clients, then the client is authenticated
- Both parties increment c

As in the case with S/Key, the result is often too large to be entered into a keypad or other device by a client. Instead of transforming the result into words as in S/Key, HOTP simply calculates the remainder when divided by the maximum value ($result\ mod\ 10^{desired\ output\ size}$). Many other systems based upon this approach exist. One could swap out HMAC with the Data Encryption Standard (DES) algorithm, the Advanced Encryption Algorithm (AES), or any other encryption mechanism. HOTP's sister standard in OATH, Time-based One Time Password (TOTP), replaces the counter c with the current time, thus converting the mathematical-based HOTP into a time-based OTP.

Secret Sharing

Secret sharing is a mechanism that allows a secret to be divided into multiple shares, such that when a certain amount of these shares are combined, the secret can be obtained. For notation, a scheme that divides a secret s into n shares and requires k shares to reconstruct the secret is called a (k, n)-threshold scheme. An attacker possessing $k - 1$ shares should have low probability of successfully reconstructing s, but anyone in possession of k shares should efficiently be able to construct s with probability 1 [9]. Simple secret sharing schemes can be divided into two phases, dealing of the secret and reconstruction of the secret.

Secret Sharing was described in 1979 independently by Shamir and Blakley and has since triggered numerous papers describing properties and uses of secret sharing schemes. The canonical use of secret sharing schemes is to distribute a long term key, such as a private key to a major bank, in a manner that prevents a single break-in from compromising the entire system. Recently this mechanism has been used to distribute the long term key signing key of the Domain Name System Security Extensions (DNSSEC) root zone with a $(5,7)$-threshold scheme [10]. In the event of the necessity to generate this key, 5 of these 7 individuals will have to meet at a base in the United States.

For a concrete implementation of such schemes, this paper will discuss the version presented by Shamir. Shamir's original secret sharing scheme is the most popular scheme and is often used in most academic examples and papers on the subject. This scheme works by a dealer defining a polynomial of degree k where the polynomial evaluated at 0 is the secret. The shares are the polynomial evaluated at other unique points.

- Establish a shared secret s
- Define $f(x) = s + a_1 \times x^1 + \ldots + a_{k-1} \times x^{k-1}$ where k is the threshold value
- The set of shares is $\{(i, f(i)) : i \in \{1, \ldots, n\}\}$ where n is the number of shares to be made

To reconstruct the secret from only the shares, use the Lagrange polynomial to interpolate the polynomial from the given points. Once the polynomial has been reconstructed, the shared secret is the function evaluated again at 0.

- Given a set S of shares of size k, define x_m to be the input to the constructed polynomial above and y_m to be the output of the constructed polynomial for share $m \in S$
- Compute $l_i = \sum_{m=0}^{m<k, m\neq i} \frac{-x_m}{x_i - x_m}$ where i is the index of each share
- Compute the secret by $\sum_{j=0}^{k} y_j \times l_j$

Shamir's scheme is information-theoretic secure. By this, we formally mean that every possible secret is equally likely after having viewed up to $k-1$ shares. This prevents an attacker from having a reduced amount of secrets to guess by observing some of the shares, but not k shares, that have been compromised. This will be shown later.

Verifiable Secret Sharing

In a conventional secret sharing scheme, a corrupt dealer can deal invalid shares to participants or malicious users may present an invalid share in an attempt to gain information about the secret. With these capabilities, a dealer can restrict which sets of shares that can actually reconstruct the correct secret. There are attacks where an attacker who was not dealt a share initially presents invalid shares to other participants to prevent them from being able to reconstruct the secret and simultaneously allowing the attacker to recreate the secret [11].

Verifiable secret sharing (VSS) solves these problems by guaranteeing that even if the dealer is malicious and delivers corrupt shares to some participants or some participants present an invalid share, there is a well-defined secret all participants will reconstruct. These schemes often provide a mechanism for detecting an invalid share with high probability. Thus, a dealer or malicious participant is unable to present an invalid share without being caught. This is usually accomplished by having the dealer commit to some information pertaining to the secret and publically release it. Then any participant can verify the share they received independently by using the public information and the information that they received. Since the information is public, the participant can be assured that everyone is using the same information to verify their shares and was not dealt bad information.

Paul Feldman proposed a popular VSS scheme based on homomorphic encryption [12]. The scheme is non-interactive, meaning bidirectional communication between the participant and the dealer of the shares is not required. The scheme is not information-theoretic secure as it leaks information about the secret – a non-computationally bound adversary can learn the secret by observing information related to the VSS procedure. The scheme does have the property that an infinitely powerful dealer is unable to generate an incorrect share.

A VSS scheme devised by Pedersen is considered the dual of Feldmen's in the sense that it is in non-interactive and information-theoretically secure [13]. However, in this scheme, if the dealer is not computationally bound, then he is capable of generating corrupt shares. Pedersen shows the impossibility of achieving a system which is simultaneously information-theoretic non-interactive and one in which a non-computationally bound dealer cannot cheat. This means that while maintaining non-interactivity, a verifiable secret sharing scheme either allows non-computationally bounded attackers to learn information about the secret or allows non-computationally bounded dealers to cheat, but not both.

Conceptually, Pedersen's scheme will share a secret using Shamir's secret sharing scheme but also have the dealer commit to each share while each participant verifies these commitments.

- Chose p and q large primes such that $q \mid p - 1$
- Define G_q the unique subgroup of order q, and g a generator of G_q
    - It can be tested if an element $a \in G_q$ by $a \in G_q \Leftrightarrow a^q = 1$
- The dealer D wishes to share an $s \in \mathbb{Z}_q$
- Chose a $g, h \in G_q$ such that nobody knows $log_g(h)$
- Define $E(s,r)$ as $g^s \times h^r$
- Chose a random $r \in \mathbb{Z}_q$
- D publishes a commitment to the secret s: $E_0 = E(s,r)$
- D constructs a second polynomial of degree $k-1$: $r(x) = r + \sum_{j=1}^{k-1}(r_j \times x^j)$

- D broadcasts $E_i = E(f_i, r_i)$ for $i \in \{1, \dots, k-1\}$ and f_i are the coefficients of $f(x)$ used to distribute the secret
- D computes $r(i)$ for $i \in \{1, \dots, n\}$
- D sends participant P_i $(x_i, r(i))$ for $i \in \{1, \dots, n\}$
- Each participant P_i verifies their share $x_i = f(i)$ for $i \in \{1, \dots, n\}$ by $E\big(x_i, r(i)\big) = \prod_{j=0}^{k-1} E_j^{i^j}$
    - For clarity: $E_0^{i^0} + \dots + E_{k-1}^{i^{k-1}} = E(f_i, r_i)^{i^0} + \dots + E(f_i, r_i)^{i^{k-1}} = (g^{f_i} \times h^{r_i})^{i^0} + \dots + (g^{f_i} \times h^{r_i})^{i^{k-1}} = g^{f_i \times i^0 + \dots + f_i \times i^{k-1}} \times h^{r_i \times i^0 + \dots + r_i \times i^{k-1}} = g^{f(i)} \times h^{r(i)} = g^{x_i} \times h^{r(i)}$

The dealer commits to the secret with $E(s, r)$ and subsequently commits to the material that generated the share with $E(f_i, \mathrm{r_i})$. This information, along with other public information such the shares, reveals no information about the secret but allow participants to verify their share is a portion of the secret.

Proactive Secret Sharing

Generally, secret sharing schemes are designed to protect long term secrets, but if more than k shares were compromised over this long period of time, the secret is no longer safe. Thus there is a need to protect the shares over a long period of time. Proactive secret sharing (PSS) is a modification to the typical secret sharing scheme that allows shares to be updated periodically [14]. A PSS takes a set of shares and updates them such that attempting to reconstruct the secret with both non-updated and updated shares fails. PSSs allows up to $k-1$ shares to be leaked within each period between updates. If an attacker gathers k or more shares from the same period, the secret is still compromised as in traditional secret sharing. Stated differently, more than $k-1$ shares can be revealed as long as no more than $k-1$ of those shares are from the same period t. This has the effect of reducing the lifetime of the secret to the length of the period chosen. This makes the length of the period an important security parameter. As this length reaches infinity, the scheme becomes less effective, but as the period approaches 0, precious resources are spent constantly updating the shares.

A PSS scheme presented by Herzberg et. al. uses a modified Shamir secret sharing scheme that requires all participants P_i for $i \in \{1, \dots n\}$ to update the shares every period. The scheme works with Shamir's scheme presented above, but after each period is expired, an update process takes place. The update process involves each participant creating a random polynomial whose free coefficient is 0 (i.e $f(0) = 0$). Each participant distributes this polynomial evaluated at certain values such that at the end, each share has been updated correctly by every participant's function. The update process for a normal Shamir secret sharing scheme is:

- Each participant P_i for $i \in \{1, \dots n\}$ generates a $k - 1$ degree polynomial $\delta_i(z)$ where $\delta_i(0) = 0$
- P_i sends P_j $u_{i,j} = \delta_i(j)$ securely
- P_i updates his share x_i by $x_i = x_i + \sum_{j=1}^{n} u_{j,i}$
- Old shares and update information are deleted

This scheme clearly shows that the original constructed polynomial $f(x)$ will be replaced with $f(x) + \delta_1(x) + \dots + \delta_n(x)$. Since zero was the free coefficient for the polynomials constructed by every participant, the new polynomial evaluated at 0 is still the original secret: $f(0) = f(0) + \delta_1(0) + \dots + \delta_n(0)$. An attacker attempting to contribute one or more previously compromised shares that have since been updated will cause the entire reconstruction process to fail as a different polynomial will be reconstructed.

This PSS is vulnerable to a man-in-the-middle attack that could lead to a denial of service. An attacker could realize that an update process was initiated by listening for update material from any participant and replacing the update material with their own. This technique corrupts the share that is being updated and prevents it from being able to participant successfully in future reconstructions of the secret.

To corrupt the share, the attacker just needs to send update material that differs from the polynomial value that was intended for the victim. As the victim's share replaces the previous version, the share is permanently corrupted. One or more attackers could repeat this kind attack to a total of $n - k + 1$ participants. This would destroy the possibility of recreating the secret as there are not enough valid shares to meet the threshold value required.

For demonstration purposes, if the attacker was a malicious participant, by constructing a polynomial $\delta(x)\ s.t.\ \delta(0) \neq 0$, the participant could destroy the secret and prevent successful reconstruction of the secret. An example of such a corruption is presented below.

- Participant P_x is malicious
- P_x generates a $k - 1$ degree polynomial $\delta_x(z)$ where $\delta_x(0) \neq\ 0$
- P_x sends P_j $u_{x,j} =\ \delta_x(j)$ for $j \in \{1, \dots, n\} \setminus x$

If the malicious participant is successful in having any of the other participants accept the corrupted update material, the secret not be able to be successfully reconstructed if that participant participates. The attacker will have successfully launched a denial of service attack on the scheme without violating any security principles.

In this scheme, there is no method to detect this attacker. VSS schemes such as the one presented above may be utilized to commit to update material. Participants would not reconstruct the secret when there is an invalid share present nor would they update their share if the update information was invalid. The prevention of this attack could also be facilitated by normal cryptographic utilities such as asymmetric encryption; however this is not information-theoretically secure.

Proposed OTP solution using Secret Sharing Techniques

Using secret sharing, verifiable secret sharing, and proactive secret sharing, I will present a mathematical-based OTP solution that is information-theoretically secure. The solution requires three phases: dealing, reconstruction, and update.

Motivation

The idea of using a shared secret as a form of authentication is nothing new as many schemes use this already. For instance, static passwords rely on a shared secret between the authenticator and the user wishing to be authenticated. Using a portion of a shared secret could also be a method of authentication.

The most important property that arises when using only the portion of a secret as an authentication mechanism is that the portion is information-theoretic secure. This means that the secret that has been split is not revealed when seeing only one portion as long as the threshold is greater than one. This is great for environments like the Internet that require information to be publically viewable to many users. For this property to be useful the secret must have some intrinsic value that the share does not or else an eavesdropper would be satisfied capturing the share. Also, since the secret is not available on any one computer, stealing the secret is more difficult. Once again, this is not important unless the secret has some value besides authentication, such as a Social Security Number. The property of information-theoretic secure shares was the driving motivation for investigating the use of secret sharing for authentication.

Another property of using partial secrets as authentication is their distributed nature. For instance, a password could be broken up among many people and only when so many are together can authentication of a group be established. The often used example of launching a nuclear missile depending on multiple high ranking officials comes to mind.

Despite these obvious uses, there has not been any research into using secret shares strictly as an authentication mechanism nor a periodic/temporary authentication mechanism as is the case with one-time passwords. If the share was just delivered to the authenticator as proof of identify, secret sharing authentication provides no additional protection to traditional passwords. However, used in an OTP system with changing shares, the properties listed above can be realized. Often time, the techniques used to construct this secret sharing based OTP come from research that had completely different visions in mind. For example, the papers that describe updating secrets in a proactive manner had in mind preventing long term shared secrets, such as private keys, from being compromised. In the case of an OTP system, updating shares is a great way to generate new passwords. Verifiable secret sharing was intended to prevent malicious dealers from preventing certain participants from being able to successfully reconstruct the secret but has in the case of a password system validating update material.

Overview

The user and an authenticator will agree on a shared secret, like a conventional password. This shared secret will then be split using a $(2, 2)$-threshold scheme (2 shares requiring both shares to reconstruct the split secret). The user receives a share and the authenticator receives the other share. Under normal situations, this process can be happen via the distribution of a smart card or other physical contact. If an encrypted connection is utilized, the theoretical solution can be compromised unless an information-theoretic secure encryption mechanism is used. Using the share as the password, the user presents this to the authenticator. The authenticator uses his share and the user's share to reconstruct the secret and compare it to the stored secret. If the secret is successfully reconstructed, the user is successfully authenticated. After successful authentication, both parties use proactive secret sharing to update the shares and unlike traditional proactive schemes, the secret is updated as well. Both parties should invalidate the old shares and secret by securely deleting the material.

Detail

Expanding on the overview above is a detailed description. The secret sharing techniques verifiable secret sharing and proactive secret sharing have been tailored to suit an OTP solution. The entire solution will be proven to be information-theoretic secure in later sections. The setup involves an OTP generator, which produces passwords, and authenticator, which verifies password, sharing and communicating information.

Setup

A password generator and authenticator need to setup a shared secret as per Shamir's classic paper. Since most of the variables are updated during the update procedure, they will be designated with superscripts describing the interval in which they are valid. For example, the polynomial $f^0(x)$ is valid only for the first password (before the first password is used).

- The generator and authenticator agree on two large primes p and q such that q divides $p-1$
    - p will be used in the verification section later
- The generator and authenticator agree on a field F of order q
- Agree on a shared secret $x^0 \in F$
- The authenticator constructs a polynomial with coefficients $f^0(x) = a_0^0 + a_1^0 \times x$ where $a_0^0 = x^0$
- The generator accepts the share $x_1^0 = f^0(1) = a_0^0 + a_1^0 \times 1$ securely
- The authenticator saves the share $x_2^0 = f^0(2) = a_0^0 + a_1^0 \times 2$ securely

Use

The password for the generator to present to the user who is to enter it into the authenticator is the generator's share.

- The user enters x_1^t as her password where t represents the current interval

- The authenticator uses x_1^t and x_2^t to reconstruct the polynomial $f^t(x) = a_0^t + a_1^t \times x$ via interpolation
- The authenticator computes $f^t(0) = a_0^t = x$ and compares to the stored secret x^t

Update

The update process changes the last used password into the next. The transformation uses proactive secret sharing to prevent information about the secret being leaked from previous passwords.

- The generator picks $\delta_{1,1}^t$ in F to construct $\delta_1^t(x) = x^t + \delta_{1,1}^t \times z$
- The authenticator picks $\delta_{2,1}^t$ in F to construct $\delta_2^t(x) = x^t + \delta_{2,1}^t \times z$
- The generator computes $u_{1,1}^t = \delta_1^t(1)$ and $u_{1,2}^t = \delta_1^t(2)$
- The user enters $u_{1,2}^t$ to the authenticator
- The authenticator computes $u_{2,1}^t = \delta_2^t(1)$ and $u_{2,2}^t = \delta_2^t(2)$
- The authenticator presents $u_{2,1}^t$ to the user to enter into the generator
- The generator constructs the a share $x_1^{t+1} = x_1^t + u_{1,1}^t + u_{2,1}^t$
- The authenticator constructs a new share $x_2^{t+1} = x_2^t + u_{1,2}^t + u_{2,2}^t$
- Both parties update the secret $x^{t+1} = 3 \times x^t$
- Both parties delete securely all invalid information

Any future verification operates as described above.

Verification

Currently, an attacker is able to perform a denial of service attack by sending corrupted update material that destroys shares. This was described on the general description of proactive secret sharing. To prevent this, digital signatures may be used. By having the authenticator and generator sign their update material they can accept the update material with the confidence the signing algorithm provides. Similarly a keyed hash function keyed with the secret may also be used for authentication as non-repudiation is not needed. However, unless the mechanism is information theoretic secure, this would still present a vulnerability to an attacker with unlimited computational ability. By breaking the asymmetric cryptographic scheme or producing collisions in the hash function, an attacker could corrupt a participant of the OTP scheme's generated OTP. To correct this denial-of-service vulnerability, verifiable secret sharing should be used.

Verifiable secret sharing (VSS) is a technique that often uses bit-commitment schemes to ensure that shares in a secret sharing scheme are valid. Presented below is a modified version of the update scheme above that uses Pedersen's information-theoretic VSS scheme to commit to the update material. Conceptually, the generator commits to the update material using the secret. The authenticator is able to verify the update material because the authenticator is in possession of the secret. The same process works for the authenticator desiring to commit to the update information intended for the generator. The outline presented below is in the case of the generator creating information for the authenticator to verify its update material. The reverse situation will be outlined afterwards.

- Define G_q as the unique cyclic subgroup of $\mathbb{Z}_p^{\times}$ order q where p was agreed upon during initialization.
    - It can be tested if an element a is contained in G_q by $a \in G_q \Leftrightarrow a^q = 1$
- Choose $g, h \in G_q$ such that nobody knows $log_g(h)$. Store these values at initialization time
- The parameters g, h and G_q are parameters defining the system
- The generator wishes to share $u_{1,2}^t = \delta_1^t(2) = x^t + \delta_{1,1}^t \times 2 \in \mathbb{Z}_q$
- Define a function E as $E(a, b) = g^a \times h^b$
- The generator constructs a polynomial of degree 1: $r_1^t(x) = r_{1,1}^t + r_{1,2}^t \times x$ for $r_{1,1}^t, r_{1,2}^t \in \mathbb{Z}_q$
- The generator outputs a commitment to the secret x^t: $E_{1,0}^t = E\left(x^t, r_{1,1}^t\right)$
- The generator outputs $E_{1,1}^t = E\left(\delta_{1,1}^t, r_{1,2}^t\right)$

- The generator outputs $r_1^t(2)$ and $u_{1,2}^t$
- The authenticator verifies $u_{1,2}^t$ is valid by checking whether $E\left(u_{1,2}^t,\ r_1^t(2)\right) =\ E_{1,0}^t \times \left(E_{1,1}^t\right)^2$ is true.
    - For clarity of the steps: $E_{1,0}^t \times \left(E_{1,1}^t\right)^2 = \left(g^{x^t} \times h^{r_{1,1}^t}\right) \times \left(g^{\delta_{1,1}} \times h^{r_{1,2}^t}\right)^2 = g^{x^t+\delta_{1,1}\times 2} \times\ h^{r_{1,1}^t+r_{1,2}^t\times 2} = g^{\delta_1^t(2)} \times h^{r_1^t(2)} = E\left(u_{1,2}^t,\ r_1^t(2)\right)$

The same process outlined above when altered slightly can be used when the authenticator is generating material for the generator to verify. The authenticator would follow the steps, generating a random polynomial $r_2^t(x)$ and sharing $E_{2,0}^t = E\left(x^t, r_{2,1}^t\right),\ E_{2,1}^t = E\left(\delta_{2,1}^t, r_{2,2}^t\right), u_{2,1}^t$, and $r_2^t(1)$. The verification of this material by the generator would be accomplished by checking whether $E\left(u_{2,1}^t, r_2^t(1)\ \right) = E_{2,0}^t \times E_{2,1}^t$. Note the differences that $r_2^t(x)$ is evaluated at 1 instead of 2 and that verification does not exponentiate $E_{2,1}^t$. Once the update material has been verified, then both parties can update their shares as shown above without worry of corruption.

Worked Example

A worked example is presented below for clarity. This example offers little security as the field chosen is small and brute forcing the current password is feasible.

Setup

- The generator and authenticator agree two large primes 23 and 11, note that 11 divides $23 - 1$
- The generator and authenticator agree on a field $\mathbb{F} = \mathbb{Z}_{11}$
- Both agree on a shared secret $x^0 = 3 \in \mathbb{Z}_{11}$
- The authenticator constructs a polynomial $f^0(x) = 3 + 5 \times x$ over $\mathbb{Z}_{11}$
- The generator accepts the share $x_1^0 = f^0(1) = 3 + 5 \times 1 = 8\ (mod\ 11)$ securely
- The authenticator saves the share $x_2^0 = f^0(2) = 3 + 5 \times 2 = 2\ (mod\ 11)$
- Let $G_{11} = \{1,2,3,4,6,8,9,12,13,16,18\}$, the unique cyclic subgroup of $\mathbb{Z}_{23}$ of order q

- The authenticator and generator agree on $g = 3$ and $h = 12$

Use

The generator uses the share as the one-time password.

- The generator presents $x_1^0 = 8$ as the user's password
- The authenticator uses $x_1^0 = 8$ and $x_2^0 = 2$ to reconstruct the polynomial $f^0(x) = 3 + 5 \times x$ via interpolation
- The authenticator computes $f^0(0) = 3 + 5 \times 0 = 3\ (mod\ 11)$ and compares it to the secret 3

Update with Verification

- The generator picks $\delta_{1,1}^0 = 6$ in F to construct $\delta_1^0(z) = 3 + 6 \times z$
- The generator computes $u_{1,1}^0 = \delta_1^0(1) = 9\ (mod\ 11)$ and $u_{1,2}^0 = \delta_1^0(2) = 4\ (mod\ 11)$
- The generator picks constructs a random polynomial $r_1^0(x) = 9 + 2 \times x$ over $\mathbb{Z}_{11}$
- The authenticator picks $\delta_{2,1}^0 = 1$ in F to construct $\delta_2^0(z) = 3 + 1 \times z$
- The authenticator computes $u_{2,1}^0 = \delta_2^0(1) = 4\ (mod\ 11)$ and $u_{2,2}^0 = \delta_2^0(2) = 5\ (mod\ 11)$
- The authenticator picks constructs a random polynomial $r_2^0(x) = 7 + 10 \times x$ over $\mathbb{Z}_{11}$
- The generator outputs
  - $E_{1,0}^0 = g^{x^t} \times h^{r_{1,1}^0} = 3^3 \times 12^9 = 16\ (mod\ 23)$
  - $E_{1,1}^0 = g^{\delta_{1,1}^0} \times h^{r_{1,2}^0} = 3^6 \times 12^2 = 4\ (mod\ 23)$
  - $u_{1,2}^0 = 4$
  - $r_1^0(2) = 2\ (mod\ 11)$
- The authenticator outputs
  - $E_{2,0}^0 = g^{x^t} \times h^{r_{2,1}^0} = 3^3 \times 12^7 = 18\ (mod\ 23)$
  - $E_{2,1}^0 = g^{\delta_{2,1}^0} \times h^{r_{2,2}^0} = 3^1 \times 12^{10} = 6\ (mod\ 23)$
  - $u_{2,1}^0 = 4$

  - $r_2^0(1) = 6\ (mod\ 11)$

- To verify, the authenticator computes the following and compares the results for equality
  - $E(4,2) = 3^4 \times 12^2 = 3\ (mod\ 23)$
  - $E_{1,0}^0 \times \left(E_{1,1}^0\right)^2 = 16 \times 4^2 = 3\ (mod\ 23)$
- The authenticator computes the new share $x_2^1 = x_2^0 + u_{1,2} + u_{2,2} = 2 + 4 + 5 = 0\ (mod\ 11)$
- To verify, the generator computes the following and compares the results for equality
  - $E(4,6) = 3^4 \times 12^6 = 16\ (mod\ 23)$
  - $E_{2,0}^0 \times E_{2,1}^0 = 18 \times 6 = 16\ (mod\ 23)$
- The generator computes the new share $x_1^1 = x_1^0 + u_{1,1} + u_{2,1} = 8 + 9 + 4 = 10\ (mod\ 11)$
- Both participants update their secret $x^1 = 3 \times x^0 = 3 \times 3 = 9\ (mod\ 11)$
- Both participants securely delete all invalid information
- The updated shares construct the correct secret 9 as demonstrated by $2 \times 10 - 0 = 9\ (mod\ 11)$

This process has been visualized in Figure 1 for understanding purposes. Next time the user wants to be authenticated, the authenticator must check the newly updated secret instead of the original secret that they agreed upon at initialization and use the updated shares to reconstruct this newer secret.

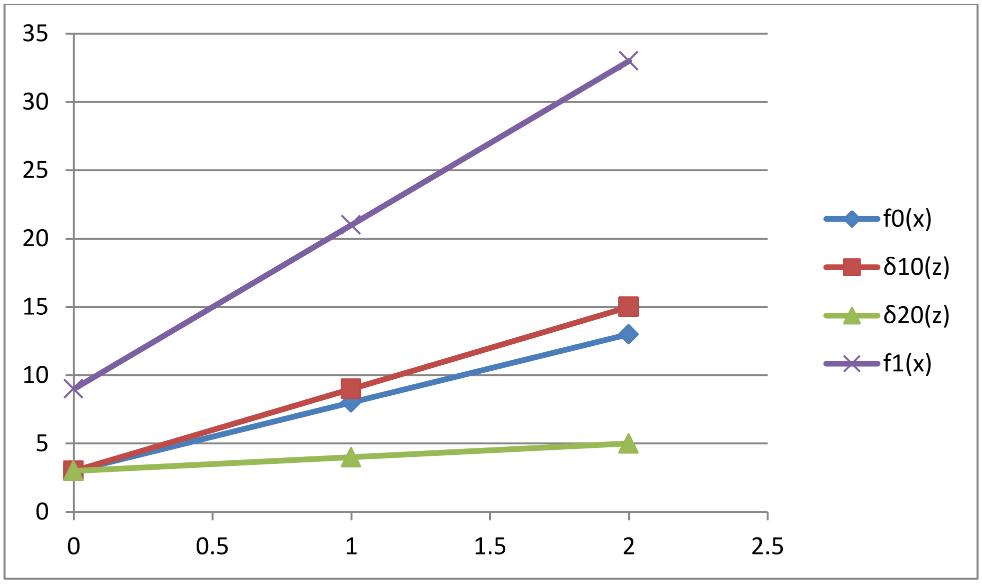


Figure 1: The functions involved in the update procedure. Note values have not been modulated into $\mathbb{Z}_{11}$ for visualization purposes.

Security Analysis

The security of the proposed OTP solution depend on the security of Shamir's secret sharing scheme, the security of proactive secret sharing, and the security of verifiable secret sharing (if it is used). Shamir's secret sharing scheme is proven information theoretic secure as any set of shares less than the threshold gives no information on the secret. By adapting the proof for general purpose secret sharing to that of a scheme with a fixed number of shares (2) and a set threshold (2), showing the security properties of the OTP scheme follows easily.

Proposed OTP Scheme Proof Outline

To show the security of the one-time password scheme, we need to show that by seeing a password, the number of possible secrets ($f(0)$) used in generating that password is unchanged. We show this by establishing a one-to-one correspondence between the possible functions that could have generated the observed password and all secrets that were initially possible. Since the functions that could have generated the share we observed are related to all possible secrets in this correspondence, then any of these functions with unique secrets $f(0)$ could have resulted in the generation of the observed password.

Proposed OTP Scheme Proof

Establish the one-time password scheme by sharing a secret s between a generator and an authenticator. Now suppose an eavesdropper observed the generator's password c generated by $\mathrm{f}(1)$. Let the set G represent all functions that could have resulted in generating this OTP.

- $\mathrm{G} = \{\mathrm{g(x)\ is\ a\ 1\ degree\ polynomial\ s.t.\ g(1) = c}\}$

Now we establish a mapping between these functions and the secrets that they represent.

- $\mathrm{m} : \mathrm{G} \rightarrow \mathbb{Z}_\mathrm{q}$ by defining $\mathrm{m}\big(\mathrm{g(x)}\big) = \mathrm{g}(0)$

The first step to show a one-to-one correspondence in this case is to show that the size of G and $\mathbb{Z}_\mathrm{q}$ are the same, more specifically they are both of size q. $\mathbb{Z}_\mathrm{q}$ is trivially of size q. For G, we first notice that there are q choices for the 0th coefficient which corresponds to $\mathrm{g}(0)$. By fixing each possible coefficient along with the known $\mathrm{g}(1)$, g is completely determined because $\mathrm{g(x)}$ is a 1 degree polynomial with 2 known values $\mathrm{g}(0)$ and $\mathrm{g}(1)$ and can be determined with polynomial interpolation. Every function is unique because $\mathrm{g}(0)$ differs for every function. By construction of G where we defined functions based on $\mathrm{g}(0)$, we can see this mapping is onto as $\mathrm{a} \in \mathbb{Z}_\mathrm{q}$ is mapped by the function g where $\mathrm{g}(0) = \mathrm{a}$. Since $\mathbb{Z}_\mathrm{q}$ and G are of the same size q and the mapping is onto, the mapping is a one-to-one correspondence.

Update Procedure Proof Outline

Proactive secret sharing is information-theoretically secure as long as seeing the publically exchanged information does not give any information about the private information used to update the password or the original secret. To show the security of the update mechanism, we use a similar approach as the proof for the password scheme above. We need to show that by seeing the update material, the number of functions that could have generated that update material still produce every possible value for the unknown update information that is necessary in updating the password. We show this by establishing a one-to-one correspondence between the number of functions capable of generating the revealed update material and all possible values for the hidden material. This will show that seeing the public update material does not give an advantage into determining the hidden update material as all possible values for the hidden update material are still possible.

Update Procedure Proof

Establish the one-time password scheme by sharing a secret s between an authenticator and a generator. Now suppose the update phase generate the next password begins. The generator picks a function $\delta_1(z) = x^t + \delta_{1,1}^t \times z$ and authenticator picks $\delta_2(z) = x^t + \delta_{1,1}^t \times z$ both over $\mathbb{Z}_q$. As per the update procedure, the generator shares $\delta_1(2)$ with the authenticator while the authenticator shares $\delta_2(1)$ with the generator. Let an eavesdropper observe this update information in the form of the ordered pair $(\delta_1(2), \delta_2(1))$. An attacker needs to be able to determine $\delta_1(1)$ to calculate the next password as the used password because the new password is $\delta_2(1) + \delta_1(1) + x_1^t$ and $\delta_2(1)$ and x_1^t have already been observed. Let the set G represent the set of all functions that could have generated this pair.

- $G = \{(g_1, g_2) : g_1(2) = \delta_1(2), g_2(1) = \delta_2(1), g_1(0) = g_2(0) = x^t\}$

Note that both these functions share the same value when evaluated at 0, but an eavesdropper does not know this value as proven previously. Now we establish a mapping between these functions that could have generated the observed update material and all the values that $\delta_1(1)$ could have initially taken.

- $m : G \rightarrow \mathbb{Z}_q$ defined by $m\big((g_1, g_2)\big) = \big(g_1(1)\big)$

The first step in establishing the one-to-one correspondence is to show that the size of both sets is q. For $\mathbb{Z}_q$, this is trivially true. For G, since these functions are of degree 1, 2 points determine the function by interpolation. That is, if we pick any 2 points, there is only 1 function of degree 1 that passes through those points. So for the functions in G which are the same when evaluated at zero, there are q choices for that point and all q choices define unique functions since two unique points are defined for each function. Thus there are q functions pairs in G and the size of G is q. To show that this relation is a one-to-one correspondence, we only need to show that the mapping in onto (surjective) as both sets are of order q. Pick a number $x \in \mathbb{Z}_q$, let the function g_1 be the function such that $g_1(0) = x$ and $g_1(2) = \delta_1(2)$. Let the function g_2 be the function such that $g_2(0) = x$ and $g_2(1) = \delta_2(1)$. Both g_1 and g_2 are in G due to our construction of the pairs in G. Both sets are of size q and the mapping is onto, thus the mapping is a one-to-one correspondence.

Initially, an attacker has q choices to guess what value is being added to the password. After seeing the update material, there are still q choices and thus an attacker has learned nothing about this value. A similar proof can show that an attacker is unable to determine which value is being added to the authenticator's share. No information was learned about x^t because no information was leaked about the update material which could be used to interpolate x^t. The strategy outlined in the proof however shows how to enumerate all q choices of hidden update material. By fixing the point where both update functions have the same value (0) and taking one of the update values, the other value is determined because the polynomial is of degree 1. This could be done for both update function, as well as the secret splitting function $f^t(x)$. An attacker could construct a table as below, but this information cannot be used to determine anything, just enumerate possible values.

x^t	$\delta_1(1)$	$\delta_1(2) = u_{1,2}^t$	$\delta_2(1) = u_{2,1}^t$	$\delta_2(2)$	$f^t(1) = x_1^t$	$f^t(2)$
0	$u_{2,1}^t \times 2^{-1}$	$u_{1,2}^t$	$u_{2,1}^t$	$\delta_2(1) \times 2$	x_1^t	$x_1^t \times 2$
…	…	…	…	…	…	…
q	$\left(u_{2,1}^t - q\right) \times 2^{-1} + q$	$u_{1,2}^t$	$u_{2,1}^t$	$\delta_2(1) \times 2 - q$	x_1^t	$x_1^t \times 2 - q$

Table 1: All possible hidden update material for each fixed secret.

This attack cannot be strengthened by observing many passwords and update material as all update material is completely random each time. This table shows that by seeing all public information, an attacker still the same number of possible guesses at the next password as before the this information was revealed.

There should be some note about the secret and the method in which it is updated. Although the shares are updated by adding random values, the secret is simply multiplied by 3 repeatedly. If 3 is not a primitive root of the group in which the secret belongs, then the secret will cycle through some subgroup of $\mathbb{Z}_q$. Even if the secret does cycle through every element of the group, the secret is still following a cycle and will have the same value at some future point of the scheme. Stronger yet, the secret will have the same progression of values in the future.

This does not alter the security of the update mechanism. Although an attacker may be able to determine that secret at some point in time is equal to a previous secret, no information about that secret was leaked so that all secrets are still possible. An attacker is not able to use this knowledge in an attempt to determine the private update material, essential for determining the next password. This is because the shares and update material are chosen independently of the secret and can vary even if the secret is not. Even if a problem were found regarding this issue, an implementation could be written to require re-initialization when a secret would cycle or check the cycle length of a given secret and determine if it is long enough for the lifetime of the authentication system.

Verification Procedure Proof Outline

This proof will be in the perspective of the generator creating verification material for the authenticator. The same method can be used to prove the other situation. To prove the security of the verification process, we need to show that by seeing $E^t_{1,0}, E^t_{1,1}, r^t_1(2)$, and $\delta^t_1(2)$ an attacker gains no knowledge of $\delta^t_1(x) = x^t + \delta^t_{1,1} \times 2$ or $r^t_1(x) = r^t_{1,1} + r^t_{1,2} \times x$. We have already shown that $\delta^t_1(2)$ gives no information on the secret and is chosen independently from $r^t_1(2)$ so $\delta^t_1(2)$ does not yield any information to an attacker. Similarly to proving secret sharing, $r^t_1(2)$ does not reveal any information on the rest of $r^t_1(x)$. Thus the security of verifiable secret sharing rests with showing that $E^t_{1,0}$, and $E^t_{1,1}$ does not yield any information to an attacker attempting to determine the secret or generate material that will result in share corruption. We will show this by showing that the function E does not give any information about its parameters.

Update Procedure Proof

We need to show that the operation $E(x, r) = g^x \times \mathrm{h}^{\mathrm{r}}$ does not reveal information about x. The strategy to show this is to demonstrate that by fixing each possible x input to E, all possible output values in G_q are possible for some r. This will show that for each value outputted by E, every possible x could have produced this value for some r which is unknown to the attacker. Define a mapping $q: \mathbb{Z}_q \rightarrow G_q$ as $q(r) = \mathrm{g}^{\mathrm{x}} \times \mathrm{h}^{\mathrm{r}}$ for a fixed x. We wish to show that this mapping is a 1-to-1 correspondence, a sort of permutation function of G_q. We will first show that both sets are of the same size. Then we will show that the mapping is onto. Trivially $\mathbb{Z}_q$ is of size q. To determine the size of G_q, we first note that $\mathbb{Z}^\times_p$ is a multiplicative group of prime power and is thus cyclic. The fundamental theorem of cyclic groups states that for each divisor d of the order of a cyclic group, there exists a unique cyclic subgroup of order d. Since q is a divisor of $p-1$, the order of $\mathbb{Z}_p^\times$, we can define G_q be the unique such subgroup of order q that is proven to exist.

To show the mapping is onto, we need to demonstrate that for every $y \in G_q$, there exists $r \in \mathbb{Z}_q$ such that $q(r) = \mathrm{y}$. First we observe that h is a generator of G_q by construction. This means that we can produce any element of G_q by taking h to some power. Similarly, g^x which is fixed, is also some member of G_q that can be expressed as h^z for some $z \in \mathbb{Z}_q$. So we can express $q(r)$ by $q(r) = \mathrm{h^z} \times \mathrm{h^r} = \mathrm{h^{z+r}}$, which is simply some member of G_q. Since we can generate any value of $\mathbb{Z}_q$ to be the sum of $z + r$ by picking certain r, we can generate any element of G_q as h is a generator.

We have shown that by picking certain r, $q(r)$ can construct any member of G_q, thus $q(r)$ is onto. Since $q(r)$ is onto and the codomain and domain are of the same size, $q(r)$ is a 1-to-1 correspondence. A similar proof can be done to show that $E(x, r)$ does not reveal any information about r. Since none of the public information reveals insight into determining any of the private information, the verification process does not weaken the scheme, but does prevent a certain denial-of-service attack.

Implementation

Communication

The biggest issue with implementing this OTP scheme is the communications required. The client wishing to be authenticated has to give some information to the authenticator between each use. This is not desirable as OTPs primary use case is as a personal identification number for bank machines and other such physical authentication scenarios where communication is limited. One way an implementation may deal with this issue is by generating a finite number of shares by following the protocol above, and having the client wishing to be authenticated and the authenticator store all OTPs and use the current OTP whenever authentication is desired. This is not unlike older methods of writing sequences of numbers or words on a sheet of paper and crossing of used passwords. The information theoretic advantages of the scheme still remain intact as an eavesdropper is unable to predict the next OTP by seeing previous OTPs, unlike pseudo-random number generators or other OTP schemes.

As with most cryptosystems, distribution of secret keys is difficult. The same problems face the proposed one-time password system. The secret material is the shared key, initial share, and any other information needed for the verification process. Secure methods of secret material distribution would be best achieved if accomplished by using physical contact to authenticate the user who will use the system. This kind of situation would be common in a corporate environment where the one-time password solution is utilized for physical access. If the distribution of the secret material were to happen remotely or with poor authentication techniques of the user, then the whole system could be compromised. Even if face-to-face or voice-based authentication mechanisms are used, attackers could use social engineering techniques and care must be taken.

Storage Requirements

Avoiding communication by storing many OTPs can cause storage requirement issues, especially on limited resource devices such as smart cards. The amount of storage required is dependent on the OTP size and the number of stored OTPs for future use. Since these numbers are finite and fixed at system setup, the amount of storage required is known in advance and simple to compute. In fact, the amount of storage as OTPs are used decreases; however this does not appear to be advantageous to limited resource devices. The amount of storage required is the size of the password multiplied by the number of passwords. 128 bits is often used as a strong key space to prevent brute force attacks [15]. For one authentication per day for a year, $128 \times 365 = 46{,}720$ bits, or 45.5 kibibits, of storage are required. If 3 128 bit OTPs were generated daily for 80 years, only 10.7 mebibytes of storage are required.

If communication is possible and the storage of keys is not required, only nominal space is required for the share, p, q, g, h, and other similar information. Unfortunately a device in this class of capabilities in terms of communication is probably less resource constrained in terms of storage.

Computational Requirements

The OTP scheme consists of polynomial evaluation, polynomial interpolation, and basic arithmetic operations. The evaluation of polynomials using Horner's method uses n multiplications and n additions for an n-degree polynomial [16]. Polynomial interpolation such as Lagrange interpolation is not computationally expensive, but since the proposed one-time password scheme only involves interpolating and evaluating lines, Lagrange interpolation is not required. To construct a more efficient interpolation method for lines, manipulate the point-slope formula $y - y_1 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \times (x - x_1)$. The values x_1, x_2, and x are always known as $1, 2$, and 0 respectively. By plugging in the values for these variables, the formula to evaluate the line represented by two points at 0 becomes $f(0) = 2 \times y_1 - y_2$, which consists of only three addition operations. This formula was used to construct Table 1 above.

After initial setup, the use of the one-time password requires:

Process	Operations Required		Performed By	
	Addition	Multiplication	Authenticator	Generator
Polynomial Interpolation	3	0	X	
Polynomial Evaluation	2	2	X	X
Polynomial Addition	4	0	X	X

Table 2 : Computational requirement breakdown

The addition and multiplication operations to support the proposed scheme, however, must take place on numbers larger than most current machine's native word sizes, often 128 bits or more each. If verifiable secret sharing is used to authenticate the update material, then more computation is required. The polynomials $r(x)$ for both the authenticator and generator need to be evaluated once, commitments to E_0 and E_1 computed, and the update material verified. These values differ for the authenticator and generator.

Process	Reason		Operations Required		
	Generator	Authenticator	Addition	Multiplication	Exponentiation
Evaluation	$r(2)$	$r(1)$	2	2	0
Commitment	$E_0 = E(x^t, r)$	$E_0 = E(x^t, r)$	0	2	4
	$E_1 = E\left(\delta^t_{1,1}, r_1\right)$	$E_1 = E\left(\delta^t_{2,1}, r_1\right)$	0	2	4
Verification	$E\left(\delta^t_1(i), r(i)\right)$	$E\left(\delta^t_1(i), r(i)\right)$	0	2	4
	$E_0 + E_1$	$E_0 + E_1^2$	2	4	1
Total			4	12	13

Table 3 : Requirements for computing update shares when verification is used.

Table 3 shows that even with verification, the amount of computation is small and does not require sophisticated operations.

Random Source and Distribution

The proposed one-time password solution frequently requires random number for polynomial construction. During initialization, one random number is needed to construct the secret sharing polynomial. During update, two random numbers are needed to construct the update polynomials for the authenticator and generator. If the update material is verified, two more random numbers are required per round and two more random numbers at initialization time. So there needs to be at most $3 \times n$ bits of random information at initialization and $4 \times n$ bits per round for an n bit password system. As with all cryptographic systems, if the random number source is predictable in any way, then the security of the system is compromised.

Comparison to other OTP solutions

Disadvantages

Communication

During the update phase of the proposed OTP solution, the authenticator and the entity being authenticated need to exchange some update material. This can be implemented either interactively by having the authenticator and authenticated entity communicate over a bidirectional channel or by having the update material generated during the initialization phase and stored for later use.

Sometimes it is infeasible to require a bidirectional communication. An example of this situation could be hardware tokens that do not have any input mechanism or networking capabilities. If the hardware token were a programmable smart card or a computer program, this is less of a problem as input mechanisms are available in these situations.

If the update material was generated at initialization time, then the OTP solution is only valid for a predetermined number of uses. S/KEY also has a finite lifetime before a re-initialization period is required. The specification requires that some secure method of re-initializing the secret be establishes

[7]. If such a re-initialization method is provided, there still must be storage for the pre-computed OTPs because, unlike S/KEY, all OTPs are needed in advanced.

The amount of material that must be exchanged between each use depends on the size of the secret material. Each round must exchange the password, the update material, and the verification material. The password, update material, and some of the verification material is all of the same size. However, some of the verification material, namely E_0 and E_1 can be bigger as they are restricted to being less than p in size instead of q. Since p and q are chosen, the amount of difference can be controlled. Assuming they are all less than n bits in size, the scheme takes no more than $n \times 4$ bits of information in each direction, $n \times 8$ bits of information in total each round. For normal symmetric security bit sizes such as 128, this is 128 bytes per round.

Advantages

Security

The proposed one-time password scheme is unique among one-time password schemes because of the information theoretic properties. Most schemes typically use a keyed-hash or encryption algorithms on a moving factor, typically a counter or timestamp. This technique does not provide security against an attacker with unlimited computational power who could use the unlimited computational resources to determine the moving factor by finding collisions with the hash algorithm or decrypting the password. In either case, the secret material used to generate the password can be discovered and future valid passwords can be generated. By using Shamir's secret sharing techniques, an eavesdropper gains no knowledge of the secret material that the OTPs are based upon. Even with unlimited computational power and previously utilized passwords, an attacker is unable to generate passwords that will be valid in the future. The only attack on a future password is brute force, trying all possible passwords. This kind of attack is very noisy and easily detectable by the authenticator. Systems such as SSH and other remote logon systems prevent this by disallowing so many authentications in a set time period, and even preventing the offending IP address from ever logging in.

The security is similar to one-time pads with a subtle difference, when attempting to decrypt a ciphertext that was encrypted with a one-time pad, each possible decryption is likely. The same is true for each future one-time password; however, the authenticator will tell an attacker if his guess is correct or not. Attackers of one-time pads would have to guess based on the decryption whether the correct key was used.

The S/KEY and mOTP schemes use cryptographic hash functions (CHF) as their building blocks. CHFs are designed to prevent collisions, pre-image, and second pre-image attacks often by building upon hard problems. This means that the output of a CHF reveals some information about the message being hashed. Taking advantage of this leaked information is considered difficult now, but techniques to determine this obscured secret may exist in the future. One-time passwords that rely on CHFs for security in turn are revealing information that could be used to construct valid passwords for future use.

HOTP and TOTP are OTP solutions that are built upon Hash-based Message Authentication Code (HMAC), a message authentication code built upon a CHF in such a way to mitigate extension attacks. Despite HMAC's proven resistance to many modern attacks upon the CHF in which it is based, HMACs do leak information about the underlying method used in the construction of the HMAC [17].

Computation

The proposed OTP scheme is very computationally inexpensive and readily available to install on smart cards and other resource constrained devices. Although has functions are designed to be extremely efficient, they still often require many rounds of inexpensive operations. Public key systems that are used in authentication are extremely expensive computationally and require many bits of information.

As discussed earlier, the proposed scheme is computationally inexpensive and suitable for use in resource constrained devices.

Future Work

The proposed password system is a first in the area of utilizing secret sharing techniques as an authentication mechanism. There are variations to this scheme such as multiple participants using multiple shares to authenticate a group that require additional research.

A hurdle to this password scheme adoption is implementation. The appendix outlines current implementations of the scheme, but there needs to more implementations that will be adopted by large corporations for the scheme to become a success. Future implementations may wish to consider methods to improving the ease of use of the system. For instance, a current transaction requires typing multiple unique hexadecimal numbers. This is not comfortable for most people. Translating the passwords and other numbers into words as S/Key does may be worth looking into. As with any authentication scheme, it should be well defined in a standard or presented in a manner to comply with existing standards to promote multiple implementations that successfully operating together.

Conclusion

A one-time password scheme based upon techniques developed for secret sharing techniques has been presented. The security of the system has been shown to be information-theoretically secure. This means that by seeing previous rounds of the password system, an adversary seeing all public information is unable to reduce the set of future possible passwords. This contrasts with previous password systems where a computationally unbounded adversary is able to determine future passwords. The computation and storage requirements of the passwords system have been shown to be low enough for resource constrained devices to implement such a system, which is typical for one-time password systems that are often implemented on a dedicated hardware token. Multiple implementations of the system are described in the appendix and are demonstrations of the systems practicality.

Bibliography

1. **Schneier, Bruce.** Two-Factor Authentication: Too Little, Too Late. *Inside Risks.* February 18, 2005, p. 27.

2. *Fourth-factor authentication: somebody you know.* **Brainard, John, et al.** New York, NY, USA : ACM, 2006.

3. What is an Identification Code? *Chase.* [Online] JPMorgan Chase. [Cited: March 18, 2011.] https://www.chase.com/psmhelp/index.jsp?pg_name=ccpmapp/shared/help/page/EandA_activation_code_HELP.

4. Advanced sign-in security for your Google account. *The Official Google Blog.* [Online] Google Inc., February 10, 2011. [Cited: March 03, 2011.] http://googleblog.blogspot.com/2011/02/advanced-sign-in-security-for-your.html.

5. *Cryptanalysis of the Alleged SecurID Hash Function (extended version).* **Biryukov, Alex, Lano, Joseph and Preneel, Bart.** s.l. : Springer-Verlag, 2003, Selected Areas in Cryptography, Proceedings SAC 2003, LNCS 3006, pp. 130-140.

6. **M'Raihi, D, et al.** TOTP: Time-based One-time Password Algorithm. *The Internet Engineering Task Force.* [Online] September 8, 2010. [Cited: November 29, 2010.] http://www.ietf.org/id/draft-mraihi-totp-timebased-06.txt.

7. **Haller, N, et al.** A One-Time Password System. *The Internet Engineering Task Force.* [Online] February 1998. [Cited: November 29, 2010.] http://tools.ietf.org/html/rfc2289.

8. **M'Raihi, D, et al.** HOTP: An HMAC-Based One-Time Password Algorithm. *The Internet Engineering Task Force.* [Online] December 2005. [Cited: November 29, 2010.] http://www.ietf.org/rfc/rfc4226.txt.

9. *How to share a secret.* **Shamir, Adi.** 11, New York : ACM, November 1979, Communications of the ACM, Vol. 22, pp. 612-613.

10. DNSSEC Root Key Split Among Seven People. *Schneier on Security.* [Online] July 28, 2010. [Cited: March 03, 2011.] http://www.schneier.com/blog/archives/2010/07/dnssec_root_key.html.

11. *A practical verifiable multi-secret sharing scheme.* **Zhao, Jianjie, Zhang, Jianzhong and Zhao, Rong.** 1, 2007, Computer Standards & Interfaces, Vol. 29, pp. 138-141. 0920-5489.

12. *A Practical Scheme for Non-interactive Verifiable Secret Sharing.* **Feldman, Paul.** Washington, D.C. : s.n., 1987, SFCS '87: Proceedings of the 28th Annual Symposium on Foundations of Computer Science, pp. 427-438.

13. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing.* **Pedersen, Torben.** [ed.] Joan Feigenbaum. s.l. : Springer Berlin / Heidelberg, 1992, Advances in Cryptology - CRYPT0' 91, Vol. 576, pp. 129-140. 10.1007/3-540-46766-1_9.

14. *Proactive Secret Sharing Or: How to Cope With Perpetual Leakage.* **Herzberg, Amir, et al.** [ed.] Don Coppersmith. s.l. : Springer Berlin / Heidelberg, 1995, Advances in Cryptology — CRYPT0' 95, Vol. 963, pp. 339-352. 10.1007/3-540-44750-4_27.

15. **National Institute of Standards.** *Federal Information Processing Standards Publication 197.* s.l. : Federal Information Processing Standards, 2001. p. 51. 197.

16. *Algorithms: Algorithm 337: calculation of a polynomial and its derivative values by Horner scheme.* **Pankiewicz, W.** 9, New York : ACM, September 1968, Communications of the AcM, Vol. 11, p. 633. 0001-0782.

17. *On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1.* **Jongsung, Kim, et al.** s.l. : Springer Berlin / Heidelberg, 2006, Vol. 4116.

Appendix

C Library

A library implementing the proposed one-time password solution has been developed in hopes of promoting its use. The library works on the basis of profiles. Each profile corresponds to a participant of the authentication system, so every user has a profile that represents a relationship between a certain authenticator and that user. If the user authenticates with more than one authenticator, they will have multiple profiles. The same holds true for the authenticator. The authenticator has a profile for each user that they authenticate. The library creates these profiles and loads them to compute the password, verify the password, update the shares, and verify the shares. Documentation of the library is provided digitally.

The library comes with example programs designed to be run as command line applications for simple experimentation. A simple usage of these command line applications is as follows:

- A user would be given a profile generated by the authenticator:
    - `./shamir-create-profile` *`user-profile`*
- The authenticator would generate his share also:
    - `./shamir-load-profile` *`user-profile authenticator-profile`*
- The user could generate a password:
    - `./shamir-generate-otp` *`user-profile`*
- The authenticator could check the password :
    - `./shamir-check-otp` *`authenticator-profile otp`*

The last two commands require communication of the update and verification material. This is done via standard input.

PAM Module

To allow authentication via native applications already written, a PAM module was written that utilizes the C library described above. By providing a PAM module, applications such as login in environments that use PAM are capable of utilizing the proposed scheme to authenticate users.

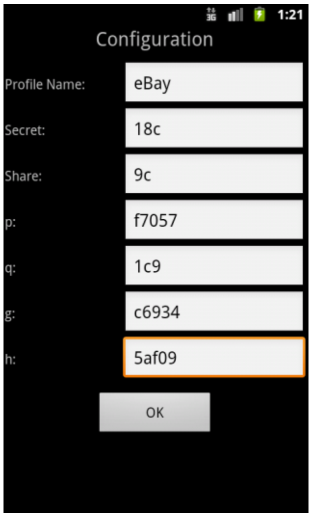
To use the PAM module, the pam module must be compiled and installed properly. Compilation generates a shared library that in most distributions should be placed in `/lib/security`. This process should be taken care of by the build system. Once properly installed, the application that wishes to use the one-time password scheme for authentication must edit a configuration file specific for that application that PAM uses. These configuration files are usually located in `/etc/pam.d/`*`application`* where application is the name of the application. The file should include a line that tells PAM to use the module.

```
auth required shamir_otp
```

For each user that wishes to be authenticated with the system, a configuration file must be placed in `/etc/init.d/`

Android Application

Since most one-time password solutions require a hardware token, time to market is often very long. To reduce the difficulty of introducing this one-time password system to a user base, an Android application was written. Android is a mobile phone operating system that has become very popular recently. The application allows the hardware token to be replaced by the user's cell phone or other Android device while still satisfying the something you have property of multi-factor authentication. The application is straight forward to use, but is unable to provide the convenience of loading profiles from files as the C library is. This just means that the user has to manually enter the contents of this file; this is demonstrated in Figure 2. The actual presentation of the password from the generator to the user is demonstrated in Figure 3.



Profile Name: eBay

Secret: 18c

Share: 9c

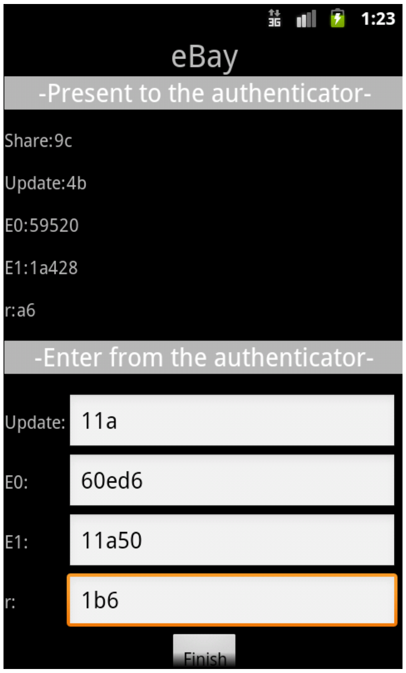
p: f7057

q: 1c9

g: c6934

h: 5af09

Figure 2: The preparation of the generator for use in the password system.



-Present to the authenticator-

Share:9c

Update:4b

E0:59520

E1:1a428

r:a6

-Enter from the authenticator-

Update: 11a

E0: 60ed6

E1: 11a50

r: 1b6

Figure 3: The generation of a password along with update and verification material.

Glossary

Android - A mobile operating system based upon Linux and developed by Google Inc.

Hash chain - A successive application of a cryptographic hash function.

HMAC - Hash-based Message Authentication Code. An algorithm that computes a message authentication code by utilizing cryptographic hash functions and a secret key.

Homomorphic encryption - A form of encryption where a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext.

HOTP - HMAC-based One-Time Password. A one-time password algorithm specified by OATH that utilizes a message authentication code and a counter to construct passwords.

Lagrange polynomial - The least degree polynomial that interpolates points using a technique named after Joseph Lagrange.

Lamport's scheme - See S/Key

MAC - Message Authentication Code. An piece of information used to authenticate and verify the integrity of a message. MACs do not provide non-repudiation.

MFA - Multifactor authentication. Authentication that happens on two or more independent factors of authentication.

mOTP - Mobile One-Time Password. A one-time password algorithm designed to be implemented on mobile devices.

OATH - Open Authentication. A group that publishes documents specifying authentication mechanisms.

One-Time Password - A temporary password that is often used in a multi-factor authentication system.

PAM - Portable Authentication Module. A software library that allows applications to hand off the process of authentication to a dedicated library.

Polynomial interpolation - The process of constructing a polynomial which goes exactly through given points.

Proactive secret sharing - Secret sharing that utilizes certain techniques to allow shares to be updated in such a way that prevents non-updated and updated shares from reconstructing the secret.

S/Key - A one-time password system developed for dumb terminals that uses a hash chain to construct passwords.

Secret sharing - The act of creating multiple shares from an initial secret in such a way that the shares can be combined to generate the secret.

Share - A portion of the secret that under some circumstances with other shares reconstruct that secret.

Threshold - The number of shares in a secret sharing scheme that are required to reconstruct the secret.

TOTP - Time-based One-Time Password. A one-time password algorithm specified by OATH that utilizes a message authentication code and the current time to construct time-based passwords.

Verifiable secret sharing - Secret sharing that utilizes certain techniques to allow share receivers to verify that the shares they are receiving are not corrupt or maliciously crafted.

Vita

The author was born in Covington, Louisiana. He obtained his Bachelor's degree in computer science from Louisiana State University in 2009. He joined the University of New Orleans computer science graduate program to pursue a Master's in Information Assurance and became a member of Dr. Richard's research group in 2010.