
How to Prevent Phishing

By Paul Almond

15 March 2010

4 Beechwood Cottages
Roman Road
Blackburn
Lancashire
BB1 2LB
United Kingdom

Telephone: +44-7838-448-634
USA Telephone: 507-298-2517
Website: <http://www.paul-almond.com>
E-mail: info@paul-almond.com
Skype: paulalmond01

A method is described for preventing phishing. In a secure login process, user details (such as the username and password) are combined with some identifying code associated with the website, and automatically obtained from it by the web browser, and a cryptographic hash function applied to this information, generating a hash code that is sent to the website. An obvious example of such a unique identifying code is the website's address, although a code associated with a digital signature, identifying the owner of the website, could be used. The hash code must match the value expected from a user for a successful login. The information sent to the server in a secure login depends on what is being logged into, causing the information stolen by a fake website, in a secure login, to differ from what the real website expects. The method involves ensuring that the user makes secure logins, in which the hashing process as described above occurs. If this happens, the user is safe, even if he/she logs into a fake website. Secure logins are made hard to avoid. They are associated with specific signals and actions that the user receives and performs, that are unique to a secure login. Various methods of achieving this are described.

Table of Contents

1 Introduction	4
2 Cryptographic Process	5
3 Ensuring a Secure Login	8
4 Getting Started.....	14
5 More Detailed Consideration.....	16
5.1 Efficient Processing	16
5.2 Delivering Extra Information with the Hash Code	16
5.3 Extra hashing can be used.	18
5.4 Alternatives to Using the Web Address.....	21
5.5 Preventing Replay Attacks and Making Things More Difficult	22
5.6 An Alternative to Cryptographic Hash Functions	23
6 Conclusion.....	25

List of Abbreviations

IP Internet Protocol

1 Introduction

Phishing is a type of crime in which a criminal entity pretends to be a legitimate entity in order to persuade people to give it sensitive information that they would normally only give to the legitimate entity. On the Internet, it often takes the form of criminals setting up websites mimicking the websites of organizations such as payment service providers or banks. Users are often tricked into visiting these fake websites by links in e-mail messages. When they attempt to log in on the fake websites, their user details are stolen and used to log into the real websites.

This article will suggest a way of dealing with this kind of phishing. The solution is in two parts, which are intended to work together, and it is the way these two parts are combined that is supposed to make phishing much more difficult. The first part of the solution is a cryptographic method which ensures that when a type of login that I will call a *secure login* occurs, it is impossible to steal user details, such as usernames or passwords, *even if they are used in an attempt to log into a fake website* – the information being obtained in such a phishing attack being made useless to the criminal. The second part of the solution consists of measures to ensure that all relevant logins attempted by a user will actually be secure logins.

The article covers the general idea, and then gives a more detailed discussion which will not interest all readers. Anyone who just wants to know about the general idea should read pages 5-14 and then go to the conclusion on Page 25.

2 Cryptographic Process

The method relies on a *cryptographic hash function*. This is a type of algorithm which takes some data and generates a code. It is infeasible, given the hash code, to work out what the original data was or, without knowing the original data, to generate data that, when the hash function is applied to it, causes it to generate that hash code. Cryptographic hash functions are sometimes described as *one-way encryption*. For what is described here, everyone can be assumed to be using the same cryptographic hash function: There does not need to be anything secret about the algorithm.

When a user wants to log in to a website, the following happens.

The user goes to the website which he/she wishes to log into.

e.g. The user goes to www.bigbank.com.

The user's web browser enters a "secure login" mode. The user enters his/her login details (e.g. username and password). This is the login data.

e.g. The login data is:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

The web browser then automatically adds the web address of the website currently being used (or possibly the full address of the login page), to the login data. (At this stage I am using the web address for simplicity. In reality, all we need is something which a fake website cannot give to the web browser. There are other things that we may use, as will be explained later.)

e.g. The login data is now:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8
Web Address: www.bigbank.com

The cryptographic hash function is now applied to the login data, generating a hash code. The login data is all treated as one unit, so you cannot resolve individual fields apart.

e.g. We now have:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

The hash of the login data is sent to the website's server.¹

When the hash of the login data arrives at the website's server, the actual data cannot be recovered: A hash function cannot be reversed efficiently. However, for any user, the web server "knows" the username and password, and of course it knows the web address of the page on which the login attempts will be made. This allows the web server to look up the account details for any user and apply the hash function to them, generating the hash code just as it should be arriving for a successful login by that user. The web server can compare this data with the hash code it receives during a login, and if a match is found, it knows that this is a successful login by the relevant user.

e.g. The web server receives the following data from the web browser.

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

(and the web server does not know what this is a hash of – it just has the code.)

The web server knows that for user JSMITH3921, the login data should be as follows. (It gets this just by looking up the account details for user JSMITH3921 and knowing that www.bigbank.com is the web address.)

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8
Web Address: www.bigbank.com

When the web server applies the hash function to this data it gets:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

When the web server compares this to the hash code it received when user JSMITH3921 logged in, a match will occur, and the web server will know that this is a successful login by user JSMITH3921. If no match is found, then it knows that the person trying to log in did not submit correct login details for any user; however, importantly, *the web server was not able to read, directly, the data submitted in the login attempt.*

Now, suppose the user is tricked by a phishing attempt. Instead of going to the website he/she meant to use, the user goes to a fake website. This website attempts to mimic the login process to capture the login details. The web browser enters the secure login mode. The user's login details with the website's address are hashed – using the same hash function as before – and sent to the fake website's server. However, the criminals are unable to determine the login details from this, because nobody can – not even the people who run the real website. What they have is a hashed version of those details,

¹ Standard encryption might be used as well, so before it is sent all the data might be encrypted using conventional, public key encryption: I will ignore such things for the most part here, as it is not important to what we are doing.

but that data includes the fake web address rather than the real one. If they try to submit it to the real website, it will not match the hash code of the *real* data.

e.g. JSMITH3921 is tricked into going to www.bigscambank.com, instead of www.bigbank.com, where a secure login occurs.

As with the real login, he enters the correct username and password to generate the following login data:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

As with the real login, the web address is automatically added to the login data by the web browser. This gives the following login data:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8
Web Address: www.bigscambank.com

Because a secure login is in effect, the web browser hashes this, giving:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigscambank.com)

which is then sent to the www.bigscambank.com server, where it is retrieved by the criminals.

The information is of no use to the criminals. There is no way of reversing the hash function to retrieve the original data of JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigscambank.com. They cannot use it to find out the username and password. What if they use the stolen hash code itself to fake a login to www.bigbank.com, causing the hash code that they captured to be submitted as if it were the data from a real login? That will not work either. The www.bigbank.com server is expecting:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

but it receives:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigscambank.com).

The web address is wrong – and they cannot just change the hash code so that it corresponds to the correct web address: With cryptographic hash functions you cannot choose the hash code that you are going to generate, but can only see what hash code results from some data. This will not match anything at www.bigbank.com's server, and cannot result in a successful login. The phishing attempt has failed.

3 Ensuring a Secure Login

For both the real login, and the login on the fake website, I have assumed that the web browser is in this secure login mode in which the login details are packaged together with the web address and processed by the hash function. Provided that this happens, there is no easy way to steal a user's login details. Suppose, however, that a user could be tricked into going to a website and entering a username and password while the web browser was *not* in the secure login mode. This would cause login details to be sent to the fake website's server without being hashed as required by the secure login process. The criminals would then know the actual login details, and could use them on the real website.

This problem is not as serious as it may sound. We have a single, special mode for a browser to be in – the secure login – and we are merely trying to ensure that users only try to log in while the web browser is in such a mode. The strength of the approach described here is that it forces the criminals to perform the fake login without the web browser being in the secure login mode: *It creates a well-defined difference between the genuine website and the fake website, and it forces the criminals to make their website different in this way.* Any website that is not using the secure login is, by definition, behaving differently: As soon as a fake website tries to use a secure login, it loses the ability to read the login information that is sent to it. We can use this to make the experience of logging into genuine websites – with the secure login – and phishing websites – without the secure login – so different that it will be hard for most people to be caught out. Furthermore, the process is standardized. A user need only become accustomed to logging in only with the web browser in the secure login mode for accounts on various websites to be protected.

We should be trying to make the user experience of a secure login as different as possible from the user experience of any other login. We can do this in a number of ways, so that it is hard to make a mistake.

An obvious starting point is to make the secure login *look* different. We could have the web browser show the user when a secure login is in process – possibly by showing a special symbol. This is similar to the way in which a padlock symbol is shown in browsers during encrypted browsing sessions, such as during online shopping. If the secure login symbol is not being shown (or the user is not being given whatever indication the web browser is supposed to be giving of a secure login) then the user should not attempt to log in. We should try to make such an indication that a secure login is in progress as obvious as possible, so that things do not look right if the user is on a fake website.

This may help, but it is not failsafe. Some users may proceed to log in, no matter how different a website or their browser looks. A further way of making secure logins different is to force the user to behave differently when making them. The idea of this is that if a user is always required to do certain things to perform a secure login, on any

website, and these things cannot be performed while not making a secure login – or, they cause a conventional login to turn into a secure login – when the user reaches the fake website, his/her natural behavior will be to try to perform the normal actions needed for a secure login, with the result that either the login turns into a secure login, making the phishing attempt fail, or the user finds it impossible to perform these actions, due to a secure login not being in progress, and realizes that there is a problem.

We should start by considering how a secure login session is initiated. Conventionally encrypted browsing sessions are initiated by the website. The user reaches a page that is sent from a secure server (https://) and this starts an encrypted browsing session. A secure login session should result in a similar way – when the website on the other end initiates it – but when a website has initiated a secure login session, the user may also be required to request, explicitly, a secure login session before being allowed to proceed. For example, the user may be required to click on a special button in the web browser, or to select a secure login option from the web browser's menu. The point of this is that the user is being required to do it every time he/she logs into a website that requires a secure login, and will get into the habit of doing it without thinking. When the user is tricked into trying to log in on a fake website, his/her first instinct will be to initiate the secure login, as is always required. One of two things might then be made to happen.

- The login might be turned into a secure login, because the user has initiated a secure login, even though the fake website has not requested one. (i.e. a secure login occurs when initiated by the website or when just initiated by the user.) A warning may also be displayed indicating that the website is a fake website.
- The user may be prevented from proceeding with the login, and told why. A “cooling off” period may also be imposed, during which the user is unable to log in on the website. This would be to prevent a frustrated and impatient user immediately trying to log in again without initiating a secure login.

Each of these has its merits. The first protects the user by turning the login into a secure login, but does not otherwise try to interfere with him/her, so that the user can continue through the login process – now useless to the fake website – and then see whatever the fake website is going to show/him her. One advantage of this is that, by not immediately interrupting the login process, it may make it less likely that an impatient user, frustrated by his/her inability to log in, does something stupid – in the form of trying again without initiating a secure login.

The second approach has the advantage of immediately stopping the user's interaction with a clearly fake website. The disadvantage is that the user may immediately respond by trying to work round it, by logging in again without initiating a secure login; however, a “cooling off” period should help with this.

Another way of getting a user to initiate a secure login, and it could be used in combination with the above one, is to have the user do it *implicitly*, by always putting some sequence of characters into the user details (such as username and/or password) that acts like an explicit request to initiate a secure login. The user would be required to enter these characters as part of the user details during real, secure logins: Otherwise the user details would be invalid. When the user tries to log in on the fake website, his/her natural behavior will be to enter the normal user details, which will include the sequences of characters that initiate a secure login, and even if the user has forgotten to start a secure login explicitly (or such an approach is not being used) one of the approaches described above for the user explicitly requesting a secure login will be followed.

e.g. A user may have the following login details.

Username: SLOGIN*JSMITH3921
Password: SLOGIN*BL3JE0PW21MUYWD8

Every time the user logs in he/she is required to enter the above username and password. The inclusion of "SLOGIN*" at the start of the username and password acts in the same way as an explicit instruction to initiate a secure login – whether the user explicitly requested it or not.

There could be situations where, by coincidence, the user has to enter the sequence that initiates the secure login as part of some other sequence of characters, when a secure login is not occurring. It should be possible to choose fairly short sequences of characters for which this problem does not arise very often. When it does arise, the user may be allowed to request, explicitly, that a secure login session does not proceed. Another solution would be for the user to be able to enter the sequence that initiates a secure login, without it being regarded as instruction to initiate a secure login, if some other, specific *canceling sequence* is entered with it. If there is a canceling sequence, it should not be possible for it to be included in any of the relevant login details. We might also use keys such as cursor keys or backspace keys in such an approach – to get sequences of key presses that are unlikely to appear in other contexts.

e.g. If the user enters SLOGIN*BL3JE0PW21MUYWD8 on a website, then this is regarded as implying an instruction to initiate a secure login. However, if the user enters CNCL*SLOGIN*BL3JE0PW21MUYWD8 this is not regarded as an instruction to initiate a secure login because, in this case, CNCL* is regarded as the canceling sequence. When the user submits this, CNCL* is removed and only SLOGIN*BL3JE0PW21MUYWD8 is transmitted.

One problem with this approach is that a user may mistype a password, causing the sequence that is supposed implicitly to request a secure login to be incorrect, so that a secure login is not initiated, and the criminals see something that makes the password obvious.

e.g. A user with the password SLOGIN*BL3JE0PW21MUYWD8 mistypes the password, entering it as SLPGIN*BL3JE0PW21MUYWD8. A secure login is not initiated, the login proceeds as a normal website form submission, and the criminals see SLPGIN*BL3JE0PW21MUYWD8: It will be obvious to them that the correct password is SLOGIN*BL3JE0PW21MUYWD8. One way of dealing with this is to incorporate special sequences of characters into more than one field – as was the case in the previous example, which had them in the username and password fields, so that a user would have to mistype both to avoid a secure login. It is still possible, though less likely, that the user could mistype all the fields. There is a way of dealing with this, which I will now explain.

Instead of having the special sequence of characters in one place, we might put some special character or sequence of characters in multiple places in any of the login fields. This might seem to require that the user perform a lot of repetitive typing, but we might have some alternative way of generating it, such as pressing a particular combination of keys which is not used for any other purpose when completing online forms – or, if used, must be preceded by a canceling instruction. We might invent a special character which appears in the online form for this purpose – and when this character appears in a login field, a unique sequence of bits would be sent to the server that is not sent for any other character or sequence of characters. From the user’s point of view, the easiest way of doing this would be to have a special key on the keyboard which generates a special character, with a unique appearance, that occurs multiple times in usernames, passwords, etc. and which acts as an implicit instruction to start a secure login.

e.g. A user may have the following login details:

Username: JSMITH☒3921
Password: BL3☒☒JE0PW21☒MUYW☒D8☒

and ☒ is a special character² obtained by some special keyboard operation, such as typing a particular sequence of characters and/or pressing some combination of keys – or by pressing a special “password character” button on the keyboard.

The advantage of a special character like this is that if it occurs multiple times in a password, or other login field, it is practically inconceivable that someone could make a typing error and miss out *all* of the special characters, while leaving the rest of the password or other login fields reasonably intact.

e.g. A user with the following login details:

² Yes, I just use the Wingdings font to get ☒, as of course I do not have any such special key on my keyboard. In reality, we should probably want to design some unique character.

Username: JSMITH☒3921
Password: BL3☒☒JE0PW21☒MUYW☒D8☒

would need to mistype the username and password as:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

even though the user should be used to having to type it correctly every time he/she logs in.

Even more security would be provided with more than one type of special character: If these were missing from the password (and this would take considerable user error), it would be even more difficult to guess the correct password. We might even imagine a whole set of special characters that go into the password. Maybe we could use the function keys on the keyboard on the keyboard for this purpose – and ensure that they cannot be used like this during a normal website form entry. There is at least one keyboard in existence that can change the appearance of its keys, using a display device in each key.³ A keyboard like this could be used to configure keys to enter a special character set – possibly by adding a few characters to the existing character set – and the mapping of special characters to keys could be changed each time, to prevent a user automatically entering data.

Instead of (or as well as) having a specific sequence to implicitly request a secure login, usernames and/or passwords for websites could be made to satisfy some mathematical rule, and entry of any text that satisfies that rule would act as an implicit instruction to initiate a secure login. There is a problem with this. If the rule applies to a high proportion of possible sequences of characters, many “false positives” will result, and the web browser will regard many sequences of characters as implicitly requesting a secure login, when it is just coincidence that they have the right mathematical properties. On the other hand, if the rule applies to only a small proportion of possible sequences of characters, it will limit the number of passwords that can exist with a certain length, making passwords easier to guess: This problem could be resolved by making passwords (and another field to which this applies) longer. There will always be the possibility of some sequence of characters coincidentally satisfying the rule, and some way of allowing the user to deal with this may be needed. As before, there is the possibility of the user mistyping the password, in this case entering a password that does not follow the mathematical rule, but the risk of this can be reduced by having more than one field satisfy a mathematical rule. My view is that a special character, as described previously is a better solution than this.

³ the *Optimus Maximus* keyboard, by Art. Lebedev Studio:
<http://www.artlebedev.com/everything/optimus/>.

That deals with some ways in which a secure login may be explicitly or implicitly requested. Some general features of the approach would always be the same. If the user is required explicitly to request a secure login, then the user would always be required to do this, to proceed further, by any website that requested a secure login. If the user requested a secure login for a website that had not requested one, then the secure login mode would be entered anyway, making any information that the website obtained worthless, or the login process would be stopped or disrupted in some other way. If a secure login is implicitly requested, such as by a sequence of characters, or a special character, in one of the login fields, it would be treated similarly: Either a secure login would be started, even if the website had not requested one, or the login would be disrupted.

A final way of ensuring that users only make secure logins is to ensure that the user experience, in terms of behavior required by the user once a secure login is in progress, is very different from the behavior required to submit data to a website when a secure login is not in progress. The idea of this is that the user will be in the habit of performing very specific actions for a secure login and, if trying to log in on a website where a secure login is not in progress, will be unable to do this. The idea is to make the user interface for a secure login completely different, in both appearance and the actions needed to interact with it, from any conventional website interaction: The user interface provided by the web browser for a secure login should be completely different to that used in other website interaction – ideally, so different that without it the user has no idea what to do. People could have lots of ideas about this. The user might be required to enter the login details into a special window (but if this is done, it must be done in such a way that it cannot be easily mimicked by a fake website using a popup window). The user might be required to go to the web browser's menu bar and pull down a menu, whereupon the login fields appear in the menu, rather than in a window. The user might be required to click on a separate desktop icon, or on the taskbar, to open what appears to be a separate program, just for the purposes of entering the login details. There are various ways in which this could be done.

4 Getting Started

Ideally, an approach like the one proposed here would be immediately adopted as a standard. The capability of providing a secure login would be built into all web browsers and the owners of websites which deal with sensitive user information would all start providing secure login functionality. This is not going to happen: It is not that easy to get a standard adopted.

A more realistic approach would be to start making secure logins available without it being adopted as a standard, so that it may become a standard later as use grows. At first, secure login capability for the client could be provided by an extra program, which augments the web browser. There might be more than one program which does this, and the capability may be built into some existing programs which augment browsers. Software to provide secure login at the server side would also be written, and made available to owners of websites, with technical documentation for programmers about how to implement secure login on a website. The system might not just be offered to organizations such as banks and payment service providers: It may be a good idea to ensure that any system is accessible to most website owners.

It may be necessary, at first, to make secure login an option, but to provide an alternative way of logging on if the computer is not set up to perform a secure login. Some users may want secure login only and their accounts could be set up to provide that. One way of getting lots of computers enabled to perform secure logins might be for the organizations that want their customers to start using it to provide them with the software; for example, as a download from their own websites.

If secure logins like this were widely available, it may be feasible to modify keyboard designs to provide extra characters, so that passwords could be arranged to make non-secure entry of them impossible.

Some more detailed consideration about what has been proposed now follows. This will not interest all, or even most, readers. If you were reading the article to get the general idea of what was being proposed, you may now wish to miss this out and go to the conclusion on Page 25.

We might consider password manager programs in all this. These are programs which store passwords, usually encrypted, for different websites, so that a user can retrieve them easily, usually with a single password. These programs can typically retrieve the correct passwords to log into websites automatically. Such programs probably make phishing harder, as they reduce the human element involved in logging into websites. Do we need a solution to phishing at all if they exist?

Even with such programs, conventional login by the user is still required sometimes. Some of them work by detecting the user's first login to the website, and capturing it

and storing the details. If this first login was made to the wrong website, user details could be at risk. Sometimes, a website may change and cause the password manager to fail to log the user in, so that the user may attempt another login to store the details again. The user may receive an e-mail from criminals, with a link to a fake website. On arriving at the website, the password manager would not work - as this is a fake website, it will not recognize it – but the user may try to login manually. Some people may be away from a computer where they have a password manager installed. This problem can be reduced to some degree by various ways of making password managers portable by having the passwords stored in hardware that the user takes with him/her, or by use of online accounts, but I suggest that there is a case for ensuring that computers have a capability to allow any users to make logins that are safe from phishing. It may be that later the idea of humans logging in to separate systems disappears, and we move on to a different system. That different system might involve a website providing a digital signature and the web browser, or some other program, sending a user's login details automatically, with or without any human interface being involved. While logins by humans are still widely expected, however, and while websites make provision for them, there seems to be a case for dealing with phishing.

5 More Detailed Consideration

5.1 Efficient Processing

I have described an approach in which the website receives nothing except the hash of the login details (comprising the username, password and/or any other login fields and the web address). This hash code then needs to be checked against the entire user database. This could be done by running through all the user accounts, generating the hash code for a login, but it would be more efficient to do this in advance, so that the hash code that should result from a login by each user is stored in a list. When a hash code is received from a user attempting to log in, it can be compared against this pre-existing list. The process could be made very quick with database indexing.

5.2 Delivering Extra Information with the Hash Code

An alternative implementation might involve also sending the username unhashed, so that the user account can be easily looked up without having to check every user record, and without having to store the hash codes for all the users. When the information is received from a user attempting to log in, the relevant user could be looked up, and then the relevant hash code for the login details retrieved or computed, before being compared with the received hash code.

e.g. For the following user:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

logging in at www.bigbank.com, the following information may be sent to the website's server.

Username: JSMITH3921
Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

When this information is received, the username can be used to look up the user record for JSMITH3921. The expected hash code can then be retrieved or computed, to see if it matches the received hash code.

An alternative to this would be to provide some user data, such as a partial username, unhashed, that may not specify the user completely, but which narrows down the number of users that it could be.

e.g. For the following user:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

logging in at www.bigbank.com, the following information may be sent to the website's server.

Username: JSMIT*****
Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

When this information is received, the partial username JSMIT***** is used to look up any user records which could match it and in each case the expected hash code can then be retrieved or computed, to see if it matches the received hash code.

Neither of these approaches is desirable. If a phishing attempt occurs they protect the password, but they allow the criminals to obtain a username or partial username. Criminals should be prevented from obtaining any information at all. An approach like this might be considered if there are a lot of users, as an alternative to computing the expected hash code for every user when a login attempt is made, but a better method would be just to pre-compute and store the hash codes, as previously described. This has the problem that the hash codes all change if the web address changes (though that problem does not occur if something other than the web address is used as an identifier of the website, as discussed later), but this problem can be avoided, without the security issues already mentioned, by sending an extra hash code, made by applying the cryptographic hash function to the user's login details only. This would be sent with the hash code corresponding to the login details and the information identifying the website. The first hash code would only be used to locate a user, and the second would be used to check that the login is valid. A phishing attempt which uses a secure login would get the first hash code correct, as it is non-site specific, but not the second one.

e.g. For the following user:

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

e.g. For user JSMITH3921, logging in at www.bigbank.com, the following information may be sent to the website's server.

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8)
Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

When this information is received by the web server, the server first looks for a user record with the hash code Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) – and it has these hash codes pre-computed and stored with the user records. When it finds a match, it checks that the second hash code – in this case Hash of (JSMITH3921 +

BL3JE0PW21MUYWD8 + www.bigbank.com) matches the user. This second hash code could be stored in the user record, or computed when needed.

We might be concerned about the (very) small risk of two users getting the same hash code. The website owners would know about any such situation, as it would be evident from their database. We could eliminate it completely by using just a hash of the username (possibly with the web address or other site identifier) to identify the user.

e.g. For user JSMITH3921, logging in at www.bigbank.com, the following information may be sent to the website's server.

Hash of (JSMITH3921 + www.bigbank.com)

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

When adding a new user record, the hash code corresponding to the username would be computed and stored with that record. If a "collision" occurred – and this is very unlikely – a different username would be assigned. The reason that the username is separated from the password for this is because a user might change his/her password later – although this could be prevented by assigning passwords automatically.

Sending even a hash code corresponding to a single user field like this might be undesirable, because it could allow criminals to steal a hash code that they know corresponds to only a single field, making a brute force attack aimed at guessing this field easier. Such an attack would involve trying to find any sequences of characters which, when the cryptographic hash function was applied to them, produced that hash code. It is better if any hash code derived from one field is always derived from the other, to make this more difficult. The computational work needed to find a match would be incredibly difficult – and even that it is only a match: It does not guarantee that you really have the right user details, as you could have just found details that "collide" with them when hashed. The small chance of collision could be eliminated completely anyway, by ensuring that the server checks for possible collisions anytime a username and password combination is assigned – including when a password is changed – so it may be preferable to avoid using any extra identifying information.

5.3 Extra hashing can be used.

Storing passwords in clear text is often considered ill-advised because, if the password file is stolen, the passwords in it can be successfully used. Some readers may have concerns that the approach discussed here has that problem, as it may seem that passwords are stored in clear text on the server, so that the hash codes can be computed from them, or that the hash code corresponding to the login details is stored in the server, and is expected to be sent from a user – so that someone might steal the hash codes from the server and then simply send them to it.

An important point about this is that this is an entirely different kind of attack to phishing. It requires the security of the server to be severely compromised – effectively the criminals have to be already “in” the server – and anyone being successfully attacked in this way has serious issues anyway. I should, however, at least answer this.

The method as discussed here is not intended to deal with this kind of attack, but is only to deal with phishing. There may be lots of other things going on, related to security, that are not a feature of this proposal, because they do not directly relate to phishing. While hashing is generally used to protect against this kind of threat, it is being used here for a different purpose, and not to prevent phishing. If we want to protect against theft of information from the server, we can apply *extra* hashing in this way.

Here are some ways in which this might be implemented.

Suppose user account details are stored in a database, with the passwords in clear text, and the hash codes to verify a user are generated as and when needed in response to a login attempt.

e.g. A user account record is as follows.

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

and this information is combined with the web address, and hashed, giving:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

to see if the result matches the hash code received from a user.

This involves storing the password as clear text, creating a possible security threat. We could avoid this, and avoid having to compute the hash code every time anyone logs in, by computing it in advance and storing it in the user record, meaning that we do not need the password itself anymore.

e.g. The account record for JSMITH3921 might be:

Username: JSMITH3921
Hash of (JSMITH3921 + www.bigbank.com) (used to allow the correct user to be found)
Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com)

This avoids the need to store the password as plain text, or to compute the hash code each time it is needed for a check, but it now raises the issue of the hash code itself being stolen by hacking the server, and being simply transmitted, directly, from another computer. This problem can be avoided if the hash code itself is hashed before being placed in the user account record.

e.g. The account record for JSMITH3921 might be:

Username: JSMITH3921

Hash of (JSMITH3921 + www.bigbank.com) (used to allow the correct user to be found)

Hash of (Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + www.bigbank.com))

Any hash code received during a login attempt would be hashed at the server, and then a search would be performed for a user record containing the resulting hash code.

We could avoid the issue of storage of passwords in clear text, and retain the ability to change the web address (or other website identifier) and password without problems, if we hash the login details and then combine this with the code (such as the web address) that identifies the website.

e.g. For the following user:

Username: JSMITH3921

Password: BL3JE0PW21MUYWD8

logging in at www.bigbank.com, the following occurs.

Hash of (JSMITH3921 + www.bigbank.com) is computed. (This is to reference the user at the server.)

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) is computed.

This is used to compute:

Hash of (Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) + www.bigbank.com).

The following information is then sent to the server:

Hash of (JSMITH3921 + www.bigbank.com)

Hash of (Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) + www.bigbank.com)

At the server the first hash code, Hash of (JSMITH3921 + www.bigbank.com), is used to look up the user record. The server stores this hash code in each user's account record.

When the user record is found, the second hash code, Hash of (Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) + www.bigbank.com) is compared with what would be expected for this user – and this hash code can be pre-computed for the user or computed when needed: It can also be recomputed if the password or web address (or another site identified which is used, changes). The server needs to store the value of Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) to allow this to be computed, and it may also store the value of Hash of (Hash of (JSMITH3921 + BL3JE0PW21MUYWD8) + www.bigbank.com) to save time.

The idea of including the web address in the hashed data is only one way of ensuring that the data is changed if it is sent to the wrong website, and I only used this for simplicity. Things other than a web address, which also cannot be faked by another

website, could be included in the login details that are hashed. I will now describe such alternatives to including the web address in the hashed data.

5.4 Alternatives to Using the Web Address

The approach described so far involves combining the user details (such as username or password) with the web address of the website. The idea of this is for the web browser to include something identifying the website, which cannot be duplicated without permission by another website, so that if a fake website is used, the hash code that it receives will not match the one for the genuine website.

I have described this in terms of web addresses, but this is mainly for simplicity. In reality, all that we need to hash with the login data is some unique identifier, which is available to a web browser when a login is occurring on a website, and which cannot be easily faked. As just discussed, it may be undesirable for all the hash codes to change if the web address changes. As I have just mentioned, this issue can be avoided by hashing the username and password separately, and then combining this with site identified to be hashed; however other approaches may be preferred.

An alternative may be to use the website's Internet Protocol (IP) address.

e.g. User JSMITH3921 has the following account details.

Username: JSMITH3921
Password: BL3JE0PW21MUYWD8

When JSMITH3921 logs in at www.bigbank.com, which has an IP address of 208.77.188.166, the following information may be sent to the website's server:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + 208.77.188.166)

This has the advantage of not relying on the domain name staying the same; however it could still cause problems if the IP address has to be changed.

Another approach is for the website owner to have an ID code, associated with an account held with a trusted organization, and which is looked up by the user's web browser using the web address or IP address. This ID code would then be combined with the rest of the login data before it is hashed, so that the information sent to the website's server is something like Hash of (username + password + ID). The trusted organization in this case might hold similar account details for many website owners.

e.g. A user with username JSMITH3921 and password BL3JE0PW21MUYWD8 logs into www.bigbank.com.

The user's website browser goes to a trusted server and requests the ID code for www.bigbank.com. The code ID107048103848 is returned.

The username, password and ID code are combined and hashed, giving:

Hash of (JSMITH3921 + BL3JE0PW21MUYWD8 + ID107048103848).

This is then sent to the website's server, where it is checked, as before, to see if it matches the expected hash code from a user.

If a user logs in at a fake website, when the web address or IP address of the fake website is used to look up the ID code on the trusted server, it cannot give the correct ID code. Even if the criminals open an account on the trusted server, they cannot have the same account as the owner of the genuine website, and so cannot cause the same ID to be generated. The hash code will therefore not be what was expected.

If the owner of the real website needs to change the web address or the IP address, he/she can just update the details of the account on the trusted server, associating a new website or IP address with his/her user ID. The owner could also add more web addresses or IP addresses to the user ID, so that the user's login details can be made to work on other websites, but only if the owner of the ID authorizes it.

Another solution is to use a digital signature provided by the website. This would allow the genuine website to provide an ID code, which could then be included with the hashed data as already described. If the user is tricked into making a secure login on a fake website, then it will be unable to provide the same digital signature as the real website, and the hash code sent by the user will not match the expected hash code.

A digital signature would protect against an attack that involved interfering with a user's computer so that the browser's information about the address of the website being visited is wrong; for example, the user's web browser "thinks" it is at www.bigbank.com, but really it is at www.bigscambank.com.⁴ However, to do this in the first place would mean you had access to the user's computer, which implies all kinds of problems anyway, and that things have already gone beyond phishing by this stage.

5.5 Preventing Replay Attacks and Making Things More Difficult

We might add extra, session-specific data to the hash code, to prevent an intercepted transmission being used in a replay attack. This data could include the time at which the login occurred and the IP address of the computer on which the login is being made.

e.g. A user with username JSMITH3921 and password BL3JE0PW21MUYWD8 logs into www.bigbank.com from IP address 208.77.188.166 on 31 January 2015 at 10:30AM. As

⁴ On a Windows computer, this could be done by changing the contents of the HOSTS file, which "points" specific web addresses at specific IP addresses.

well as a hash code to identify the user, the following information is sent to the web server might be as follows:

IP Address: 208.77.188.166
Date, Time: 31 January 2015, 10:30AM
Hash of (JSMITH3921 + BL3JE0PW21MU YWD8 + www.bigbank.com
+ 208.77.188.166 + 31 January 2015, 10:30AM)

The login attempt would be rejected if the date/time is too far in the past. To perform a replay attack with an intercepted transmission you would need to generate the above hash code again – and you cannot do this unless you know the username and password.

Another way of doing this is for the web server to send a pseudo-randomly generated code to the web browser, which is different for every login, and to require that this code is added to the data which is going to be hashed to verify the user.

If hashed user data is going to be sent separately, to allow the correct user to be found, it may be desirable to ensure that this data becomes outdated as quickly as possible; however, it may be undesirable for such a hash code to change on every login attempt, as the information stored in the database to allow records to be found quickly would not be usable. One solution would be to require that the approximate date/time is hashed with the information being used to identify the user. Another solution is for the web server to send a code, which is combined with this data before hashing it, the code being changed regularly and the relevant hash codes stored in the user records being updated in the user records being updated at the same time.

We might also request some extra information from the user, such as a date of birth, and add it to the data being hashed, to make things more difficult for anyone trying to work with a hash code corresponding to this data.

This is not part of the main approach being suggested here, and I have only included it to show that an approach like this can be combined with other ways of providing security.

It should be apparent that there are numerous ways of dealing with the user data, not all of which are described here, and a specific approach can be chosen based on the general idea.

5.6 An Alternative to Cryptographic Hash Functions

I have described the process working by using a cryptographic hash function. An alternative approach might be to encrypt the data using public key cryptography, with everyone having the encryption key available, but making sure that nobody knows the decryption key: that it has been destroyed. This would work just like a standard, cryptographic hash function. The public key used for this purpose might be built into the

web browser as a “common”, public key, or it may be provided when the user goes to the website. If the public key comes from an external source, it is important that the web browser can be assured that it comes from a trusted source, and that nobody knows the decryption key. For example, it might be associated with a special security certificate for this purpose. Using a standard, straightforward, cryptographic hash function may be a more practical approach.

Using public key encryption has one advantage, in that the issue of collisions is avoided completely. There would be no need to separate details needed to identify the user from the password: It could be all be sent as one encrypted message, *which nobody would ever be able to decrypt*, even the recipient. It would be desirable not to separate any fields from the user data, because it might conceivably help some brute force attack aimed at working out what the user details are from the hash codes.

With such an approach, the encrypted message expected from each user would be stored on the server, and when a login attempt was made the encrypted message would be treated just as the hash codes described previously, with a check being made for a match with a user record.

Whether this is worth doing is debatable. The occurrence of collisions with cryptographic hash codes is of very low probability, and even this small risk can be eliminated completely if the server takes steps to prevent it when usernames and passwords are being assigned, or when passwords are being changed.

6 Conclusion

A method has been described for dealing with the problem of phishing. The method has two main parts.

The first part of the method involves using cryptographic hash functions in a *secure login* process so that user details (such as the username and password) are combined with some identifying code associated with the website and a cryptographic hash function applied to this information, generating a hash code that is sent to the website. The identifying code associated with the website needs to be something that the web browser can obtain, automatically, when the user is trying to log in at a website, and it must be something that a fake website cannot easily provide to a user's web browser. An obvious example of such a unique identifying code is the website's web address, as the web browser has this information and a website cannot easily present another website's address to the web browser. Alternatively, the website's IP address may be used. Another solution may be to use a code associated with a digital signature, identifying the owner of the website.

When the hash code is received by the web server it is compared with the hash codes expected from known users. If a match is found, this is a successful login by the relevant user. If no match is found, then the login fails.

This works, because *the information sent to the web server in a secure login is dependent on what is being logged into*. If a secure login occurs for a fake website, the hash code which it receives will be one based on the identifying code (such as the web address) associated with the fake website, and it will be useless for trying to log into the website. It is not practical for anyone to extract the user details themselves from such a hash code.

That is the cryptographic part of the solution. We now need to deal with the other part: making sure that the user performs secure logins. Provided that the user makes secure logins, in which the hashing process as described above occurs, the user is safe, even if he/she logs into a fake website. The only real problem could occur if the user made a conventional login into a fake website, revealing his/her user details directly.

The solution is to make secure logins hard to avoid. They need to be associated with specific signals that the user receives, and specific actions that the user performs, that can never happen while not performing a secure login. One possibility is to show a symbol on screen during a secure login, in the same way that a padlock symbol is shown by browsers during encrypted browsing sessions. Users may ignore the absence of such a symbol, however. Other solutions are based on the idea of getting the user into the habit of performing certain actions during a secure login, and ensuring that these actions always either initiate a secure login (whether one has been requested or not) or disrupt further interaction with the fake website. One way of doing this is to require the

user to issue an *explicit* instruction to begin a secure login at the start of every secure login, even though a secure login will have been requested by the website. Another way of doing it is to incorporate information that will be regarded as an *implicit* instruction to begin a secure login session into the user information itself, so that the user's normal behavior of entering this information will act just like an explicit instruction to begin a secure login. This information might be special sequences of characters or it may be special characters which are reserved for this purpose. We might imagine a keyboard with a special key for this purpose. An incidental benefit of this is that, by putting special characters at random positions in a password, the password is made invalid if the user misses them out, and the criminals then have to guess where the missing characters should be, while if the special characters are included then a secure login is started and the information is made useless to the criminals – and the difficulty would be increased for the criminals if more than one special character were contained in the password. Another way of getting the user into particular habits for secure logins is to use a completely different user interface for secure logins, so that the user's experience is very different to that in normal Internet use. The idea is that if this interface is not provided, the user would not be able to perform his/her usual behavior in logging in, and would be suspicious.

The idea behind all this is to make a secure login so different from a normal login that it will be difficult for a fake website to trick a user without using a secure login itself – and if it does so it turns on the cryptographic hashing process which makes any information it gets from the user useless.

An approach like this would work best if adopted as a standard, but this would not happen immediately. In the early stages, it may be a good idea to provide a program which can allow secure logins for the client, and software and documentation which can be used to provide secure logins on servers.

I have described this approach in terms of websites and web browsers, but it could be applied in other contexts.