

Duress Detection for Authentication Attacks Against Multiple Administrators*

Emil Stefanov
UC Berkeley
Berkeley, CA

emil@cs.berkeley.edu

Mikhail Atallah
Purdue University
West Lafayette, IN

mja@cs.purdue.edu

ABSTRACT

An authentication system is duress-resistant if it allows a user or system administrator to covertly send a silent alarm during the login process, indicating that they are being forced to authenticate against their will. The adversary knows that the system has this feature, e.g., if two passwords are used (one normal and one duress) then the adversary will demand from a victim both passwords. We require that the adversary is not able to distinguish a non-cooperating victim from a cooperating victim, even if there are multiple victims some of whom cooperate while others do not. To avoid a false alarm, we also require that the probability of a user accidentally sending a duress signal (e.g., through typos) is small. After arguing that existing techniques are inadequate for such requirements, we present our design and implementation of a duress-resistant authentication system that can be used by any number of administrators and users. Our system is compatible with existing authentication systems, and can be implemented as an augmentation of their capabilities that does not require modification of their internals.

Categories and Subject Descriptors

D.4.6.b [Operating Systems]: Security and Privacy Protection – Authentication.

General Terms

Security, Algorithms, Human Factors, Management.

Keywords

Duress, Panic Password, Authentication, Multiple Administrators, Privacy, Non-Linkability.

* Portions of this work were supported by National Science Foundation Grants CNS-0915436, CNS-0913875, and Science and Technology Center CCF-0939370, by Grant FA9550-09-1-0223 from the Air Force Office of Scientific Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

InsiderThreats'10, October 8, 2010, Chicago, Illinois, USA.
Copyright 2010 ACM 978-1-4503-0092-6/10/10...\$10.00.

1. INTRODUCTION

A duress-resistant authentication system allows a user to trigger a silent alarm during the login process, indicating that they are being forced to authenticate against their will. The system is designed such that even though the adversary knows about this feature, he has no way of detecting the alarm.

Although academic research in this area is very limited, there are already several duress-resistant authentication systems used in practice by US government agencies [1], some ATM's, and are even an option on the popular BlackBerry smartphones [5]. These kinds of systems also have the potential to be used by banks (to prevent robberies inside the bank or online) and corporations with sensitive data such as defense contractors.

In this paper, we first categorize and review some of these existing systems and show that many of them have serious security issues. We then define our threat model and present two duress authentication schemes. The first scheme can only protect a single administrator from being coerced, whereas the second one does not have this restriction.

We stress that there is a major difference between a normal user and an administrator being attacked when duress detection is present. Administrators by definition are privileged and may have access to the duress authentication system and its password store. After an attacker coerces an administrator to log in, the attacker has full administrator access. At this point, the system must be in such a state that it is impossible to determine if the victim provided real or duress credentials. In other words, even with full access to the authentication system and its data after authentication, the attacker must not be able to determine if the victim lied. The problem is further complicated if there are two or more administrator victims and the attacker can use the credentials of one to verify the duress credentials of the other.

Our **contributions** for both schemes can be summarized as follows:

- **Duress Detection:** We specify an authentication system that guarantees the user the ability to trigger a silent alarm during authentication without the attacker being able to detect it (even though he knows the system has this ability).
- **Usability:** The scheme is designed to be easy to use and only requires users to remember a single word in addition to their existing password.
- **Avoiding Typos:** Our schemes use a carefully chosen keyword dictionary to avoid false alarms caused by typos.
- **Authentication Privacy:** A third party monitoring company that inspects authentications for duress signals is not able to

determine the identity of the user unless a duress signal is in fact present.

- **Stronger Security:** A much lower probability that the attacker eludes the alarm than many existing systems such as the BlackBerry.
- **Backwards Compatibility:** Our schemes allow the use of existing client software such as SSH without modification. Also, our duress authentication checking scheme can be layered on top of existing authentication systems and there is no need to modify their internals.

Organization: Section 2 provides a brief survey of existing work and points out some of the security issues by describing possible attacks. Section 3 lists the participants in the threat model with their capabilities and goals. Sections 4 and 5 describe the single and multiple administrators schemes respectively. Section 6 focuses on the keyword dictionaries that are used by our two schemes. Section 7 discusses the security and privacy aspects of the schemes SSH. Finally, in section 8 we analyze performance.

2. RELATED WORK AND BACKGROUND

There has been very little academic research done on duress detection schemes and panic passwords [1]. In the industry, there are a few patents [2] [3] [4] and commercial products [5] [6] [7] [8] that employ the use of panic passwords, but many of them are flawed because they allow the attacker to gain access to the system and bypass the alarm with high probability of success (e.g., $1/2$). These schemes are discussed next in this section.

2.1 Two-Password Schemes

Two-password schemes involve the use of two passwords: a correct password p^* , and a duress (panic) password p^d . When the user authenticates with p^* , the system grants them access and continues operating normally. When p^d is used for authentication the system still grants access but sets off the silent alarm. All other passwords result in a failed authentication attempt.

These schemes are susceptible to a simple attack that allows the attacker to authenticate with $1/2$ probability of bypassing the alarm.

Attack 1 – Two Passwords: The attacker demands two passwords from the victim p^* and p^d (he does not ask the victim to perform the authentication) and says that he will pick one at random and use it. The victim knows that if she gives the attacker a password not from the set $\{p^*, p^d\}$, he might attempt to use it and discover that it is fake because the system would not grant him access. Being afraid of violating the stealth property, she must give him the two passwords, but has a choice about whether to lie about which one is correct. Since the attacker does not know whether the victim lied or not, he chooses one of the two password at random and authenticates with it. Hence the attacker's probability of bypassing the alarm is $1/2$.

The popular BlackBerry smartphones have an option to enable something called "duress notification" [5]. This feature allows the user of the BlackBerry to signal duress by using a modified version of her correct password. Specifically, p^d is constructed by taking p^* and moving its first character to the end. Since this is a two-password scheme, it is susceptible to Attack 1. However, since users often choose passwords that are not entirely random, the BlackBerry duress scheme is often susceptible to another even more effective attack:

Definition 1 – Coherent Password: A password that does not appear entirely random. Examples: "dolphin273ball", "05031982" (a birth date), "joshuaVHz3xK*bL8". Note that a coherent password does not necessarily need to be weak (e.g., the last password example provides over 63 bits of security).

Attack 2 – Modified Password: The attacker demands to know the real password (he does not ask the victim to enter it herself). The victim must provide a password s which is equal to either p^* or p^d because any other password will cause her to lose the stealth property. The attacker constructs s' by taking the last character of s and moving it to the front (effectively reconstructing p^* if $s = p^d$). Next,

- (a) if s is coherent but s' is not then the attacker authenticates with s .
- (b) if s' is coherent but s is not then the attacker authenticates with s' and the victim loses the stealth property.
- (c) if both or neither s and s' are coherent then the attacker picks one at random and authenticates with $1/2$ probability of bypassing the alarm. If he picks s' and it works, then the victim loses the stealth property.

Example: The victim gives $s = "olphin273balld"$ to the attacker. The attacker constructs $s' = "dolphin273ball"$. Since s' is coherent and s is not, the attacker authenticates with s' and she loses the stealth property.

This attack can be generalized to any two-password scheme that uses a reversible method for constructing p^d from p^* .

Another major disadvantage of two password schemes is that the user is likely to forget her duress password, because unlike her correct password that she uses almost every day, her duress password is only needed in the rare case of an emergency.

2.2 N-Password Schemes

N-Password schemes work by assigning N strong passwords to each user, only one of which is the correct one and the rest are duress. These password schemes suffer from the same kind of problems as the 2-password scheme except that the probability of a successful attack decreases with N. However, it is undesirable to use large N because the amount of information the user needs to memorize increases linearly with N. For example, a 5-password scheme would require the user to remember 5 strong passwords, 4 of which she never uses.

2.3 PIN Schemes

PIN schemes use a combination of a strong password p^* and a four digit number q^* called a PIN. For example, a user's p^* may be "VHz3xK*bL8" and q^* may be "8394". When a user authenticates regularly (not under duress) using this scheme, she provides p^* and q^* to the system. When she is under duress, she provides p^* and t where t is any possible PIN other than q^* . An attempt to login with a password other than p^* fails and does not set off an alarm (regardless of whether the PIN is correct).

The idea behind PIN schemes is to split the user's credentials into two parts: a strong password which provides basic security and a PIN to be used for duress detection. If the user wants to provide duress information to the attacker, she only needs to lie about the PIN.

She must be able to come up with a fake PIN which she can recall later if the attacker tests her. Unfortunately, a newly fabricated

PIN number can be easily forgotten, especially in a stressful situation. Also, PIN's are very easy to accidentally mistype and can lead to many false alarms as it has been pointed out by [1].

2.4 Dictionary Schemes

Like PIN schemes, dictionary schemes use a combination of a strong password p^* and a simple weak keyword q^* . The keywords are elements of a predefined publicly known dictionary (e.g., English words or names of animals). For a normal (not under duress) authentication, the user must provide both p^* and q^* . For a duress login, she must provide p^* and t where t is any keyword in the dictionary other than q^* .

As described in section 6, our schemes use keywords from an implicitly defined dictionary that was specifically selected to avoid typos.

3. THREAT MODEL

This section defines the participants in our scheme and specifies their goals and capabilities. We have designed our authentication scheme to allow the victims and the authentication system to achieve their goals (described later in this section), and at the same time, to prevent the attacker from achieving his.

3.1 Participants

The participants in our threat model are:

- **Alice and Bob:** The victims of Oscar's attack. They can be normal users or administrators depending on the context. Bob exists only when there is more than one victim.
- **Oscar:** The attacker who wants to know the credentials of Alice and Bob. Oscar might actually consist of multiple attackers, but they will still be referred to as a single entity throughout this paper.
- **Authentication Server:** The server responsible for authenticating Alice and Bob. Depending on the scheme, this server may also be responsible for detecting duress authentications or delegating the task to the monitoring server.
- **Monitoring Server:** A server owned by a third party company that monitors the authentication server to detect duress authentications.

Depending on the context, Alice or Bob may be administrators. We now define what that means.

Definition 3 – Administrator: An administrator is a user that when authenticated has any of these capabilities:

- (a) Read from the keyword database on the authentication server that contains the hashes of the keywords of each user.
- (b) Write to the keyword database on the authentication server.
- (c) Observe the authentication server during an authentication.
- (d) Act as the authentication server (e.g., replace the authentication software).

Administrators are usually users such as "root" or the "Administrator" account in Windows. It is important to note that large organizations can have many such administrators.

3.2 Types of Authentications

Throughout this paper, we refer to different types of authentications. For clarity, we define them here:

- **Normal Authentication:** An authentication that results in the user being granted access to the system without signaling the alarm.
- **Duress Authentication:** An authentication that results in the user being granted access, but also signaling the alarm. Our schemes ensure that to the attacker, this authentication is indistinguishable from a normal authentication.
- **Failed Authentication:** An authentication that results in the user being denied access without signaling the alarm. This could happen if a user mistakenly enters the wrong password.

3.3 Attacker's Goal

In our threat model, Oscar's goal is to authenticate as Alice without setting off an alarm. This way, Oscar has enough time to carry out his attack (e.g., find the information he is looking for in Alice's account) and escape before the authorities are alerted.

3.4 Attacker's Capabilities

Our schemes assume that Oscar is very powerful. Oscar can interact with Alice and Bob and coerce them to cooperate or at least seem to cooperate. Oscar can keep them separated and ask them question to test if they contradict one another and prevent them from colluding against him. The single administrator scheme assumes that Oscar can only attack Alice, but in the multiple administrators scheme, Oscar can attack them both and any other users/administrators.

Oscar has full knowledge of how each part of the system works. Prior to the attack, he does not know the private data such as passwords, keywords, and cryptographic keys of the authentication and monitoring server. After the attack, Oscar may gain access to the private data of the authentication server and the user, but we have designed our schemes to ensure that the intrusion has already been detected and reported to the authorities.

For the purpose of the attack, Oscar has full physical and logical access to client computers which have the ability to authenticate against the authentication server. He can force Alice and Bob to use these clients to authenticate, or Oscar can use them himself after getting credentials from Alice and Bob. These clients, of course, do not contain any of the private data of Alice and Bob until after authentication.

Oscar is assumed to not have already compromised the authentication server; otherwise his attack would be pointless. The Multiple Administrators Scheme uses a monitoring server, which is owned by a company that specializes in security. Our schemes assume that Oscar has not been able to compromise the monitoring server.

3.5 Victim's Goal

If Alice is under attack, her goal is to signal the alarm during authentication while satisfying the stealth property defined below. Since Oscar might not let Alice perform the authentication herself, Alice's goal might involve tricking Oscar to authenticate as Alice with duress credentials.

Definition 2 – Stealth: The victims of the scheme (Alice, Bob, etc) are said to have the stealth property if for every authentication attempt, the attacker (Oscar) cannot distinguish between a normal authentication and a duress authentication.

It should be noted that Alice can choose to dthe rules of the schemes, and this scenario is outside of our model. However, Alice has no disincentive to break the rules because the attacker cannot detect the alarm.

3.6 Victim's Capabilities

Our schemes require very little from Alice (the victim). In the long term, Alice must be able to remember her password, just as with any other password based authenticating system. Additionally, she must remember her keyword, which in our schemes is a simple and easy to remember word (e.g., the name of a U.S. state: "California"). Since Alice needs to use her password and keyword every time she authenticates, it is very unlikely that she will forget them.

During an attack, Alice will likely be very stressed out and she can only be expected to follow very simple rules for lying to the attacker. In our schemes, the only task required of Alice is to be able to pick a new keyword from the dictionary and lie to the attacker that the new keyword is her real keyword. She must be able to remember the keyword she picked because the attacker might ask her for it again later. Since the keywords in the dictionary are designed to be very simple words, we assume that Alice will be able to perform this task.

3.7 Authentication Server's Responsibilities

The authentication server in our scheme has several goals. If Alice provides correct credentials, the server must authenticate her without setting off an alarm. If Alice provides duress credentials, the server must set off a silent alarm that is unobservable to Oscar. Of course, if the credentials provided are neither correct nor duress, the authentication server should respond the way it normally does by refusing authentication.

In the Single Administrator Scheme, the authentication server itself is responsible for notifying the authorities or someone who can notify the authorities. It must send out the alert message to a separate system, but it should be careful to make sure that the message is not logged, and so Oscar cannot use his new administrator identity to read the logs and detect the alert.

In the Multiple Administrators Scheme, the goal of the authentication server is to forward authentication information to the monitoring server and let it determine if there was a duress authentication. In order to protect multiple administrators from being attacked, the server must not be able to distinguish between a correct and duress authentication.

Since the Multiple Administrators Scheme uses an external monitoring service to verify its authentications, privacy is naturally a concern. The organization which owns the authentication server might not be comfortable with releasing information about which, when, and how often each user authenticates. Our Multiple Administrator authenticate scheme provides complete privacy about each authentication except when it is a duress authentication. This way, the monitoring server learns the identity of the user/administrator **only** when he/she is being attacked. It should be noted that the authentication server is actually able to provide this selective privacy without knowing if the user/administrator is being attacked.

3.8 Monitoring Server's Responsibilities

The monitoring server is administered by a third party monitoring company. It is responsible for inspecting authentications made to the authentication server and distinguishing between correct and

duress credentials. The monitoring server is used in the Multiple Administrators Scheme, but not in the Single Administrator Scheme.

When the monitoring server detects a duress authentication, it is responsible for immediately notifying the monitoring personnel that Alice is under attack. The monitoring personnel then are responsible for explaining to the authorities who is being attacked and what the circumstances are. It is important to note that the cryptographic properties of the scheme ensure that the monitoring server and personnel learn the username (Alice) and her information **only** when there is a duress authentication.

One monitoring server can service many authentication servers belonging to many different organizations and hence this service can be provided by third party security companies.

4. SINGLE ADMIN SCHEME

This section describes a simple authentication scheme that allows duress detection when a single administrator is under attack. In section 7, we show that this scheme fails when more than one administrator is under attack. For this reason, section 5 describes a scheme that handles multiple administrators being attacked.

The Credentials: Alice's credentials consist of her username u^* , a password p^* , and a keyword q^* . The keyword q^* is an element from the authentication system's publicly known keyword dictionary. See section 6 for a discussion about which dictionaries are appropriate.

4.1 Account Creation

Creating a new account is straightforward. Alice should choose a strong password p^* as usual and the system should assign her a random keyword q from the dictionary. Alice must not be allowed to choose the keyword herself. This restriction ensures that the keyword is picked uniformly randomly from the dictionary. For example, if the dictionary is the set of U.S. states and if Alice lives in California, she might be more inclined to pick California as her keyword, so she should not be allowed to make that choice.

Once Alice creates her credentials $\{u^*, p^*, q^*\}$, they should be sent to the authentication server over an encrypted connection such as an SSH session. The authentication server should store salted hashes of the password and keyword as follows:

$$\begin{aligned}h_{p^*} &:= H(\text{salt} \parallel p^*) \\ h_{q^*} &:= H(\text{salt} \parallel p^* \parallel q^*)\end{aligned}$$

where H is a strong cryptographic hash function such as SHA-256. This is essentially the same way existing authentication systems store passwords. The idea is that h_{q^*} must not just simply be $H(q^*)$ or $H(\text{salt} \parallel q^*)$ because q^* is a word from a dictionary of a very small size (e.g., less than 100 words) and can be easily inferred by a small brute force attack. Also, salt can be reused between h_{p^*} and h_{q^*} for efficiency.

4.2 Authentication

When Alice is not under duress, she uses her normal credentials $\{u^*, p^*, q^*\}$ to authenticate. When Oscar attacks Alice, he threatens her and demands that she give him her real credentials. Under this scheme, Alice has been instructed to lie to Oscar by randomly choosing a keyword t from the dictionary such that $t \neq q^*$, and giving Oscar $\{u^*, p^*, t\}$ as her credentials.

Example: Alice's credentials are $\{u^*, p^*, q^*\} = \{\text{"alice283"}, \text{"W$PDz7Bmdf"}, \text{"texas"}\}$. When Oscar attacks Alice, Alice

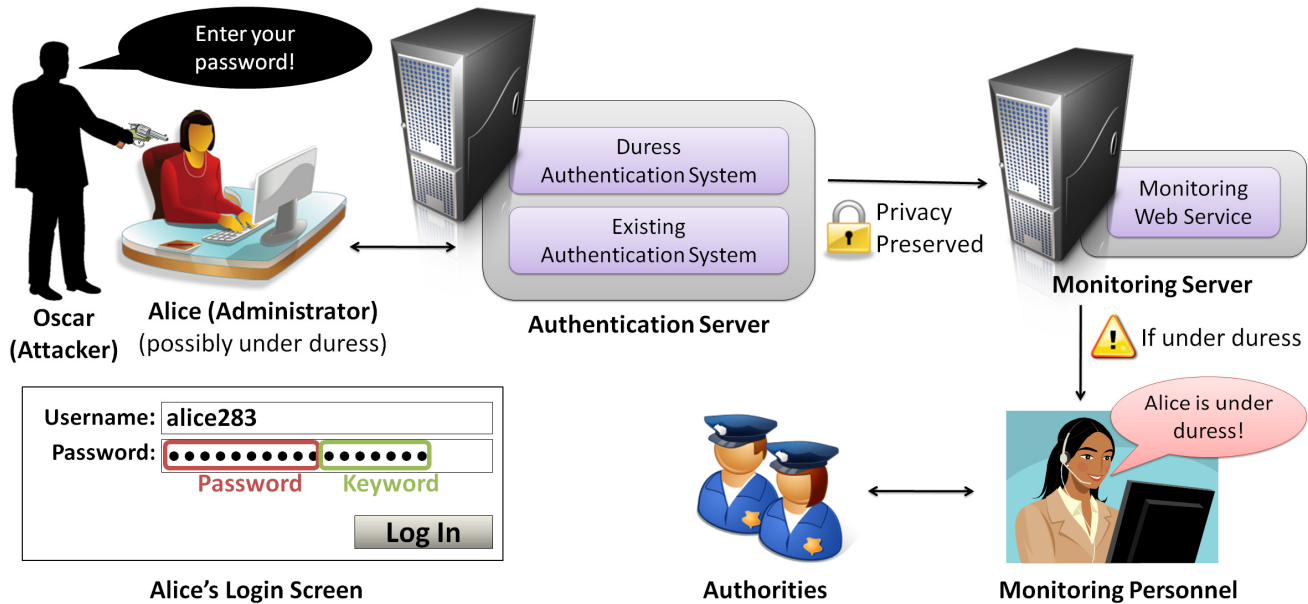


Figure 1: An illustration of the multiple administrators scheme.

randomly chooses $t = \text{"montana"}$ and lies to Oscar that her credentials are $\{u^*, p^*, t\} = \{\text{"alice283"}, \text{"W$PDz7Bmdf"}, \text{"montana"}\}$.

Oscar has no way of telling if Alice lied or not.

When the authentication server receives Alice's credentials $\{u, p, q\}$, it computes $h_p := H(\text{salt} \parallel p)$ for all values of salt observing if there is a value for salt where $h_p = h_{p^*}$.

If such a value for salt does not exist, then the password is wrong and the authentication fails (without an alarm). The client is receives the message "incorrect username or password".

If such a value for salt does exist, then the authentication server uses that value of salt to compute $h_q := H(\text{salt} \parallel p \parallel q)$ and checks to see if $q = q^*$ by testing whether $h_q = h_{q^*}$. The authentication server triggers a silent alarm if $q \neq q^*$. However, if $q = q^*$, the authentication server should wait for as long as it would otherwise take to trigger a silent alarm so that Oscar cannot determine if the silent alarm was triggered by timing the authentication.

If $q^* \neq q$, the server also updates Alice's credentials by setting $h_{q^*} := h_q$. This is done in order to fool Oscar when Alice is an administrator because after authentication, Oscar might use Alice's administrator permissions to get the value of h_{q^*} to check if $h_q = h_{q^*}$.

Afterwards, the authentication system should notify the client that the authentication succeeded and grant it access. The client cannot determine if the authentication triggered a silent alarm.

4.3 Changing Credentials

Changing the password and keyword should be done just like when creating an account as described in section 4.1. However, the authentication server must first ask the user to authenticate before allowing him/her to change their own password and keyword. In the case of an administrator changing another user's password, the administrator must of course be authenticated. By first forcing authentication, the scheme ensures that any attacker attempting to change credentials will be subjected to duress detection.

5. MULTIPLE ADMINS SCHEME

Unlike the single administrator scheme, the multiple administrators scheme can protect multiple administrators that are being attacked at once. This section defines the behavior of the client, the authentication server, and the monitoring server. It also specifies how the user should behave in different scenarios. Figure 1 and Figure 2 help illustrate the scheme.

5.1 Account Creation

The CreateAccount pseudo code is shown in the appendix. The following summarizes and explains it.

We wish to create an account with the parameters $\{ \text{username}, \text{password}, \text{keyword} \}$. CreateAccount performs some basic checks to ensure that an account with this username does not already exist and that keyword is valid (it is in the dictionary). Then it calls CreateRegularAccount which performs all of the account creation steps of normal non-duress authentication system such as creating a user profile and adding the username to the list of users. Afterwards, the SetAccountKeyword function is called.

SetAccountKeyword first generates a random key (called *secret*) for a strong symmetric encryption algorithm such as AES. This key will be later used during an authentication in order to encrypt user information for the monitoring server. The details are described in the authentication section. The scheme uses Shamir's secret sharing scheme to split the value of *secret* in two parts. The function CreateShamirPolynomial generates the polynomial coefficients used by Shamir's scheme and stores it as *account.Polynomial*.

Next, SetAccountKeyword picks a random integer mod *KEYWORD_COUNT* (the number of keywords in the dictionary) storing it in *account.SecretIdOffset* and then gets the ID of the keyword (its index in the dictionary). *account.SecretIdOffset* is then used to randomly offset the value of *keywordId* and the result which is stored as *partialSecretId*. *account.SecretIdOffset* will also later be used during authentication to offset the ID of the keyword the user is attempting to authenticate with. The purpose of the offset is to make sure that the monitoring server cannot use *partialSecretId* to infer the keyword. Afterwards, the function

GenerateShare uses *partialSecretId* and *account.Polynomial* to generate a share of *secret* with the ID *partialSecretId*.

Lastly, the secret share and its ID are encrypted with the monitoring server’s public key so that only the monitoring server can decrypt them. Note that they are not saved and hence the authentication server purposely loses their unencrypted values. However, the encrypted values are stored as *account.EncryptedIdAndPartialSecret* and will be sent to the monitoring server during authentication.

5.2 Authentication

When authenticating, the user should be instructed to behave the same way as in the single administrator scheme. The pseudo code for authentication is shown in the appendix. The following summarizes and explains it.

The purpose of the function *Authenticate* is to determine if the username and password provided are correct (regardless if the keyword is incorrect), and if so to notify the monitoring server of this authentication. The monitoring server will be responsible for determining if it is a duress authentication. In fact, the authentication server has no way of telling if the authentication was duress.

The function *Authenticate* first checks to see if the username and password are correct. If not, the authentication immediately fails. Then it obtains the ID of the keyword provided by the client (the ID is its index in the dictionary). Next, a random offset (which was chosen the last time *SetAccountKeyword* executed) is added to the ID to get the partial secret ID. The purpose of the offset is to make sure that the monitoring server cannot infer the keyword. Shamir’s secret sharing scheme is used to generate a share of the secret with the ID corresponding to the keyword the client is authenticating with.

Since the authentication server does not know if the login is duress, it always prepares an authentication record for the monitoring server in case of a duress login. This record contains information that would be useful to the authorities in case of duress. As an example, in the pseudo code this information consists of the username, the user’s first/last name, and their address. It’s important to note that the size of the information for any user must not exceed a fixed pre-defined threshold *MAX_USER_DATA_LENGTH* so that the authentication record can be padded to that length to prevent the monitoring server differentiating between users based on the record lengths. Then, the record is encrypted with *account.Secret* and sent to the monitoring company. It might seem contradictory that we send information to the monitoring company but prevent the company from inferring its contents, but this is done because the monitoring company should only know the contents of the record if the keyword used during authentication is incorrect.

Lastly, we run *SetAccountKeyword* to generate a new secret key and polynomial. This is done so that if the user/administrator authenticates again, the authentication record sent to the monitoring server is unlinkable to this one. This prevents the monitoring server from distinguishing between the same user authenticating twice and two different users each authenticating once. We call this property *authentication privacy* and prove it in section 7.1. Since the server does not know the correct keyword for the account, it uses the one provided by the client, which will be wrong during a duress authentication. That is not a problem because the authentication record sent to the monitoring server

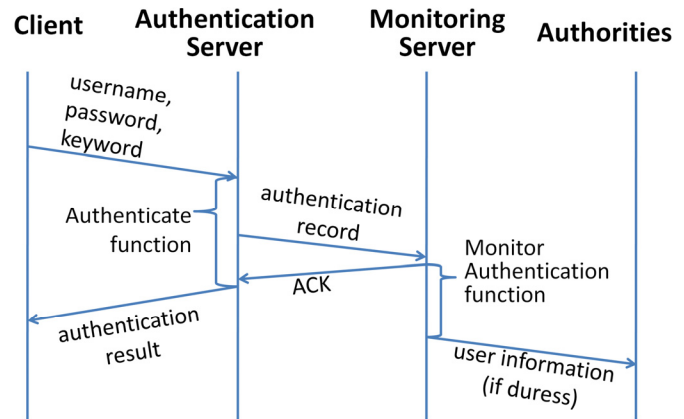


Figure 2: Sequence diagram of the multiple admin scheme.

already signaled the alarm. The authentication server waits to receive an ACK or retransmits the authentication record to make sure it gets to the monitoring server. Note that this ACK is just an acknowledgement of the receipt of the record and does not indicate if there is duress.

5.3 Changing Credentials

Changing the password for an account should be done through the existing normal authentication system and changing the keyword can be done by executing *SetAccountKeyword* as shown in the appendix. Just like in the single admin scheme, the system should only let a user change their password after they authenticate. In the case of an administrator changing another user’s password, the administrator must of course be authenticated.

5.4 Authentication Monitoring

The monitoring server should provide a web service that can receive the authentication records sent by the authentication server. The pseudo code for the monitoring server is shown in the appendix as the function *MonitorAuthentication*.

This function is responsible for inspecting an authentication that was performed by the authentication server and determining if it is duress. If it is duress, then the monitoring server will be able to determine which user is under attack and notify the authorities (e.g., the police). If the authentication is not duress, then the monitoring server will not have enough information to infer which user is involved in the authentication.

The basic idea is that the monitoring server receives two shares of a secret key and information about the user symmetrically encrypted using that key. The secret sharing is done using Shamir’s secret sharing scheme. Each share has an ID (the *x* value of the Shamir polynomial) and each ID is uniquely mapped to a keyword. The ID of the first share corresponds to the correct keyword for the user while the second ID corresponds to the keyword provided by the client during authentication.

When the authentication is not duress, the two keywords match and hence so do the share ID’s. Shamir’s secret sharing scheme requires two shares with distinct ID’s in order to be able to reconstruct the secret. Therefore when the authentication is not duress, the monitoring server does not have enough information to reconstruct the secret key and cannot decrypt the user information.

When the authentication is duress, the two keywords are different and hence so are the share ID’s. This allows the monitoring server two use the two shares to reconstruct the secret key using

Shamir’s secret sharing scheme. Then, the monitoring server decrypts the user information and learns which user is under attack. Lastly, it reports this information to the authorities.

It is possible to replace the monitoring company in the multiple administrators scheme with separate in-house server that is dedicated to monitoring authentications. No one (including administrators) should have non-physical access to it and it will essentially act as a black box. Because of its limited functionality and because it does not need to store any user data (such as usernames, passwords, and keywords), the in-house server will rarely need to be accessed by an administrator and hence the physical access limitation would not be a problem. Although eliminating the need for the monitoring company sounds good, we advise against eliminating it for the following reasons:

- (1) The in-house server would need to have a direct connection to the internet, otherwise the attacker (with his newly acquired administrator privileges) can sniff the network and determine if the alarm is triggered on subsequent logins. He can also configure the firewalls to block the distress signal from leaving the network.
- (2) It is useful to have an organization dedicated to monitoring authentications and alerting the authorities. Attacks can happen at any time, and most companies will doubtfully hire 24/7 staff that is trained to quickly respond to a duress authentication alarm. A monitoring company can service many organizations simultaneously and share the operating costs.

6. KEYWORD DICTIONARIES

In our authentication schemes, the user’s keyword is randomly chosen from a dictionary. This section defines the properties that make a good keyword dictionary and provides an example one that satisfies those properties.

6.1 Requirements

The dictionary used in the schemes must be chosen very carefully. To avoid false alarms, we must make sure that the user does not accidentally mistype their keyword and accidentally cause a duress authentication. We must also make sure that the user knows the dictionary well and can easily lie about their keyword when they are under attack.

Definition 5 – Dictionary Typo Quality: For each pair of distinct keywords (w_1, w_2) , the probability of accidentally mistyping w_1 to form w_2 must be greater than a fixed quality θ . A high typo quality dictionary must have θ be sufficiently large so that the false alarms due to typos can be eliminated.

Definition 6 – Well Defined Dictionary: The user must be able to randomly pick a keyword and be certain that it is in the dictionary by only knowing the topic of the dictionary. In other words, the user should not have to memorize keywords that are in the dictionary for the purpose of the authentication scheme; she should only need to know the general theme. For example, the dictionary may be defined as the names of U.S. states – something which most U.S. residents already know.

It is not trivial to define a dictionary which meets the requirements defined above. For example, “the set of all English words” is a poor choice because it has a very low typo quality. The word “rent” can easily be mistyped to “rent”, especially since ‘r’ and ‘t’ are right next to each other on a QWERTY keyboard.

If the dictionary is not well defined, when a user needs to lie about her keyword as our schemes instruct her to do under duress, she may accidentally choose an obscure keyword which happens not to be in the dictionary. Since the dictionary is public knowledge, the attacker would be able to immediately identify that the user is lying and she loses the stealth property.

6.2 A Sample Dictionary: States of the U.S.A.

To make our scheme practical, we propose an example dictionary which has high typo quality and is well defined. Our dictionary consists of the states of the U.S.A. states typed as one word without spaces and they are not case sensitive.

For this example, we assume that the user knows at least a few states in the U.S.A. and we argue that the dictionary is well defined. Users who live outside of the U.S.A. or may be unfamiliar with the names of states, can be offered the choice of a different dictionary. Each user can choose their own dictionary and must lie about their keyword by picking from the dictionary they chose.

To demonstrate that this dictionary has a high typo quality, we present the following table. We generated this table by computing all pairwise edit distances and then for each keyword in the dictionary, we found the closest keyword and recorded its distance. The table is shown sorted by edit distance. This table essentially describes the nearest neighbor graph using the edit distance metric. The purpose was to find the most likely typo for each keyword.

As can be seen from the table, the most likely typos are to type “arkansas” instead of “kansas”, “northcarolina” instead of “southcarolina”, “northdakota” instead of “southdakota”, and any of those vice versa. Even the most common typos seem quite unlikely because the user would have to type “north” instead of “south” or mistakenly place “ar” before “kansas”.

#	Keyword	Closest Keyword	Edit Distance
1	arkansas	kansas	2
2	kansas	arkansas	2
3	northcarolina	southcarolina	2
4	northdakota	southdakota	2
5	southcarolina	northcarolina	2
6	southdakota	northdakota	2
7	alabama	alaska	3
		...	
45	rhodeisland	louisiana	6
46	washington	michigan	6
47	newhampshire	newmexico	7
48	connecticut	kentucky	8
49	pennsylvania	indiana	8
50	massachusetts	arkansas	9

7. SECURITY AND PRIVACY

The security of both schemes is at least as strong as existing modern authentication systems because they behave exactly the same until they determine that the password is correct. Only then do they proceed with duress detection.

However, unlike existing authentication schemes, our schemes are also required to preserve the stealth property (defined in section 3.5). The single administrator scheme does not satisfy the stealth

property when more than one administrator is being attacked. We demonstrate this weakness with the following attack.

Attack 3: Oscar attacks two administrators: Alice and Bob. He demands that they each tell him their correct password and keyword. Oscar then picks between Alice and Bob randomly and authenticates with the credentials of the administrator he picked. Without loss of generality, let's assume he picked Alice. After authenticating as Alice, he accesses the keyword database (which contains hashes of the keywords and checks to see if the keyword Bob provided is correct. If Bob lied about his keyword in an attempt to signal the alarm, Oscar would be able to find out and Bob loses the stealth property. Similarly, if Oscar had picked to authenticate as Bob, he would have verified Alice's keyword and been able to tell if Alice lied. Hence if either Alice or Bob attempted to signal the alarm, Oscar would have a $\frac{1}{2}$ chance of detecting it. If they both lie, Oscar would be able to detect it with certainty.

Because the single administrator scheme is susceptible to the stealth attack above, we presented the multiple administrator scheme. The multiple administrator scheme is designed such that the authentication server can never verify a keyword and instead delegates that task to the monitoring server. This way, it is not possible for Oscar (even with administrator privileges) to determine if Alice or Bob provided him with the wrong keyword and hence the stealth property is satisfied.

7.1 Authentication Privacy

Our scheme is designed such that the monitoring server can only decipher duress authentications. Normal authentications are encrypted and it does not possess nor can it compute the keys necessary to decrypt them. We call this property authentication privacy:

Definition 4 – Authentication Privacy: To the monitoring server, normal authentication records are indistinguishable from randomly generated authentication records. This implies that during normal authentication, the monitoring server cannot tell whether Alice is the user authenticating. It cannot even tell if two authentication records are associated to the same user.

Theorem 1 – Authentication Privacy: The authentication privacy property holds for the multiple administrators scheme.

Proof: By definition, for a normal authentication the keyword provided by the user during authentication is the same as the one stored by the authentication server. As can be seen in the pseudo code, `partialSecret1Id` and `partialSecret2Id` are computed from those two keywords using the formula:

$$1 + ((keywordId + account.SecretIdOffset) \bmod KEYWORD_COUNT)$$

Since `account.EncryptedIdAndPartialSecret` (from which we get `partialSecret1Id`) is always set immediately after `account.SecretIdOffset` changes, the value of `account.SecretIdOffset` used to compute `partialSecret2Id` is the same as that used to compute `partialSecret1Id`. Hence, since the keyword providing during authentication matches the stored one, then `partialSecret1Id = partialSecret2Id`.

These two partial secret ID's are used in Shamir's secret sharing algorithm such that recovering the secret requires two distinct shares, and hence they must have distinct partial secret ID's. Since the two ID's are identical, it is not possible to recover (or even infer partial information about) the secret.

The secret is the key for the encrypted user data in the authentication record, which contains all of the user-specific information. Since the monitoring server does not know this key, `encryptedUserData` is indistinguishable from random. Since the partial secret ID's and the Shamir polynomial are randomly generated for each authentication they are also indistinguishable from random. Hence, the entire normal authentication record is indistinguishable from random.

8. PERFORMANCE

We implemented the multiple administrators scheme and ran an experiment to measure its running time performance. The code was implemented in C# 3.0 and the experiment was run on a typical server processor, Xeon X3460 2.8GHz processor (about \$350), running on 64-bit Windows Server 2008 R2.

We timed 2,000 authentications for different RSA key sizes of the monitoring server (referred to as `MONITOR_PUBLIC_KEY` and `MONITOR_PRIVATE_KEY` in the pseudocode). The following table gives the average authentication time (the function `Authenticate` in the pseudocode) and the average monitoring time (the function `MonitorAuthentication`).

	Authentication Time	Monitoring Time
1024-bit Keys	0.203 ms	0.125 ms
2048-bit Keys	0.250 ms	0.671 ms
3072-bit Keys	0.343 ms	2.075 ms
4096-bit Keys	0.468 ms	6.318 ms

The authentication time represents the load on the authentication server for a single login. It is dominated by the asymmetric (RSA) encryption in the function `SetAccountKeyword` and the time to send the authentication record to the monitoring server.

The monitoring time represents the load on the monitoring server for a single login. It is dominated by the asymmetric (RSA) decryption in the function `MonitorAuthentication`.

It should be noted that although authentication does an RSA encryption and the monitoring does an RSA decryption, the authentication time for 1024-bit keys is still slightly higher because it includes communication latency. However, for larger key sizes, the RSA decryption clearly dominates the monitoring time and hence the monitoring server incurs a larger performance cost than the authentication server.

Our results demonstrate that for 1024-bit keys, the authentication server can handle about 5,000 logins per second and the monitoring server can handle about 8,000 logins per second. It is unlikely that an organization will have more than that many simultaneous authentications, but even if it does, the scheme is parallelizable and the problem can be solved by using additional servers.

Since asymmetric encryption/decryption account for a significant portion of the running time, the scheme can be optimized by using elliptic curve cryptography, which is known to be faster than RSA because it allows the use of smaller keys for the same level of security.

9. CONCLUSION

This paper presents authentication schemes that provide duress detection for attacks against one or more users/administrators. Our schemes are designed in such a way, that the attacker cannot

determine if the victim provided him with real or duress credentials which signaled a silent alarm. By knowing that there is no way to coerce the victim into avoiding the alarm, the attacker has little incentive to attempt an attack.

We proposed a keyword dictionary that can be used with our scheme to minimize the chance of setting off false alarms via typos. Also, the burden on the user is minimal and effectively consists of augmenting their password with a word such as "indiana".

Our scheme can be easily augmented to the code of an SSH server and does not require that the SSH client be modified. Our implementation shows that the multiple administrators scheme performs fast enough to be used in practice even by large organizations with many users.

10. REFERENCES

- [1] Clark, Jeremy and Hengartner, Urs. Panic passwords: authenticating under duress. In *Proceedings of the 3rd Conference on Hot Topics in Security* (2008), USENIX Association, 1--6.
- [2] Russikoff, Ronald K. Computerized password verification system and method for ATM transactions. United States Patent, 6871288, March 22, 2005.
- [3] Leemon C. Baird, et al. Apparatus and method for authenticating access to a network resource. United States Patent, 6732278, May 4, 2004.
- [4] Michael Wayne Brown, Rabindranath Dutta, Michael A. Paolini, Newton James Smith, Jr. Cash register and method of accounting for cash transactions. United States Patent, 6550671, April 22, 2003.
- [5] RESEARCH IN MOTION. Duress Notification Address IT policy rule. Retrieved July 2, 2009 from BlackBerry Enterprise Solution Security - Policy Reference Guide: http://na.blackberry.com/eng/deliverables/4222/Duress_Notification_Address_204132_11.jsp.
- [6] SPRINT. WebID Authentication with a SecurID PINPAD. Retrieved July 2, 2009 from: http://cagate.sprint.com/documentation/securid/documents/SecurID_Pinpad_Token.pdf.
- [7] ALMEX LTD. Bioscrypt Fingerprint readers for door access sold by Almex. Retrieved July 2, 2009 from: <http://www.almexltd.com/fingerprint-readers.htm>.
- [8] Howie, John. Authentication Options. *Windows IT Pro* (July 2006).
- [9] Weinshall, Daphna and Kirkpatrick, Scott. Passwords you'll never forget, but can't recall. In *Conference on Human Factors in Computing Systems* (2004), ACM, 1399--1402.

A.1 APPENDIX: MULTIPLE ADMINISTRATORS SCHEME PSEUDOCODE

```
1 Authenticate(username, password, keyword)
2 {
3   account = GetAccount(username);
4   if (account == NULL or password is not correct for this user)
5     return FALSE;
6   keywordId = GetKeywordId(keyword);
7   partialSecretId = 1 + ((keywordId + account.SecretIdOffset) mod KEYWORD_COUNT);
8   partialSecret = GenerateShare(partialSecretId, account.Polynomial);
9   userData = { username, account.FullName, account.HomeAddress };
10  userData = Pad(userData, MAX_USER_DATA_LENGTH);
11  encryptedUserData = SymmetricallyEncrypt(userData, account.Secret);
12  record = { account.EncryptedIdAndPartialSecret, partialSecretId, partialSecret, encryptedUserData };
13  SendToMonitoringCompany(record);
14  SetAccountKeyword(account, keyword);
15  return TRUE;
16 }

1 CreateAccount(username, password, keyword)
2 {
3   if(there is already an account with username as the user)
4     return FALSE;
5   if(keyword is not the dictionary)
6     return FALSE;
7   account = CreateRegularAccount(username, password);
8   SetAccountKeyword(account, keyword);
9 }

1 SetAccountKeyword(account, keyword)
2 {
3   account.Secret = GenerateRandomSymmetricKey();
4   account.Polynomial = CreateShamirPolynomial(account.Secret);
6   account.SecretIdOffset = Random integer in the range [0, KEYWORD_COUNT - 1]
8   keywordId = GetKeywordId(keyword)
9   partialSecretId = 1 + ((keywordId + account.SecretIdOffset) mod KEYWORD_COUNT);
11  partialSecret = GenerateShare(partialSecretId, account.Polynomial);
13  idAndPartialSecret = {partialSecretId, partialSecret};
14  account.EncryptedIdAndPartialSecret = AsymmetricallyEncrypt(idAndPartialSecret, MONITOR_PUBLIC_KEY);
16 }

1 MonitorAuthentication(record)
2 {
3   { encryptedIdAndPartialSecret1, partialSecret2Id, partialSecret2, encryptedUserData } = record;
6   partialSecretAndId1 = AsymmetricallyDecrypt(encryptedIdAndPartialSecret1, MONITOR_PRIVATE_KEY);
8   {partialSecret1, partialSecret1Id} = partialSecretAndId1;
10  if (partialSecret1Id ≠ partialSecret2Id) {
12    secret = ReconstructSecret(partialSecret1Id, partialSecret1, partialSecret2Id, partialSecret2);
14    userData = SymmetricallyDecrypt(encryptedUserData, secret);
15    userData = RemovePadding(userData);
16    { username, fullName, homeAddress } = userData;
18    Notify the police that a person is under duress and that
19      his/her name is fullName,
20      his/her address is homeAddress,
21      his/her username is username;
23  }
23 }
```