Abstract

There are many methods of authentication in use today; however they all have rather similar problems in common. Ruling is a rule based authentication method that works between a user and a system, and seeks to rectify some of these problems. First this paper will analyse the current level of research in the field of authentication methods. The common problems will be discussed, followed by the proposed solution called Ruling.

Ruling’s main aim is to keep the secrets of an individual’s authentication method a secret. The rules, which this method is based on and the user creates, are only transmitted once during registration. Challenges are randomly generated for the user to apply their set of rules against in order for them to create an appropriate Response. The implementation of Ruling used in this paper successfully demonstrated the proof of concept. Ruling’s trade-offs were that it was that it appeared more difficult for an observer to replicate the necessary information in order to reliably authenticate, but it also seemed more difficult for a user to remember their Ruleset. Different styles of setting rules would also affect how secure, precise, and rememberable the authentication steps were.
Contents
Abstract.................................................................................................................................1
1 Introduction..........................................................................................................................3
  1.1 Rationale.....................................................................................................................3
2 Literature Review...............................................................................................................4
  2.1 Literature Review Strategy..........................................................................................4
  2.2 Defining Authentication..............................................................................................4
     2.2.1 The “Knows”, “Has”, and “Is” Abstractions.......................................................4
  2.3 What a user “Knows”...................................................................................................4
     2.3.1 The Password and its Flaws................................................................................4
     2.3.2 GridWord..............................................................................................................5
     2.3.3 Win8 “Picture Passwords” ................................................................................5
  2.4 What a user “Has”.........................................................................................................5
  2.5 What a user “Is”............................................................................................................5
     2.5.1 Physiological Biometrics.....................................................................................5
     2.5.2 Behavioural Biometrics.......................................................................................6
     2.5.3 Sound-Based Authentication.................................................................................6
  2.6 Private Glass................................................................................................................6
  2.7 Literature Summary.......................................................................................................6
  2.8 The Common Problem..................................................................................................6
3 Methodology.....................................................................................................................7
  3.1 Finding a Solution..........................................................................................................7
  3.2 “Ruling” – The proposed Solution................................................................................7
  3.3 Solution Implementation..............................................................................................8
     3.3.1 Requirements.........................................................................................................8
     3.3.2 Technologies..........................................................................................................10
     3.3.3 Technical Specification.........................................................................................10
4 Results................................................................................................................................10
  4.1 Testing............................................................................................................................10
  4.2 Analysis..........................................................................................................................26
5 Discussion........................................................................................................................27
  5.1 Potential Further Work.................................................................................................27
6 Conclusion........................................................................................................................28
Bibliography..........................................................................................................................29
Appendix...............................................................................................................................31
  1. Class CLetter Extract....................................................................................................31
1 Introduction

Authentication is part of everyday life. A user is constantly logging into user accounts, using PIN services, and unlocking devices. In some cases, stealing someone’s identity can be as easy as looking over their shoulder as they are signing in. This paper discusses the current methods of user authentication, and what their common flaws are. Ruling, a rule based authentication method, is proposed a possible solution to these problems, and Ruling’s pros and cons are also discussed.

1.1 Rationale

Keeping data secure is consideration for every website that employs user accounts. This problem stems from the idea that different people should only be allowed to know certain pieces of information. You don’t want your bank details public, and neither does your bank. Exposing sensitive information to only those people who it should be has been a problem for centuries. (Bonneau, 2012)

The traditional method of keeping your data secure is to require a username and password, however this technique is quite easily observable and user behaviour creates a number of different security flaws. (Constantin, 2009) A new method of authentication method is required if safeguarding against observation attacks is paramount. Ruling could be a solution.
2 Literature Review

Methods of authentication have long been an issue within computing, they have two basic requirements, to be efficient as well as secure. (Chen, et al., 2013) (O'Gorman, 2004) This section will go into detail about how the computing industry is trying to deal with this problem for the average user, as well as point out flaws in these works to be discussed later.

2.1 Literature Review Strategy

The objective for this literature review is to compile an overview on how well current technologies deal with authentication security, the assumption being that networks and protocols are secure, and are not under attack from a truly determined attacker. Literature that discusses methods of improving end-user security when authenticating, as well as common methods of authentication coupled with their advantages, and drawbacks, will be dissected. Interesting, important, common, or otherwise relevant ideas that offer a well thought out solution to keeping authentication secure should get a mention.

2.2 Defining Authentication

For the purposes of this paper, authentication is the practice of verifying a user’s, system’s, or person’s, identity. Authentication often relies on the user, system, or person, to know, have, or be, something unique.

2.2.1 The “Knows”, “Has”, and “Is” Abstractions

It will be useful to split up the coming sections based on this definition as what category of authenticator they are. The coming sections will discuss what a user “Knows”, “Has”, or “Is”.

What a user “Knows” will refer to some kind of secret information. Only the user should know this information. A common example is a password.

What a user “Has” will refer to a physical key or object which holds identifiers within itself. Only the user should have access to the unique physical object. A common example is the use of RFID.

What a user “Is” will refer to the characteristics of the user themselves. Each user should be unique enough for their being to be used as a method of authentication. A common example is a fingerprint scanner.

2.3 What a user “Knows”

“Know” authenticators are things that a person, user, or system, should be able to easily recall, and it should not change often, if ever. It could be a user’s birthday, it could be a particular response to a particular challenge, it could be the capitol of Angola. “Know” authenticators rely on the information security of the unique knowledge of the person, user, or system, who authenticates with it.

2.3.1 The Password and its Flaws

Passwords, as a method of secure identification, have been around since at least the 1960’s in computing terms (McMillan, 2012), with the first recorded use of a password being in the roman era (Bonneau, 2012). Fifty years of technological development, and roughly a millennia for social development, should have seen the security and best practices of The Password become incredibly rugged and well-enforced. But databases are lost, websites are cracked, details are phished, and user credentials are inevitably stolen. There are several fundamental flaws with how humans, in general, create and manage their passwords.
A Rule Based Authentication Method By Simon Ranson

Passwords are supposed to be unique, secure, and secret identifiers for an individual to prove who they are to a system; however this is routinely not the case. If we look at security fails in the past, we can see that many users have simply been too lazy to create a password that would stand up to a brute force, that is not obvious, or that are simply not unique. (Constantin, 2009) (Emam, et al., 2011) (Florencio & Herley, 2007)

One major and common problem is how routinely easy it is to set an easily guessed password (such as the famed: “1,2,3,4…”). These are easily cracked with off-the-shelf password recovery tools. (Emam, et al., 2011) (Florencio & Herley, 2007)

![Diagram of password strength](image)

(XKCD, 2014)

Sometimes people share passwords. This might be incredibly convenient for spouses managing their money, or for people with limited bank access, but it drastically increases the area of vulnerability for the associated account. (Singh, et al., 2007) (Florencio & Herley, 2007)

Clearly, the traditional thinking behind passwords needs to be changed. A new system needs to be developed that will make a password of “1,2,3,4,5” or “FirstnameLastname1” to simply be impractical, or maybe even impossible, for a user to set.

**Password Compromising Techniques**

The simplest way to obtain a user’s password without their consent is to observe them as they input their password. This kind of shoulder surfing is easily repeatable at your local pub over a pint.
Another method of capturing passwords would be to attach a keylogger to a public machine. The same principle as above applies, you see exactly what the victim is typing, and all you have to do is replicate the inputs.

Failing this, you can always try to guess the password of the identity you are attempting to steal. In some instances, such as using partial passwords, guessing the three characters asked might be easier than capturing the entire password to always ensure a correct response. (Aspinall & Just, 201?)

Poor storage is also to blame for password loss. When a company loses their users database, the inherent problems with passwords come to light. (Goodin, 2012) The common tendencies are quite shocking, and if the average user was responsible when creating their passwords, this would not be a problem. However, is it unrealistic to expect users to remember entirely different passwords for every single identity they have?

A technique of hash-busting a set of passwords is to hash every possible combination of characters beforehand. All you have to do then is search the standard table for the hash you have in the table you stole. This sacrifices disk-space for speed, but is incredibly good at cracking obvious passwords. Rainbow tables take this a step further. (Avoine, et al., 2008) The disk space required to store every 10-character password with its corresponding MD5 hash is around 3000 terabytes, whereas a rainbow table can store 99.9% of the first table, while only using 167 gigabytes. (Goodin, 2012)

This exposes a critical flaw with passwords, they are static pieces of information. Their unchanging nature might make them easier to remember, but it makes them incredibly vulnerable to replication. The pre-guessing also betrays a different flaw with one-way
encryption. A hash, like the password used to create it, is just another secret piece of static information.

**Partial Passwords**

HSBC has come up with a very novel idea of having a user set a whole password, but only have to reveal a few characters of it. This was adopted to increase the security of telephone banking, so the user would not have to reveal their entire password to another human being out loud and over the phone. Partial Passwords are quite resistant to shoulder surfing and recording attacks, but is more vulnerable to guessing. It only requires three characters to be correct at any one time to be authenticated. (Aspinall & Just, 201?)

**Security Questions**

Security questions are essentially the exact same as passwords, but are slightly more vulnerable. They suffer from all the pit-falls of regular passwords, but with the added down-side of being easily guessed by an identity thief. A common question is for a user’s mother’s maiden name; this information is publicly available.

### 2.3.2 GridWord

An attempt to make authentication easier for both mobile devices and desktops, GridWord caters for devices with and without hardware keyboards. GridWord is a variation on basic passwords. It offers users a pre-defined list of words, of which a combination may be used to create a passphrase. Users can either manipulate drop-down menus, textual inputs with auto-complete, or the grid of words, to input their passphrase. The design is such that it aids memory by displaying the grid in a static state; it is identical every time. (Bicakci & van Oorschot, 2011)

A notable part of this is how much consideration there is for both desktop and mobile users. When developing an alternative to passwords, accessibility for mobile devices must be considered. However, having a completely static grid means that inputs can be very easily observed and repeated. Large, static features that display the status of inputs are undesirable when password secrecy is top priority.

### 2.3.3 Win8 “Picture Passwords”

The picture password of Windows 8 was designed to reduce the time it took for users to authenticate into mobile devices while making the login screen more personal. As this was intended for individual users, it became clunky when switching between users compared to the traditional password. (Sinofsky, 2011)

During testing, the developers found that using a constrained set of gestures helped reduce the time it took for users to authenticate. It has also been suggested to have little impact on security when compared to “freeform” gestures. (Sinofsky, 2011) However, the overall security of the implementation is questionable with a Russian password recovery company selling a product that can exploit alternative logins. (Goodin, 2012) This also does not address the issue of people looking at finger smudges on the screen to see the three gestures.

### 2.4 What a user “Has”

“Have” refers to items and objects a person, system, or user, will possess. It could be a key, a key card, an RFID chip, or even a USB dongle. “Have” authenticators rely on only the valid user “having” the unique physical item, and thus inherit the same level of security as the physical object does. In other words, if an ID card gets stolen, the attacker can assume an identity with extreme ease. “Has” authenticators do not tend to have problems in which better programming techniques can solve, thus they are omitted for being out of scope for this paper.
### 2.5 What a user “Is”

“Is” refers to what a person’s, system’s, or user’s physical characteristics are. It can be a user’s fingerprints, an iris scan, or DNA. “Is” authenticators rely on the statistical improbability of people, users, or systems, having identical physical characteristics, but are often incredibly difficult to change once they become compromised.

#### 2.5.1 Physiological Biometrics

There are many ways biometrics can be used to authenticate a user; some common methods use fingerprinting, retinal scans, hand-written signatures, and facial recognition. In essence, these methods are both easy and difficult to compromise. A hacker would require the correct tools for the job, this could be as obscure as a fake finger made out of gelatine, or as simple as a photocopier, but once a biometric system has been broken, it is very difficult for the legitimate user to regain the security of their account. The practice of creating a fake finger can be quite challenging, you would need to find a suitable fingerprint to copy, as well as form the material to use to a high enough standard that the reader, and anyone else around you, will be fooled as to its use. (Schneier, 2013) (Schneier, 1998)

One drawback of using biometrics is the difficulty of ensuring the sample was taken at the correct time, and by the same user seeking authentication. Taking a picture of someone discreetly and using that for photo ID is not too challenging. Intercepting the transmission between a fingerprint reader and its server, and replicating the data being sent, might also be easier than making a physical finger. (Schneier, 2013) (Schneier, 1998)

These types of authentication methods might well be quite appealing to end users, since the simplicity and perceived difficulty of reconstruction is reassuring. (Schneier, 2013) But the fact remains that biometrics relies on data that is not secret. Your fingerprint is left of hundreds of objects every day, and placing trust into a method of authentication that does not have secrecy may as well be serving your sensitive information up to competent attackers. (Schneier, 1998)

#### 2.5.2 Behavioural Biometrics

Behavioural Biometrics refers to measuring how a user acts. This can be a voice sample, a hand-written signature, or keystroke dynamics. The main problems are either inherent to Biometrics as a whole, or are that a user’s behaviour can change based on their state. A user who is drunk might not be able to replicate their speech sample well enough in order to pass authentication. (Bergadano, et al., 2002)

#### 2.5.2.1 Keystroke Dynamics

Keystroke Dynamics involves analysis on how a user interacts with a keyboard, and therefore fall under behavioural biometrics. This can run into extra problems as implementations for remote authentication are required to take varying latency into account. Giving multiple samples tends to solve this issue. (Bergadano, et al., 2002)

A huge advantage to using keystroke dynamics, when compared to other forms of biometrics authentication, is that a keyboard is relatively cheap compared to an iris scanner. (Bergadano, et al., 2002) Not all devices have keyboards, and a user who tries to authenticate on their desktop, using a standard keyboard, and then on their phone, using a comparatively tiny on-screen keyboard while only using one hand to type, will have trouble trying to access the same account.

#### 2.5.3 Sound-Based Authentication

Sound-Based authentication methods are quite rare. Loud-and-Clear (L&C) is system that enables two, previously unknown to each other, devices to authenticate each other’s identities. Actual authentication at the first stage is verified by the human user. What’s
interesting about this authentication method is that both wireless and sound are used to pair the two devices. L&C uses a highly glorified public/private key authentication method. This was designed to ensure users were pairing their devices correctly, and not to an unknown third party device by accident. (Goodrich, et al., 2006)

One implementation of this is employed by a company called SlickLogin, using your phone as an authenticator when logging in with your desktop. (Hidenbrad, 2014) (JC, 2014) SlickLogin works as a trusted third party that authenticates you into a supporting website. However, it seems to have similar weak spots as to the Biometrics authentication methods that have been discussed previously. It does not seem to solve the problem and does not seem to make authentication any easier. It is a two-step process and relies on your phone being connected to the internet. (JC, 2014)

2.6 Private Glass

The subject so far has been methods of authentication, accompanied with a very brief discussion on where they are weak or strong. One of the common weaknesses is simply having a login screen on a mobile device being observed by others as a legitimate user authenticates.

One way to reduce this area of vulnerability is to reduce the visible angle of the input. Private Glass, as patented by apple, has the capability to reduce the viewable angles of a display at the user’s command. (Purcher, 2011) This might be incredibly useful for viewing sensitive documents, or authenticating on a touchscreen device. Its usefulness largely depends on how well the technology is implemented, and how accurate, and small, the visible angle is.

This could make shoulder surfing for passcodes more difficult, and require a degree of sophistication higher than simply leaning over. This could also be circumvented if things like finger-smudges are left on static key positions, of which their application could still be easily observed no matter how smart the glass is.

2.7 Literature Summary

User authentication is important for having a secure system, but major problems still exist today. Good passwords are not often created by users. (Constantin, 2009) (Emam, et al., 2011) (Florencio & Herley, 2007) (Forget, et al., 2007) Passwords are sometimes shared between users. (Singh, et al., 2007) (Florencio & Herley, 2007) The Passwords that users are encouraged to set are often easy to guess and difficult to remember. (XKCD, 2014) (Florencio & Herley, 2007) It is possible to set insecure, obvious, or “common” passwords. (Constantin, 2009) (Emam, et al., 2011) (Florence & Herley, 2007) (Forget, et al., 2007) (McMillan, 2012) Passwords can be easily observed, and replicated.

There are a few alternatives to passwords; each has their flaws, which are quite often similar. GridWord considers mobile devices well, but is more easily observed. (Bicakci & van Oorschot, 2011) Picture Passwords from Windows 8 also work on mobile devices, but are also simple to capture and replicate, and are exploitable. (Sinofsky, 2011) (Goodin, 2012) Biometrics might seem appealing and secure, but once they have been broken, the difficulty of regaining security of the compromised account is high. Also, it can be difficult to determine if samples were taken at the time of authentication. (Schneier, 2013) (Schneier, 1998) Behavioural analysis has similar pitfalls to other biometrics authentication methods, but a legitimate user can’t exactly replicate their own behaviour. (Bergadano, et al., 2002)

2.8 The Common Problem

The common link with all these authentication methods is that they rely on static authenticators that end up being quite observable during input. Static authenticators might be
easy to remember, but they are also easy to capture and imitate. A truly secure method of authentication must rely on truly secret information, and none of the previously discussed methods do.
3 Methodology

This paper has explored current works for authentication methods and has come across a common problem. Here, a solution will be proposed, explored, and designed, for the issues outlined before.

3.1 Finding a Solution

Static and observable authentication methods create vulnerabilities when observation attacks are a possibility. A method to slightly alleviate this issue could be to enforce changing the static information every so often, making it more dynamic. The problem with this is obviously, a compromised password is compromised until it is changed, and constantly changing your password is impractical.

Instead of changing a password every few months, for example, what if the correct response was different for every login? The user would no longer be remembering a single password that is observable; the user would remember the method to build their password and not each password itself. This means the method of authenticating would never have to be reviled and only ever transmitted once.

One way to provide a base for each password to be built by a user could be to issue a challenge, this way there would be no reliance on date, time, location, etc. The challenge could be randomised and therefore very rarely repeated; reducing the number of times a challenge-response pair could be seen. This would mean that, despite individual pairs could be observed, the next challenge could be completely different and thus impossible to authenticate to without knowing the secret method.

This is a good base. Taking into consideration mobile device users, the character set required would have to be small. Numerical, base ten, digits should suffice in keeping any methods simple enough to input that it’s easily possible for a user with an onscreen keyboard to accomplish. Simple mathematical functions could be used for each rule of the authentication method, and the results of those functions could be verified in the response. This will be the base idea of the proposed solution.

3.2 “Ruling” – The proposed Solution

The proposed solution is called: Ruling. It is a rule based authentication method. Rules are only ever transmitted once during account creation, and assuming the single transmission and the storage are both secure, third party discovery of a user’s ruleset will be difficult.

An important consideration to make is for mobile devices, a simple solution is to reduce the character set used. Thus, the authentication terminology will look like follows:

- A “Letter” is a single, random, number whose value is between 0 and 9.
- A “Word” is a set of four Letters.
- A “Phrase” is a set of exactly words, each Word being spaced accordingly.
- A “Challenge Phrase” is generated but is not entirely random, no Letter can have the value of 0. This is the challenge a system poses to a potential user.
- A “Response Phrase” can contain any Letter, and is the response a user give to be authenticated.
- An “Anchor” is a Letter that has been used within the user’s rule set. It is denoted as “(<Word number>,<Letter number>)”. Example: (3,1) references the first Letter of the third Word.
A Rule Based Authentication Method By Simon Ranson

- A “Rule” is a simple function involving the value of the Anchor Letters, paired with simple mathematical functions (+, -, *).

Rules are formatted as follows:

**Rule Format:** (cw0, cl0) + (cw1, cl1) - (cw2, cl2) * (cw3, cl3) ... => (rw, rl)

“c” Represents Anchors in the Challenge Phrase, “r” represents Anchors in the Response Phrase. “w” represents the position of the Word the Anchor is in, “l” represents the Letter position within that word. The first Word in a Phrase is Word “1”, the first Letter in a Word is Letter “1”, this intentionally leaves out “0” for two reasons: Rules are going to be used primarily by humans, and humans tend to work with “1” representing the very first item in a list; “0” could then be used for something else, for example representing all the Letters in a Word at the same time, for future expansion.

In essence, a legitimate user remembers a small set of Rules. He or she will be given a randomized Challenge Phrase, in which the user then applies their Rules. The user’s Rules dictate what a legitimate Response Phrase could look like. It’s important to realise that there are more than one possible correct response as only a relatively small portion of the letters in a phrase are actually inspected for each individual’s authentication. Specifically, only the Anchors in the Response Phrase are inspected, all other Letters must exist but their values do not matter.

Each Rule defines its set of Anchor Letters and mathematical functions. Rules are executed left to right with the final Anchor being a Letter that appears in the response phrase to be authenticated. As a Rule’s response Anchor is only ever a single Letter, it is calculated from the least significant digit of the absolute value of a Rule’s equation. Example of how this works is a follows:

![Diagram](image)

-Figure 1

“Rule0” in this case comprises of adding the first Letter of the first Word, the third Letter of the second Word, and subtracting the fourth Letter of the fourth Word, with the result being placed in the third Letter of the fourth Word in the Response Phrase. 1+3-8 = -4, but we take the absolute value of this result, thus the Letter “4” is placed at “(4,3)”. 

Page 13 of 42
“Rule 1” in this case involves the addition of Anchors (2,2) and (4,4), which have the values of “8” and “8” respectively. The product of “8” and “8” is “64”, we place the least significant digit, the digit furthest to the right, which has the value “4”, in the Anchor (1,2) in the Response Phrase.

Thus, any response phrase with the Anchors of (4,3) and (1,2) having the values of “4” and “4” respectively, will positively authenticate. The rest of the Letters in the Response Phrase are randomly chosen by the User, they serve no purpose apart from obscuring the possibilities of what the user’s rules could be, and what their Anchors are.

The security of this solution will come from how difficult it is to reconstruct the Rules based on the Challenge Phrase and the Response Phrase, and how difficult it is to find the Anchors.

Another example:

![Diagram](image)

**Figure 2**

This example demonstrates how execution of Rules is from left to right, and how “0” is could occur in Response Phrases.

### 3.3 Solution Implementation

The solution has been defined, but a specification that simply records the requirements will assist in tracking progress of completion. A testing specification will also be useful to prove when user requirements have been met. Use Cases will also provide the basic outline of what is expected from the solution. The solution will be as simple as possible.

Test Driven Development will be used to guide the implementation. TDD provides a rugged platform for ensuring proper implementation. Waterfall just takes too long, and is nowhere near as precise. (Canfora, et al., 2006)

#### 3.3.1 Requirements

This section details the exact requirements to fulfil the solution outlined in the methodology. “Priority” is given in the MoSCoW format. M being “Must”, S being “Should”, C being “Could”, and W being “Would”:

**User Specification**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A user can input a Rule</td>
<td>M</td>
</tr>
<tr>
<td>2.</td>
<td>A user is issued a Challenge Phrase</td>
<td>M</td>
</tr>
<tr>
<td>3.</td>
<td>A user can input a Response Phrase</td>
<td>M</td>
</tr>
<tr>
<td>4.</td>
<td>A user is shown if they pass verification</td>
<td>S</td>
</tr>
</tbody>
</table>
A Rule Based Authentication Method By Simon Ranson

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.</td>
<td>Testing of the provided solution is simple</td>
</tr>
<tr>
<td>6.</td>
<td>The Rule can include subtraction</td>
</tr>
<tr>
<td>7.</td>
<td>The Rule can include addition</td>
</tr>
<tr>
<td>8.</td>
<td>The Rule can include multiplication</td>
</tr>
<tr>
<td>9.</td>
<td>No 0’s can be used in the Challenge Phrase</td>
</tr>
<tr>
<td>10.</td>
<td>0’s can be used in the Response Phrase</td>
</tr>
<tr>
<td>11.</td>
<td>A Response Phrase is full of Letters</td>
</tr>
<tr>
<td>12.</td>
<td>More than two Challenge Anchors can be used in one Rule</td>
</tr>
<tr>
<td>13.</td>
<td>Negative values do not exist in the Challenge Phrase</td>
</tr>
<tr>
<td>14.</td>
<td>Negative values do not exist in the Response Phrase</td>
</tr>
<tr>
<td>15.</td>
<td>Multiple Rules can be used</td>
</tr>
<tr>
<td>16.</td>
<td>A Rule must be entered</td>
</tr>
</tbody>
</table>

| Use Cases |

This section will detail the use cases for the proof of concept. The numbered items are the expected route, and any branches (“a”, “b” etc ...) are exceptional cases. Each case details an expected function of the coding project.

3.3.1.1.1 Testing the Authentication Mechanism
1. The user inputs a test rule.
2. The user is issued a random challenge phrase.
3. The user inputs a response phrase.
4. The user clicks Submit
5. A confirmation of authentication is displayed
   a. The confirmation is not displayed on unsuccessful authentication

Test Specification
This is the testing specification. It will be used to ensure the user requirements are met. Each is numbered to provide easy reference later on. The “U.S.No.” refers to the User Specification Number that the test is inherent to, sometimes one test is important for multiple user requirements.

<table>
<thead>
<tr>
<th>No.</th>
<th>U.S. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>A text box for Rule input exists</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>A text box for Challenge Phrases exist</td>
</tr>
<tr>
<td>3.</td>
<td>2</td>
<td>The text box for a Challenge Phrase is filled automatically</td>
</tr>
<tr>
<td>4.</td>
<td>5</td>
<td>The text box for Challenge Phrases can be manually edited</td>
</tr>
<tr>
<td>5.</td>
<td>3</td>
<td>A text box for a Response Phrase exists</td>
</tr>
<tr>
<td>6.</td>
<td>16</td>
<td>The text box for a Rule must be filled by the user</td>
</tr>
<tr>
<td>7.</td>
<td>4</td>
<td>A method of displaying a successful authentication is displayed</td>
</tr>
<tr>
<td>8.</td>
<td>5</td>
<td>All user testing can be done on one page</td>
</tr>
<tr>
<td>9.</td>
<td>6</td>
<td>Subtracting two Anchors correctly, authenticates the user</td>
</tr>
<tr>
<td>10.</td>
<td>7</td>
<td>Adding two Anchors correctly, authenticates the user</td>
</tr>
<tr>
<td>11.</td>
<td>8</td>
<td>Multiplying two Anchors correctly, authenticates the user</td>
</tr>
<tr>
<td>12.</td>
<td>9</td>
<td>It is impossible to authenticate with any 0’s in the Challenge Phrase</td>
</tr>
<tr>
<td>13.</td>
<td>9</td>
<td>No random generations of Challenge Phrase create 0’s</td>
</tr>
<tr>
<td>14.</td>
<td>10</td>
<td>0’s Can be used as normal in the Response Phrase</td>
</tr>
<tr>
<td>15.</td>
<td>11</td>
<td>A Response Phrase is filled by the user before Authenticating</td>
</tr>
<tr>
<td>16.</td>
<td>12</td>
<td>Three Challenge Anchors are used with the same Operators</td>
</tr>
<tr>
<td>17.</td>
<td>12</td>
<td>Three Challenge Anchors are used with two different Operators</td>
</tr>
</tbody>
</table>
A Rule Based Authentication Method By Simon Ranson

<table>
<thead>
<tr>
<th>No</th>
<th>Test Spec No.</th>
<th>Description</th>
<th>Data Used</th>
<th>P/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>Rule Text Box appears</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Challenge Text Box appears</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3</td>
<td>Challenge Text Box automatically filled</td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

3.3.2 Technologies

The Rules require parsing and computation. It is possible for parser builders, such as ANTLR, to be used to solve this problem. However, due to lack of knowledge, using ANTLR is undesirable. Instead, a higher level language, such as Java or C# will be used to parse, and compute, the Rules.

There are a number of ways to accomplish this:

1. A “heavy” C# application could do all the work.
2. A tiny asp.net website could handle user inputs with c# conducting the logic.
3. Java is used instead, which can be injected directly into certain database engines.

Option two is most desirable. A tiny webpage will mimic a full implementation better than a heavy application, and could be tested on mobile devices more easily. The use of Java could help maintain the secrecy of rules by never having them be served outside a database, but that is more work than is required to prove the concept.

3.3.3 Technical Specification

The implementation will consist of a tiny asp.net website project. It will consist of a single page, with a textbox for a Rule, a textbox for a Challenge Phrase, a textbox for a Response Phrase, a text box to prove the underlying code can interpret the rule correctly and spit it back out, and some sort of indicator as to whether the previous authentication attempt was successful.

This is a minimalistic implementation, looking solely at the features of the authentication algorithm itself, and to ensure it works correctly. This makes things much easier to test, and helps ensure the method of authentication is what is being tested.

4 Results

The results section contains the results of testing, as well as an analysis of the results. Each test in numbered, for easy reference, and any data that is used is also shown. P/F refers to weather the test is considered to be a “pass” or a “fail” based on the outcome of the test.

4.1 Testing

With respect to the use of TDD, the preliminary tests that were written beforehand, all failed as expected, and have been omitted for simplicity. These are the same tests run after the final sprint.

<table>
<thead>
<tr>
<th>No</th>
<th>Test Spec No.</th>
<th>Description</th>
<th>Data Used</th>
<th>P/F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1</td>
<td>Rule Text Box appears</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Challenge Text Box appears</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>3</td>
<td>Challenge Text Box automatically filled</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>Test Spec No.</td>
<td>Description</td>
<td>Data Used</td>
<td>P/F</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Challenge Text Box randomly filled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Manually editing the Challenge possible</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Manual change of Challenge computed Challenge before: 8628 4842 1185 5663 Challenge after: 1111 1111 1111 1111 Rule: (1,2) + (1,3) =&gt; (1,1)</td>
<td>Ru: (1,2) + (1,3) =&gt; (1,1) Ch: 8628 4842 1185 5663 Re: 8923 9210 3849 2617 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>^</td>
<td>Ru: (1,2) + (1,3) =&gt; (1,1) Ch: 1111 1111 1111 1111 Re: 8923 9210 3849 2617 Authed? No.</td>
<td>P</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>^</td>
<td>Ru: (1,2) + (1,3) =&gt; (1,1) Ch: 1111 1111 1111 1111 Re: 2384 9218 1039 4853 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>Replicate previously successful authentication possible</td>
<td>Ru: (1,1) + (1,2) =&gt; (4,1) Ch: 3658 7173 3344 3868 Re: 2318 2138 7821 9213 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>Replicate previously unsuccessful authentication possible</td>
<td>Ru: (1,1) + (1,2) =&gt; (4,1) Ch: 7256 2154 2316 3386 Re: 2178 7216 7218 3982 Authed? No.</td>
<td>P</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>Response Text Box appears</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>Rule Text Box is a required field?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>Is there a checkbox which displays the outcome of the last attempt?</td>
<td>Ru: (1,1) + (1,2) =&gt; (4,1) Ch : 7888 3524 7558 7748 Re: 6273 2173 3674 5823 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>Checkbox indicates successful attempt</td>
<td>Ru: (1,1) + (1,2) =&gt; (4,1) Ch : 7888 3524 7558 7748 Re: 6273 2173 3674 5823 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>Test apparatus is on one form</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>9</td>
<td>Subtracting Anchors Ch: 8628 4842 1185 5663</td>
<td>Ru: (1,2) - (1,3) =&gt; (1,1) Ch: 8628 4842 1185 5663 Re: 4012 3948 6237 8492 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
<td>^</td>
<td>Ru: (1,3) - (1,1) =&gt; (1,1) Ch: 8628 4842 1185 5663 Re: 6482 9237 7238 7592 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>No</td>
<td>Test Spec No.</td>
<td>Description</td>
<td>Data Used</td>
<td>P/ F</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>18.</td>
<td>9</td>
<td></td>
<td>Ru: (1,3) - (2,4) =&gt; (1,1) Ch: 8628 4842 1185 5663 Re: 0123 2173 8456 1678 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>19.</td>
<td>9</td>
<td></td>
<td>Ru: (1,3) - (2,4) =&gt; (1,1) Ch: 8628 4842 1185 5663 Re: 1123 2173 8456 1678 Authed? No.</td>
<td>P</td>
</tr>
<tr>
<td>20.</td>
<td>10</td>
<td>Adding Anchors Ch: 3543 2846 3552 2134</td>
<td>Ru: (1,1) + (1,2) =&gt; (1,1) Ch: 3543 2846 3552 2134 Re: 8913 1238 4738 1923 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>21.</td>
<td>10</td>
<td></td>
<td>Ru: (2,2) + (1,2) =&gt; (1,1) Ch: 3543 2846 3552 2134 Re: 3294 1839 4759 6942 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>22.</td>
<td>10</td>
<td></td>
<td>Ru: (2,2) + (1,2) =&gt; (1,1) Ch: 3543 2846 3552 2134 Re: 9294 1839 4759 6942 Authed? No.</td>
<td>P</td>
</tr>
<tr>
<td>23.</td>
<td>11</td>
<td>Multiplying Anchors Ch: 3141 8121 1631 6568</td>
<td>Ru: (1,3) * (2,3) =&gt; (1,2) Ch: 3141 8121 1631 6568 Re: 9837 0564 2837 5649 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>24.</td>
<td>11</td>
<td></td>
<td>Ru: (2,1) * (4,2) =&gt; (3,3) Ch: 3141 8121 1631 6568 Re: 2374 8193 4901 3849 Authed? Yes.</td>
<td>P</td>
</tr>
<tr>
<td>25.</td>
<td>11</td>
<td></td>
<td>Ru: (2,1) * (4,2) =&gt; (3,3) Ch: 3141 8121 1631 6568 Re: 2374 8193 4991 3849 Authed? No.</td>
<td>P</td>
</tr>
<tr>
<td>26.</td>
<td>12</td>
<td>0’s rejected in Challenge Phrase</td>
<td>Ru: (1,2) + (1,1) =&gt; (1,1) Ch: 0000 0000 0000 0000 Re: 0239 4839 1203 9485 Authed? Yes.</td>
<td>F</td>
</tr>
<tr>
<td>No</td>
<td>Test Spec No.</td>
<td>Description</td>
<td>Data Used</td>
<td>P/F</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-----</td>
</tr>
</tbody>
</table>
| 27 | 13           | No random 0’s in Challenge Phrase | Ch0:1855 5622 6332 6321  
Ch1:6275 6458 3462 8414  
Ch2:7645 7418 8142 1238  
Ch3:1711 8286 3631 8833  
Ch4:3841 7632 1766 8471  
Ch5:2721 3856 7621 1881  
Ch6:6272 7214 5781 5268  
Ch7:7282 7383 3631 4222  
Ch8:7617 5538 6655 8415  
Ch9:8337 1724 2581 7813 | P |
| 28 | 14           | 0’s accepted in Response Phrase | Ru: (1,4) - (2,2) => (1,3)  
Ch: 6738 2832 8841 7438  
Re: 2903 2183 4730 0493  
Authed? Yes. | P |
| 29 | 15           | Response Phrase must be filled by user before authentication accepted. | Ru: (1,3) * (2,1) => (2,1)  
Ch: 5611 8328 4551 6226  
Re: 1239 8  
Authed? No. | P |
| 30 | 15           | ^ | Ru: (1,3) - (3,4) => (2,1)  
Ch: 5611 8328 4551 6226  
Re: 3921 0  
Authed? Yes. | F |
| 31 | 15           | ^ | Ru: (1,3) - (3,4) => (2,1)  
Ch: 5611 8328 4551 6226  
Re: <empty>  
Authed? Yes. | F |
| 32 | 16           | Three Anchors, Same Operators | Ru: (1,3) + (1,4) + (3,1)  
=> (1,4)  
Ch: 3144 6767 5146 2173  
Re: 1233 8493 0213 5839  
Authed? Yes. | P |
| 33 | 16           | ^ | Ru: (1,3) - (1,4) - (3,1) => (1,4)  
Ch: 3144 6767 5146 2173  
Re: 1235 8493 0213 5839  
Authed? Yes. | P |
<table>
<thead>
<tr>
<th>No</th>
<th>Test Spec No.</th>
<th>Description</th>
<th>Data Used</th>
<th>P/F</th>
</tr>
</thead>
</table>
| 34. | 16 | $\wedge$ | Ru: $(1,3) \times (1,4) \times (3,1)$ $\Rightarrow (1,4)$  
Ch: 3144 6767 5146 2173  
Re: 2900 2183 4730 0493  
Authed? Yes | P |
| 35. | 17 | Three Anchors, Different Operators | Ru: $(1,4) + (1,3) - (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 2718 1394 0313 3921  
Authed? Yes | P |
| 36. | 17 | $\wedge$ | Ru: $(1,4) + (1,3) - (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 5718 1394 0313 3921  
Authed? No | P |
| 37. | 17 | $\wedge$ | Ru: $(1,4) + (1,3) \times (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 5312 4839 1302 4839  
Authed? Yes | P |
| 38. | 17 | $\wedge$ | Ru: $(1,4) + (1,3) \times (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 9312 4839 1302 4839  
Authed? Yes | P |
| 39. | 17 | $\wedge$ | Ru: $(1,3) - (2,2) \times (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 0312 4839 1302 4839  
Authed? Yes | P |
| 40. | 17 | $\wedge$ | Ru: $(1,3) - (2,2) \times (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 3312 4839 1302 4839  
Authed? No. | P |
| 41. | 17 | $\wedge$ | Ru: $(1,3) - (2,2) + (1,2)$ $\Rightarrow (1,1)$  
Ch: 8561 6832 4642 8545  
Re: 3312 4839 1302 4839  
Authed? Yes | P |
<table>
<thead>
<tr>
<th>No</th>
<th>Test Spec No.</th>
<th>Description</th>
<th>Data Used</th>
<th>P/ F</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>17</td>
<td>^</td>
<td>Ru: (1,3) - (2,2) + (1,2) =&gt; (1,1) Ch: 8561 6832 4642 8545 Re: 9312 4839 1302 4839 Authed? No</td>
<td>P</td>
</tr>
<tr>
<td>43</td>
<td>18</td>
<td>Four Anchors, Three different Operators</td>
<td>Ru: (3,2) + (2,1) - (1,4) * (4,3) =&gt; (4,2) Ch: 8561 6832 4642 8545 Re: 2713 1847 3627 3148 Authed? Yes</td>
<td>P</td>
</tr>
<tr>
<td>44</td>
<td>18</td>
<td>^</td>
<td>Ru: (3,2) + (2,1) - (1,4) * (4,3) =&gt; (4,2) Ch: 8561 6832 4642 8545 Re: 2713 1847 3627 3948 Authed? No</td>
<td>P</td>
</tr>
<tr>
<td>45</td>
<td>19</td>
<td>Negative Values not computed</td>
<td>Ru: (1,2) - (1,1) =&gt; (1,1) Ch: 8561 6832 4642 8545 Re: -1342 2819 4839 4039 Authed? No</td>
<td>P</td>
</tr>
<tr>
<td>46</td>
<td>^</td>
<td></td>
<td>Ru: (1,2) - (1,1) =&gt; (1,3) Ch: 8561 6832 4642 8545 Re: 13-1 2819 4839 4039 Authed? No (Exception Thrown)</td>
<td>P</td>
</tr>
<tr>
<td>47</td>
<td>20</td>
<td>Multiple Rules are Supported</td>
<td>Currently indemonstrable, but theoretically possible</td>
<td>F</td>
</tr>
</tbody>
</table>

4.2 Analysis
Out of a total of forty seven tests, four have failed after the final development process. Three are relating to the use of 0’s and one is currently impossible to demonstrate within the current testing environment. This means that User Requirements numbered eleven and nine were either not entirely met, or were missed completely, and requirement 15 could possibly work.

Test 26 failed because 0’s in the Challenge Phrase were computed, and could be used to successfully authenticate a user. This is marginally bad as on an end-user implementation you should not expect a user to be able to change their Challenge Phrase, but equally you have to accept that it may be possible and should be defended against. It seems quite certain that the bug restricting this functionality occurs within a small snippet of code [See appendix 1]

Bug fixing would require the inspection of the overridden attribute “L”, confirming if the setter is called and weather the if statement is also functionally correct.

1 There is no technical reason why looping through rules, using R.B.R., could not be used to achieve this functionality, assuming all other base functionality exists.
Tests 30 and 31 have a problem in which firstly there are currently no client-sided validations, and secondly the default value of Letters is “0”. Having placeholder Letters, that are initialised and accessible, are currently a functional requirement for generating a Challenge Phrase. This could be fixed by changing the variable holding the value of the letter to nullable, and should be initialised as null. Then when it is accessed before being assigned an actual value, an exception is thrown. As other parts of the code require initialisation of objects, for things like Challenge generation, using nullable variables both accurately depicts the passed data, and provides more correct functionality.

Test 20 should pass by simply looping through of a user’s ruleset. However, there are a number of possible problems that could arise from this. If a user somehow manages to set two rules which point to the same Response Letter, this could cause issues. If, after the Operations, the answer to each Rule is different, but the Response Anchor in each of the Rules is the same, the user will find it impossible to authenticate. This should be made impossible during the registration process.

However, apart from the discussed issues, the basic user requirements have been met. An observation is that it appeared quite easy to compute the simple Rules when they were on the screen, but this would not be representative of an actual login process. Work would need to be done to determine how well an average user could both remember, and compute, multiple Rules.
5 Discussion

The major problems discussed before where that: passwords were easy to observe, that authentication methods regularly rely on information that is not secret, that sometimes authenticating was harder to achieve on mobile devices, and that the secret information is what’s being passed around in order to authenticate a user.

The proposed solution deals with these problems effectively. Although a Challenge and Response pair might be observed, there is only a one in roughly ten quadrillion chance\(^2\) that the same Challenge Phrase would appear next time. It is also difficult for an attacker to be given a Challenge and Response pair and know how to authenticate to any Challenge Phrase.

This poses a question, does a Challenge need to be exactly identical to another Challenge for the authentication to still pass? No. This scales with the total number of Anchors with each of the user’s Rules. The scale being \(9^n\) Anchors used, because each Anchor can refer to a different Letter within the Challenge Phrase, which would then affect the authentication in the Response Phrase. Two, three Anchor rules, have about half a million different possible combinations which affect the two Response Anchors, which only have a tiny \(10^2\) number of combinations.

This shows that a large number of very simple rules provide a much more precise authentication method than a small number of very complex rules. A user will not be inherently aware of this fact when making rules, perhaps a small diagram that moves as the user changes their rules could help bring this point across. This idea will be discussed further in the Further Work section.

From this perspective, it appears that Ruling contains a similar problem to those that where inherent to the biometrics methods: There is a degree of error which could still result in a successful authentication of a non-authentic user, and that users with similar Rules could possibly authenticate as each other. These are still problems with passwords and usernames as well, but to a much lesser extent. But the fact remains that Rulesets that are different will not authenticate to the same Challenge and Response Phrases all the time – the rules will still be secure, and it should be possible for a service to notice a large number of failed attempts.

Ruling shines here; all other authentication methods rely on information that is readily passed around. Ruling’s secret information, as long as the initial transmission was secret, remains as secret as possible, and is never openly revealed publicly. It is possible to observe fifty authentic Challenge and Response Phrases and still not know the user’s Ruleset for certain. At best you might be able to narrow the Rules down; at worst you know fifty out of ten quadrillion authentication possibilities, which is a huge improvement over static passwords.

If you want to keep your secret information out of the hands of potential attackers, you’ve got to stop passing it around. Ruling achieves this completely at the user side by eliminating the need to ever re-input secret information. The idea that easily observable information, physical objects, or physical attributes of the users themselves, are classed as “secret” is absurd. If systems need to keep secrets safe, Ruling demonstrates one possibility of ensuring that information is only transmitted by the end user, once.

Ruling also addresses the problem caused by mobile devices and their relative difficulty, compared to desktops with full-sized keyboards, regarding user input. By reducing the character

\(^2\) A 16 digit Challenge Phrase constructed with 9 out of the 10 possible letters for each digit, has \(9^{16}\) possible combinations.
set to only numerical digits, this problem has been effectively solved. Using language to authenticate a user rather than mathematics may well be more secure, but it is also going to be much more complicated and difficult to remember.

The main difficulty with Ruling is how hard it is for a user to remember their Ruleset. This is something that was not tested for, and should be considered before implementing Ruling at all. It could be the case that lots of simple rules, around four for example, could be very easy to remember. It could be the case that a few, around two for example, complex rules are simple to remember. It could be the case that a single rule is actually very difficult to remember.

Ruling has not addressed the problem of users who find it difficult to remember passwords. It remains a problem because remembering secret information must be done by the user who wishes to use that secret information to authenticate. Having users write down their rules would break the principle of secrecy employed in this design, and is undesirable.

### 5.1 Potential Further Work

None of the material in the literature review discussed the topic of *when a password is at its most vulnerable*. No research was found into this subject and the assumption was made that a password is vulnerable when a user is authenticating with it. Finding out exactly when and where secrets such as passwords are at their most vulnerable for being compromised would have made a good addition to the literature review in this paper. It is not known if such research exists.

As a user registers using Ruling, it might be difficult to make the user understand how to make a “good” Ruleset. Implementing a radar plot might offer a solution for representing this complex concept to users. This would have been considered if the implementation was more than just a proof of concept.
6 Conclusion

In conclusion, current authentication methods lack in terms of security of secret information. Users who attempt to authenticate will be regularly offering opportunities for the capture of their secret data to potential attackers. It is not difficult to acquire a single user’s password if you have the power of observing them logging in. Relying on physical objects as a means of authentication only arbitrates the security in your system to how secure the physical object is. Authenticating purely based on the user’s physical properties also relies on data that is simply not a secret by definition.

Ruling, the rule based authentication method, relies on a set of rules that are transmitted once during registration. This helps ensures that the secret information remains a secret. A potential attacker might be able to observe several different challenges and responses, but they will not be able to observe how a user creates their responses.

Ruling may be slightly less precise than using a password, and remembering a Ruleset might be difficult for a user. However, preventing the transactions of secret data between users and systems will help reduce threats from observational attacks, such as key logging or shoulder surfing.
Bibliography


Goodin, D., 2012. A screenshot from ocl-Hashcat as it cracks a list of password hashes leaked online. [Art] (http://arstechnica.com/).


Appendix

1. Class CLetter Extract

```csharp
// CLETTERMAX = 9
// CLETTERMIN = 1
public class CLetter : Letter
{
   ...
    public CLetter(int l)
    : this()
    {
        L = l;
    }
    ...
    public override int L
    {
        get
        {
            return _l;
        }
        set
        {
            if (value <= CLETTERMAX || value >= CLETTERMIN) _l = value;
            else throw new ArgumentException("Incorrect Challenge Letter Value");
        }
    }
}
```
namespace RulingTestBed
{
    public partial class TestPage : System.Web.UI.Page
    {
        Ruling.Rule r = new Ruling.Rule();
        protected void Page_Load(object sender, EventArgs e)
        {
            if (IsPostBack)
            {
                r = new Ruling.Rule(Request.Form["UserRule"]);
            }
            tb_Challenge.Value = Ruling.GenerateChallenge().ToString();
            tb_myrule.Text = r.ToString();
        }
    }
}
3. TestForm.aspx

```csharp

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body runat="server">
<form id="ruleform" method="post">
  <div>
    <label>User Rule</label>
    <input type="text" name="UserRule" value="" id="tb_UserRule" required="required" />
  </div>
  <label>Challenge:<br />
    <input type="text" name="tb1" id="tb_Challenge" runat="server" />
  </label>
  <asp:TextBox ID="tb1" runat="server" ReadOnly="True"></asp:TextBox>
  <br />
  <div>
    <label>Response:<br />
      <input type="text" name="ResponsePhrase" id="tb_Resp" />
    </label>
  <div>
  <label System Rule:
    <asp:TextBox ID="tb_myrule" runat="server" Width="415px"></asp:TextBox>
  <div>
  <label Authed?<br />
    <input type="checkbox" name="chk_Auth" id="chk_Auth" runat="server" aria-readonly="True" />
  <label>
  &nbsp;
  <input type="submit" value="Submit" class="Submit" />
  <div></form>
</body>
</html>
```
4. Ruling.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace RulingTestBed.App_Code
{
    public class Ruling
    {
        private static Random rnd = new Random();
        public static readonly string REGEXOPERATIONS = "[+\-]";
        public static readonly int LETTERMIN = 0,
            LETTERMAX = 9,
            CLETTERMIN = 1,
            CLETTERMAX = 9;

        public class Anchor
        {
            public int Word { get; set; }
            public int Letter { get; set; }
            public Anchor() { }
            public Anchor(int word, int letter)
            {
                Word = word - 1;
                Letter = letter - 1;
            }
            public Anchor(string anchor)
            {
                anchor = anchor.Replace("(", "");
                anchor = anchor.Replace(")", "");
                string[] s = anchor.Split(',');
                Word = int.Parse(s[0]) - 1;
                Letter = int.Parse(s[1]) - 1;
            }
            public override string ToString()
            {
                return String.Format("({0},{1})", Word + 1, Letter + 1);
            }
        }

        public enum Operator
        {
            ADD,
            SUB,
            MUL
        }

        public class Letter
        {
            protected int _l;
            public virtual int L
            {
                get
                {
                    return _l;
                }
                set
                {
                    if (value <= LETTERMAX || value >= LETTERMIN) _l = value;
                    else throw new ArgumentException("Incorrect Letter Value");
                }
            }
        }
    }
}
public Letter()
{
}

public Letter(int letter)
: this()
{
    L = letter;
}

public Letter(char letter)
: this(int.Parse(letter.ToString()))
{
}

public override string ToString()
{
    return L.ToString();
}

public class CLetter : Letter
{
    public CLetter()
        : base()
    {
    }
    public CLetter(int l)
        : this()
    {
        L = l;
    }
    public CLetter(char l)
        : this(int.Parse(l.ToString()))
    {
    }

    public override int L
    {
        get
        {
            return _l;
        }
        set
        {
            if (value <= CLETTERMAX || value >= CLETTERMIN) _l = value;
            else throw new ArgumentException("Incorrect Challenge Letter Value");
        }
    }
}

public class Word : IEnumerable<Letter>
{
    public Letter[] _Word { get; set; }
    public Word()
    {
            new Letter(),
            new Letter(),
            new Letter(),
            new Letter()
        };
    }
}
A Rule Based Authentication Method By Simon Ranson

```csharp
public Word(IEnumerable<Letter> word) : this()
{
    if (word.Count() == 4)
    {
        _Word[0] = word.ToArray()[0];
        _Word[1] = word.ToArray()[1];
        _Word[2] = word.ToArray()[2];
        _Word[3] = word.ToArray()[3];
    }
    else
    {
        _Word[0] = new Letter(0);
        _Word[1] = new Letter(0);
        _Word[2] = new Letter(0);
        _Word[3] = new Letter(0);
    }
}

public Word(IEnumerable<char> word) : this()
{
    if (word.Count() == 4)
    {
        _Word[0] = new Letter(word.ToArray()[0]);
        _Word[1] = new Letter(word.ToArray()[1]);
        _Word[2] = new Letter(word.ToArray()[2]);
        _Word[3] = new Letter(word.ToArray()[3]);
    }
    else
    {
        _Word[0] = new Letter(0);
        _Word[1] = new Letter(0);
        _Word[2] = new Letter(0);
        _Word[3] = new Letter(0);
    }
}

public Letter GetLetter(int i)
{
    return _Word[i];
}

public IEnumerator<Letter> GetEnumerator()
{
    foreach (Letter x in _Word)
    {
        if (x == null)
        {
            break;
        }
        yield return x;
    }
}

{
    return this.GetEnumerator();
}

public override string ToString()
{
    return String.Format("{0}{1}{2}{3}", _Word[0], _Word[1], _Word[2], _Word[3]);
}
```
public class CWord : IEnumerable<CLetter>
{
    public CLetter[] _CWord { get; set; }
    public CWord()
    {
            new CLetter(),
            new CLetter(),
            new CLetter(),
            new CLetter()
        };
    }
    public CWord(IEnumerable<CLetter> cword) : this()
    {
        if (cword.Count() == 4)
        {
            _CWord[0] = cword.ToArray()[0];
            _CWord[1] = cword.ToArray()[1];
            _CWord[2] = cword.ToArray()[2];
            _CWord[3] = cword.ToArray()[3];
        }
        else throw new ArgumentException("Invalid Challenge Word");
    }
    public CWord(IEnumerable<char> cword) : this()
    {
        if (cword.Count() == 4)
        {
            _CWord[0] = new CLetter(cword.ToArray()[0]);
            _CWord[1] = new CLetter(cword.ToArray()[1]);
            _CWord[2] = new CLetter(cword.ToArray()[2]);
            _CWord[3] = new CLetter(cword.ToArray()[3]);
        }
        else throw new ArgumentException("Invalid Challenge Word");
    }
    public CLetter GetLetter(int l)
    {
        return _CWord[l];
    }
    public IEnumerator<CLetter> GetEnumerator()
    {
        foreach (CLetter x in _CWord)
        {
            if (x == null)
            {
                break;
            }

            yield return x;
        }
    }
    {
        return this.GetEnumerator();
    }
    public override string ToString()
{  
    return String.Format("{0}{1}{2}{3}", _CWord[0], _CWord[1], _CWord[2], _CWord[3]);
}

**public class Phrase : IEnumerable<Word>**

{  
    public Word[] _Phrase { get; set; }  

    public Phrase()
    {  
            new Word(),
            new Word(),
            new Word(),
            new Word()
        };
    }

    public Phrase(IEnumerable<Word> phrase) : this()
    {  
        if (phrase.Count() == 4)
        {  
            _Phrase[0] = phrase.ToArray()[0];
            _Phrase[1] = phrase.ToArray()[1];
            _Phrase[2] = phrase.ToArray()[2];
            _Phrase[3] = phrase.ToArray()[3];
        } else
        {  
            _Phrase[0] = new Word();
            _Phrase[1] = new Word();
        }
    }

    public Phrase(IEnumerable<string> phrase) : this()
    {  
        if (phrase.Count() == 4)
        {  
            _Phrase[0] = new Word(phrase.ToArray()[0]);
            _Phrase[1] = new Word(phrase.ToArray()[1]);
            _Phrase[2] = new Word(phrase.ToArray()[2]);
            _Phrase[3] = new Word(phrase.ToArray()[3]);
        } else
        {  
            _Phrase[0] = new Word();
            _Phrase[1] = new Word();
        }
    }

    public Phrase(string phrase) : this(phrase.Split(' '))
    {  
        }

    **public Letter GetLetter(Anchor a)**
    {  
        return _Phrase[a.Word].GetLetter(a.Letter);
    }
}
public IEnumerator<Word> GetEnumerator()
{
    foreach (Word x in _Phrase)
    {
        if (x == null)
        {
            break;
        }
        yield return x;
    }
}

{
    return this.GetEnumerator();
}

public override string ToString()
{
    return String.Format("{0} {1} {2} {3} {4}", _Phrase[0], _Phrase[1], _Phrase[2],
    _Phrase[3], _Phrase[4]);
}

public class CPhrase : IEnumerable<CWord>
{
    public CWord[] _CPhrase { get; set; }
    public CPhrase()
    {
        _CPhrase = new CWord[4]{
            new CWord(),
            new CWord(),
            new CWord(),
            new CWord()};
    }
    public CPhrase(IEnumerable<CWord> cphrase)
    : this()
    {
        if (cphrase.Count() == 4)
        {
            _CPhrase[0] = cphrase.ToArray()[0];
            _CPhrase[1] = cphrase.ToArray()[1];
            _CPhrase[2] = cphrase.ToArray()[2];
            _CPhrase[3] = cphrase.ToArray()[3];
        }
        else throw new ArgumentException("Invalid Challenge Phrase");
    }
    public CPhrase(IEnumerable<string> cphrase)
    : this()
    {
        if (cphrase.Count() == 4)
        {
            _CPhrase[0] = new CWord(cphrase.ToArray()[0]);
            _CPhrase[1] = new CWord(cphrase.ToArray()[1]);
            _CPhrase[2] = new CWord(cphrase.ToArray()[2]);
            _CPhrase[3] = new CWord(cphrase.ToArray()[3]);
        }
        else throw new ArgumentException("Invalid Challenge Phrase");
    }
    public CPhrase(string cphrase)
    : this(cphrase.Split(" "))
    {=}
public CLetter GetLetter(Anchor a) {
    return _CPhrase[a.Word].GetLetter(a.Letter);
}

public IEnumerator<CWord> GetEnumerator() {
    foreach (CWord x in _CPhrase) {
        if (x == null) {
            break;
        }
        yield return x;
    }
}

System.Collections.IEnumerable System.Collections.IEnumerable.GetEnumerator() {
    return this.GetEnumerator();
}

public override string ToString() {
    return String.Format("{0} {1} {2} {3}\n", _CPhrase[0], _CPhrase[1], _CPhrase[2], _CPhrase[3]);
}

public class Rule {
    public List<Anchor> CAnchors { get; set; }
    public Anchor RAnchor { get; set; }
    public List<Operator> Operators { get; set; }
    public Rule() {
        CAnchors = new List<Anchor>();
        RAnchor = new Anchor();
        Operators = new List<Operator>();
    }
    public Rule(List<Anchor> canchors, Anchor ranchor, List<Operator> operators) : this() {
        if (canchors.Count == operators.Count + 1) {
            CAnchors.AddRange(canchors);
            RAnchor = ranchor;
            Operators.AddRange(operators);
        } else throw new ArgumentException("Number of provided Challenge Anchors not consistent with number of Operations.");
    }
    public Rule(string rule) : this() {
        //remove whitespace
        rule = rule.Replace(	', ');
        rule = rule.Replace(
', ');
        rule = rule.Replace(', ');
    }
}
A Rule Based Authentication Method By Simon Ranson

rule = rule.Replace(" ", "");

//find the Response Anchor
RAnchor = new Anchor(rule.Substring(rule.IndexOf("=>") + 2));
rule = rule.Remove(rule.IndexOf("=>"));

//Find all Challenge Anchors
foreach (string s in Regex.Split(rule, REGEXOPERATIONS))
{
    CAnchors.Add(new Anchor(s));
}

//Find all the Operations
Match ops = Regex.Match(rule, REGEXOPERATIONS);
while (ops.Success)
{
    switch (ops.ToString()[0])
    {
    case '*':
        Operators.Add(Operator.MUL);
        break;
    case '+':
        Operators.Add(Operator.ADD);
        break;
    case '-':
        Operators.Add(Operator.SUB);
        break;
    default: throw new ArgumentException("Unknown Operator encountered!");
    }
    ops = ops.NextMatch();
}

public override string ToString()
{
    if (CAnchors.Count <= 1) return "";

sb.Append(CAnchors[0]);
for (int i = 1; i <= CAnchors.Count - 1; i++)
{
    switch (Operators[i - 1])
    {
    case Operator.ADD:
        sb.Append(String.Format(" + {0}", CAnchors[i]));
        break;
    case Operator.SUB:
        sb.Append(String.Format(" - {0}", CAnchors[i]));
        break;
    case Operator.MUL:
        sb.Append(String.Format(" * {0}", CAnchors[i]));
        break;
    default:
        throw new Exception("Unknown Operator encountered!");
    }
}
    sb.Append(String.Format(" => {0}", RAnchor));
    return sb.ToString();
}

public static CPhrase GenerateChallenge()
A Rule Based Authentication Method By Simon Ranson

```csharp
{  
    CPhrase cp = new CPhrase();
    for (int w = 0; w < 4; w++)
    {
        for (int l = 0; l < 4; l++)
        {
            cp._CPhrase[w]._CWord[l].L = rnd.Next(CLETTERMIN, CLETTERMAX);
        }
    }
    return cp;
}

public static bool Evaluate(CPhrase cphrase, Phrase rphrase, Rule rule)
{
    int total = 0;
    total += cphrase.GetLetter(rule.CAnchors[0]).L;
    for (int i = 1; i <= rule.CAnchors.Count - 1; i++)
    {
        switch (rule.Operators[i - 1])
        {
            case Operator.ADD:
                total += cphrase.GetLetter(rule.CAnchors[i]).L;
                break;
            case Operator.SUB:
                total -= cphrase.GetLetter(rule.CAnchors[i]).L;
                break;
            case Operator.MUL:
                total *= cphrase.GetLetter(rule.CAnchors[i]).L;
                break;
            default:
                throw new Exception("Unrecognised Operator in Rule!");
        }
    }
    return rphrase.GetLetter(rule.RAnchor).L == Math.Abs(total % 10);
}
```