

Authenticating Factors

The Battle for Two Factor Authentication Secure Science Corporation

As security developers we all would like to see our clients performing their tasks online in a manner that provides them the lowest probability of falling victim to cyber attacks such as eavesdropping, keylogging and phishing. However, as many financial institutions still employ single factor authentication (i.e. password) techniques, their clients are still at risk. Worse yet, many online services will e-mail a non-encrypted password back to a client upon request from any source.

Single-factor authentication attackers can come at this from many different angles.

In this paper, when we speak of ‘*credentials*’ we are referring to the pieces of information which uniquely identify a person, group or entity from all others in the same category. For instance, a person has a name. However, names are not always unique. There are more than one ‘John Smiths’ in any given country let alone the entire world. So we invent other means of separating our identities. We have Social Security (SSN) and Social Insurance numbers (SIN) in both the US and Canada, along with an address where we live, a phone number unique to oneself, a passport ID, medical insurance cards, and credit cards (hopefully) that are hopefully unique..

Forms of Credentials

- Name
- User name
- e-mail address
- Account number
- SSN or SIN
- Passport ID

Any given person has a variety of means of proving who they are or more specifically, they are who they say they are. However, not all information is safe to release indiscriminately and therein lies one of the most significant problems facing authenticators. With a credit card for instance, one can make online purchases with essentially no other means of proving identity. That is, there is a weak sense of ‘*authentication*’. Actually with three pieces of

information (card number, back of card number and expiration date) anyone can make charges they want without a victim immediately noticing. Similarly with SSN or SIN information another could apply for credit in anyone’s name.

So clearly there are at least two classes of credentials with respect to security: those which are public domain, eg. your name or account number; and there are those credentials which are private, eg. your SSN/SIN, credit card number or passwords.

The goal of an authenticator is to somehow pair the private and public information in such a manner that will be secure over insecure mediums such as the Internet. More about specifics of these threats will be discussed later on in this paper.

We say a piece of information such as a login attempt is authentic when the credentials of the author can be bound reliably to a given transaction. For example, a famous work of art can have a name like Picasso as easily as forging the signature. The piece can only be authenticated for sale through multiple channels or ‘factors’ tested against the object's credentials. There are transactional details which are relevant to the process such as the date it was created, where, on which medium (type of canvas) and so on.

Forgery is not the only threat we have to consider in our ‘*threat model*’. With a persons credentials one could also impostor another. A classic example of which would be the e-mail ‘joe-job’ attack. In this attack the attacker will send e-mails (or other forms of posts such as usenet) with a forged ‘from’ address. The messages will usually incite other third parties to then email a response back thinking the from address is authentic.

An attacker may also want the ability to eavesdrop on a body of information. This is useful for corporate espionage. In this attack role the attacker will usually attempt to obtain login credentials for a system they wish to spy on. How they obtain the credentials can mirror exactly how an attacker attempts to obtain banking or other financial credentials.

Overall e-mail messages are by far the weakest link we will examine in this paper. They are rarely signed, even less likely to be verified (in any meaningful manner) and often subject to header modifications to obscure the origins of the message.

In terms of online transactions, the transactional details usually include the users Internet protocol (IP) address, server time and date, transaction number, user name or other piece of public credential. These pieces of information are usually used in an 'after the fact' forensics scenario where a forgery has already been committed and law enforcement are attempting to track down the offender.

Cryptography is the field of study dedicated to all aspects of computer security. It includes methods of achieving privacy, integrity, authentication and non-repudiation with mechanical means.

Privacy¹ is a fairly well known concept that is often misrepresented in the media. Privacy is the status of concealment of meaning. That is, a message may be 'privacy protected', given out over an insecure medium such as the Internet (or phone) and not have the meaning of the message leaked to eavesdroppers. For example, a credit card number given out over the web is usually protected in this manner.

The usual manner of achieving privacy is through the use of a ciphering algorithm. Popular examples include the relatively new Advanced Encryption Standard² (AES) and the older RSA stream cipher RC4. Both of these share in the fact that they are 'symmetric key' algorithms³. That means that both of the recipient and sender must share an identical key to handle the message.

Integrity and authenticity are closely related in intent but not limits. Both deal with the state of a message in transit with regards to the original. Where they differ is that integrity is entirely based on public information whereas authenticity is based on private

information (shared or otherwise).

The usual means of achieving both is with a one-way 'hash function' which accepts as input a variable length message and produces a fixed length output. Hash functions⁴ have several useful cryptographic properties which allows them to be used throughout authentication protocols. Of importance to this paper are two key features, collision resistance and one-wayness. The former implies given two distinct messages M and M' when sent through the hash function should produce two distinct outputs. The latter implies that given the output of a hash function determining the input should be hard.

Non-repudiation is a rarely used property in online activities which is really unfortunate as it can help relieve quite a bit of the problems faced by clients today. Non-repudiation is the inability to refuse acceptance of a binding agreement. For example, if you sign a loan contract you cannot later refuse to accept the terms. In cryptographic terms this is accomplished with what is known as a 'digital signature' on an email or other electronic content.

Digital signatures are accomplished with 'asymmetric key' algorithms such as RSA, DSA or Elliptic Curve Cryptography (ECC) variations (such as EC-DSA). These are usually lumped into the category of 'public key algorithms'⁵.

The most common place to see digital signatures is on Secure Socket Layer (SSL) certificates as issued by Certificate Authorities (CA). A certificate is simply a set of credentials such as the owners name, date issued, date of expiry and issuing authority combined with public and private credentials known as the public and private key.

When a company such as *Geotrust* issues an SSL certificate, they digitally sign the public information contained in the certificate. Next when you browse a website using SSL (e.g. by going to a site with the https protocol) your browser will download the certificate and verify the signature using the CA

1 About privacy: <http://en.wikipedia.org/wiki/Privacy>

2 AES information: <http://csrc.nist.gov/CryptoToolkit/aes/>

3 Symmetric key algorithms:
http://en.wikipedia.org/wiki/Symmetric_key

4 Hash functions: http://en.wikipedia.org/wiki/Hash_function

5 Public Key Cryptography:
http://en.wikipedia.org/wiki/Public_key_cryptography

public keys stored in the web browser. The logic being that the public CA certificates that come with your web browser are a trustworthy source to compare against. Therefore, if the signature on the certificate of the website you are visiting is valid, then the site must also be trustworthy.

Threat models are what security experts design, review and scrutinize to ensure users are safe. 'Safe' from what, is the question they answer. To first understand how to create a threat model one must first understand the system being protected, where its vulnerabilities are and where the liabilities are exposed.

Let us consider a simple web based e-mail system. This works well since many people have them and they are easy to understand. The users public credentials can be a name, user name or e-mail address. Typically the user name and e-mail address are directly related. E.g. 'John Walker' would map to 'jwalker@company.com'. So what is the immediate liability? If a user must only give their public credential it would mean that anyone including attackers could use the service as any particular user.

The obvious and most often chosen solution to this problem is to have a user create a password. This will be their sole private credential they can offer. The login is considered authentic if the pair of public and private information is valid for the given service.

This introduces a new attacker consideration which is the '*window of vulnerability*'. Once a user creates a password, the service is vulnerable for the lifetime of the password. That is, if an attacker gets a hold of the password they can abuse the system so long as the password does not change.

This introduces another obvious vulnerability that the password would be sent over the medium as 'plaintext', or not encrypted. This means the meaning of the message (e.g. the contents of the password) would be readable and copyable by any eavesdropper. There are several solutions to this problem but the two most common are challenge-response systems and encrypted logins.

A challenge-response system is where you must prove that you hold a particular private credential, not by handing it over, but by responding to a query about it. For example, if a financial institution knows your date of birth they could ask you your astrological sign instead of the date. Obviously that is not secure and in practice it is not accomplished in that manner. A typical challenge response protocol would work as follows.

```
Client: I'm $USER and want to login.  
Server: Hash your password and this  
random string together.  
Client: The result is EAD511...
```

Here we are using a one-way function on the password and a server supplied random string. The server has the password so it can compute the hash as well and compare what the client returns. The resulting reply does not directly leak the password but should be hard to compute without the password.

If done correctly this reduces the window of vulnerability to this single login, as ideally the challenge response will be different on the next login attempt.

So how did we get here? Well in the original case of just a password we have

```
Client: I am $USER and want to login.  
Server: Give me your password.  
Client: It is "tj13ffd4".
```

Which as we discussed is clearly a weak method of pairing credentials. We can obviously update this with a hash as follows.

```
Client: I am $USER and want to login.  
Server: Give me the hash of your  
password.  
Client: It is E5D144F3...
```

Which seems more secure but actually is not. While you do not give out your password, you essentially are giving out exactly what an attacker needs anyways. Imagine two different services using the same scheme. Once an attacker has the hash they can use it on either of the services.

Well how about mixing in the time into the equation?

```
Client: I am $USER and want to login.  
Server: Give me the hash of your password  
and the current time.  
Client: It is F133BEA4...
```

Which sounds like it should be changing with respect to the current time. This is still vulnerable to passive eavesdropping. Imagine the two services again; an attacker can listen to the reply for one service and use it for the other.

Why are we discussing only passive eavesdropping? Quite simply it is the easiest and lowest risk form of communication tampering. The victim does not know it is happening and the attacker can carry it out for relatively high profit.

So clearly we need random challenges to avoid what essentially amount to replay attacks.

This type of solution is useful when a full blown public key infrastructure (PKI) is not practical. However, it has several limitations. While it avoids passive eavesdroppers who can only read data on the wire, it does not stop active eavesdroppers who take control of the medium after the client responds.

Once the client provides the response in the clear, anyone who is between the server and the client can assume control.

This solution is also prone to implementation difficulties. For example, a naïve implementation may pair the client response with the hash and the challenge string. Then the server simply looks at the reply and uses the client provided pairing to perform the challenge verification locally. This introduces a *'replay attack'* vulnerability where the attacker simply resends older login attempt responses to gain access.

For a correct implementation, the server has to bind a challenge query to transactional information such as the user name, IP address, time and date. Then provide mechanisms to expire and create new challenges at reasonable intervals.

Login challenges must be fresh. Typically in cryptography these fresh pieces of information (such as challenge queries) are called 'salts' or 'nonces' (the latter literally means "N once").

The second solution to the problem are encrypted logins. For example, a website may use SSL to encrypt the login data being sent. This implies (but does not mean) attackers cannot decode and read the passwords of users as they are sent over the medium.

In theory this implies the site is trustworthy and it is safe to hand over all of your relevant private information. However, what many seem to overlook is that anyone can apply for a valid SSL certificate regardless of their ethical practices. In essence, SSL with unbound certificates (also known as self-issued) are just as safe to use. All that matters is that for any server you trust, its public certificate does not change.

Other forms of 'logins' include debit transactions where the card contains your account (and other) information and you must provide the P.I.N. to access the account. Debit cards are notoriously vulnerable to card 'skimmers' which are typically placed directly inside existing bank machines. They read your magnet stripe as you enter the card then log the P.I.N. you enter. Since the P.I.N. never changes (or specifically doesn't change fast enough for most users) an attacker can immediately produce a replica card and use it.

In effect using a debit card in North America (the magstripe sort) is a lot like using your bank by sending your plaintext in the clear over the Internet. Nine times out of ten you'll be safe but once someone manages to get in between you and the bank there is nothing you can do.

Another common 'login' procedure is with service providers (e.g. cable, gas, video rental, cell phones, etc.). In an attack style known as *'social engineering'* an attacker can impersonate a legitimate service user by either determining, from the service provider or client themselves, enough information to make a convincing client.

Where security protocols get things wrong is exactly where attackers get things right. Seems like a fairly simple concept yet it is still someone many developers ignore to this day. Constructing a threat model is usually the first step before setting out to develop the software to implement a given system.

Basic user authenticators must fulfill several properties simultaneously.

1. The authenticator must take into account public credential(s) and pair it with private credential(s) that has been shared between the client and server.
2. The authenticator must be stateful and resist replay attacks.
3. The authenticator must not leak the private credentials over the communication medium.
4. Knowledge of previous successful logins must not enable an attacker to make a successful login of their own.

Property #4 is mostly a catch-all statement that implies that leaking login responses (such as to challenges) does not leak sufficient information for an attacker to produce their own credentials for valid logins.

Security usually breaks down when the static private credentials are compromised. Since they seldom change, the systems have long windows of vulnerability that attackers prey on. Attackers have several means at their disposal of extracting secrets from unsuspecting victims. The three most common methods are phishing, malware and protocol breaks.

Phishing is the act of sending out messages (usually e-mail) in an attempt to solicit information directly from a victim. They usually arrive in the form of an e-mail informing the victim to 'update' their account information. They also come as prize packages and 'free credit checks'.

The e-mail will at best look legitimate but usually they are easy enough to pick out (especially since banks don't ask for updates that way) by their typographical mistakes and common HTML errors. The e-mail will contain an HTML reference (link) to

an attacker controlled website which also tries to mimic the victims service provider. The site will typically ask the user for unrelated pieces of information all at once such as name, date of birth, credit card number and debit number. This essentially increases the value of any yield they do achieve. As sad as it sounds, many people still fall victim to such attacks even despite their obvious scam appearance.

The attacker (or phisher) is exploiting the fact that most financial institutions are still using single static forms of authentication. Once the phisher retrieves your password they can access your account as much as they want until you change the password or close the account.

Typical 'solutions' to phishing include bulk e-mail spam filters, education and websites resilient to phishers (such as those using an anti-fraud web module). None of these solutions are 100% effective, especially education, which is very expensive to all parties involved in the process.

Malware is a type of virus program specifically aimed at modifying the client side of transactions. It is meant to get to the data before it is sent encrypted over the medium. A typical type of malware would be a key logger.

A key logger records keystrokes from users and transmits them to a remote 'drop' site which the attackers can later filter through. They come in all shapes and sizes but are usually fairly non-intelligent since they log all traffic. They typically are installed as 'keyboard hook callback' programs which run in the background of the victims computer and are fed (by the operating system) every key that is pressed. Legitimate key loggers include programs which use hotkey combinations to activate.

Key logging is performed in real time but the actual exploitation of the data is performed afterwards, usually by as much as a day. This fact is important for security solutions such as login tokens, which provide an authentication 'value' (as we shall discuss shortly) that is only valid for a short period of time (typically one minute).

Another form of malware include DNS redirect programs which redirect your legitimate requests to attacker websites. For example, when you open your web browser and type (or click on a bookmark) for 'citibank.com' your computer performs a DNS request to find out what the address of the 'citibank.com' server is. A malware program could intercept this request (often as trivially as by putting entries in the hosts file) such that the returned address is not for the real 'citibank.com'.

Malware is generally harder to use but more profitable when successfully installed. It relies mostly on users opening and running attachments to e-mails or instant messaging posts but can also exploit vulnerabilities in client software, such as web browsers. Fortunately, more people are aware of viruses than phishing, so they are more likely to not fall victim.

Protocol breaks are far more rare in practice (though not in theory) than phishing or malware for the extraction of secrets. A protocol break occurs when the cryptographic process of dealing with secrets is insecure. For example, a cipher may not in fact require the secret key to determine the plaintext, as an authenticator may easily create forgeries or the digital signature may be transferable.

The most common form of protocol break are rollback attacks on session initialization protocols. These have been used against SSL v2 and the GSM A5 protocols, at least in laboratory experiments. In a rollback attack, an attacker in the middle impostors being a legitimate party of the discussion, and requests the use of cryptographic algorithms which are weaker than either party would have normally used.

For example, in SSL v2 an attacker could modify the list of server supported algorithms to limit them to ciphers with 40-bit keys (keys below 64-bits are generally considered very weak). Similarly in the GSM A5 protocol an attacker can impostor a legitimate base station and request the clients use the weaker A5/1 cipher instead of A5/2 or A5/3.

Generally protocol breaks are not generally put to practical use because they are more expensive to maintain, must be executed live and generally are working against targets which can easily upgrade software (most people for instance use SSLv3 or TLS v1).

Attackers, at least the more successful attackers, consider several factors before mounting an attack. They must consider the difficulty of the attack, the risk and the reward.

Most phishing and e-mail scams are sent as bulk e-mail and trivial to filter. The more successful attackers will send bulk e-mails which are harder to automatically filter out. Various techniques such as hash busters, word lists and MIME recoding are the more popular tricks used. Usually the difficulty factor is fairly low since many bulk mailer programs already exist. This is partly the reason why bulk e-mail is so common (another reason is just an overly trusting SMTP protocol...).

After delivery of the e-mail a phisher or malware author are not finished. They must get the victim to trust the content they are seeing. The more legitimate the e-mail appears the more likely they are to lure the victim into visiting (or running) the resulting attack.

The difficulty factor weighs heavily on the type of attack. Most attackers are not particularly tech savvy. More often than not spammers are caught by identifying features (e.g. bugs) in their bulk mailers. This is also why protocol breaks are not that common. Few, if any, reported attackers are cryptographers by training and would not typically have the talents required to investigate a protocol for weaknesses.

However, that is not a form of security since in many cases breaks to protocols can be automated. Once someone figures out how to exploit a vulnerability they can package up the exploit and let others use it.

After the actual 'how' of the attack is figured out, the attacker must then calculate the risk and profit. The risk being how likely they are to be caught, or at

least shut down and prevented from gathering profit. A phishing site setup in the United States would not be very successful, as most (but not all) Internet service providers (ISP) are diligent enough to shut down offending hosts. As a result, the sites are usually set up off-shore in other countries where shut down requests are not honored, or take some time to be initiated, allowing ample time for the phishers to gather and store data. Once they have secured a site, they then need a method of collecting the information and making use of it (e.g. trafficking money).

The phishers final step is to analyze the profit to be had by the attack. A phishing attempt which attacks users of a service that are infrequently used, would have little financial return. This is why we see more phishing attempts against large financial institutions and service providers (such as E-bay, Amazon and e-gold). The smaller institutions are rarely targeted, as most foreign phishers are not even aware of their existence, let alone consider them worthy targets.

What can be done to combat the proliferation of phishers, scammers, skimmers, carders, social engineers and other forms of attackers? It seems almost like a constant duel of attrition.

However, there are quite a few lessons that could be replicated, such as proper design of both the website and servers to avoid cross site scripting attacks, buffer overflow protection and the like. So clearly a very good initial step is to have competent developers behind the scenes.

Most phishing and scam attempts are also fairly easy to detect and avoid with minimal training. A bank, for instance, would never ask for private credentials via an e-mail. However, there are clients out there that do not think the request is unreasonable.

Aside from educating the developers and the clients/users, there are other options. First and foremost is replacing the login procedure to change how public and private credentials are paired. Second, how to establish a reliable secure connection over an inherently insecure medium (e.g. the Internet).

Hashing a password with a random server generated string is a good manner of pairing the user name and password without directly revealing the password. However, it is still vulnerable to malicious servers and requires a secure (or at least trusted) connection between the client and server.

Other pairings such as digital signatures (from the client) can work but are frequently not used in practice outside of remote SSH logins.

Cellphones usually use a symmetric algorithm (such as GSM COMP128 or CMEA) between the server and client with a known static secret. The server queries the client (cellphone) and it computes the result as a function of the shared static secret and the query. It does lack non-repudiation factors since the server can impostor being the client.

What is needed more than just one good pairing function is a protocol based on multiple factors. Usually quoted as 'things you know, things you are, things you have'. That is, you know your password, you are yourself (think biometrics) and you have a token, key, card or other device.

Typically new designs strive for at least 'two-factor authentication' which means you need at least two different *sources* of proof of your public credentials and private credentials matching up.

Multiple factors have a clear benefit over typical single factor authenticators. That is, you have to do more work to actually leak enough information to an attacker to become a victim.

For example, the current generation of smart cards used in European banking perform a cryptographic authentication on the card itself and require the user to enter a P.I.N. to unlock it. This means if the reader records the P.I.N. (which is possible due to the unfortunate location of the entry pad) it does not have enough information to victimize the user.

Another typical (though rarely enforced) form of authenticators are bank tellers which require a bank card and your presence. That is, what you have and

who you are, although most banks rarely enforce any form of stringent check. The better banks will have the tellers force the clients to enter their P.I.N. in front of them so at least skimmed cards are not usable for fraudulent activities.

This brings the discussion to an important point. More often than not people assume that two queries are the same as two-factor authentication. They are not. Consider this new policy from Scotiabank.

Maintaining the confidentiality of your information remains a priority for us. That's why we are pleased to introduce Access Code, an easy-to-use security feature that adds an extra level of identity verification to your banking and brokerage transactions. Your Access Code is a unique 5-8 digit code that is separate from your existing Sign-on password. Once you've set it up, like all passwords, it's critical to keep your Access Code confidential and never share it with anyone.

[http://www.scotiabank.com/cda/content/0,1608, CID6995_LIDen,00.html](http://www.scotiabank.com/cda/content/0,1608,CID6995_LIDen,00.html)

This implies the user will have to enter what essentially amounts to a second password. They even warn to not share it with anyone just like your original login password. This additional layer does provide some extra security in practice, but not in the same way as an additional *factor* would.

Similarly, showing a bank card and photo I.D. at a bank would not constitute two factors of authentication since they both could be forged credentials.

RSA 'SecurID' tokens⁶ are by far the most popular of additional factor authentication. They are small devices (about the size of a household key) which update periodically (typically every 20 to 60 seconds) and display a number for the user.

The way they are meant to be used involves the user connecting to an ACE compatible server and typing (usually) their user name, password and the current number shown on the token.

The displayed number is a one-way function of the current time and a secret shared between the token

and the server. The secret is supposed to be specifically shared only between *one* token and *one* server. The user themselves never know the secret which means they cannot leak it. If the token is lost it can be deactivated on the server side fairly quickly and they are inexpensive to produce a replacement.

Since the user never knows the shared secret and presumably it is difficult to determine if the token constitutes 'something you have'. Combined with 'something you know' the user can enjoy a two-factor authentication system.

While the token is a nice solution to a serious problem, it has several nasty drawbacks which slow its adoption rate.

The 'RSA SecurID (tm)' device is not based on any public standards. The algorithm is a trade secret and sadly there are numerous patents which claim to cover the device (oddly enough a few are not owned by RSA such as Patent #20030182241⁷). Such legal obstacles do nothing to promote safe computing practices and are hardly legitimate monopolies.

In fact, the trade secret algorithms used in (at least) older tokens were subject to cryptanalysis and could be compromised without depriving the client of their token for a noticeable length of time.

There are also logistical problems with the device. Suppose that all banks, services and employers used tokens such as these. One could easily have five or six of them on a key chain. It would simply be too much of a problem to carry multiple tokens around at all times.

As well as the logistical problems the shared secrets cannot be changed. Although making the secrets changeable introduces the possibility that an attacker may phish for it, there are reasonable precautions to be had.

There are also security issues to address. The

6 RSA SecurID:
<http://www.rsasecurity.com/node.asp?id=1157>

7 This patent was filed in March 2002 yet there are papers on older SECURID tokens from 1996 (<http://www.homeport.org/~adam/dimacs.html>). Clearly the 2002 patent is invalid but it still does not help adoption.

authentication of these tokens is only one way. This means that the client authenticates themselves to the server, but the server does not have to authenticate itself to the user. This also implies the cryptographic protocols used to security the medium are in no way related to the tokens, which means an active eavesdropper can still pretend to be a machine in the middle.

The ideal solution would involve a new device and protocol changes that secure the medium to lock out eavesdroppers altogether.

The new device would need to have some form of input, such as a numerical keypad with arrows. It would also have to have a multi-line display capable of displaying numbers and letters (matrix LCD would work).

The device would be capable of storing multiple time bases (relative to the remote server) as well as shared secrets. Programming of the shared secret could be done with the client on-site (e.g. at the bank), over the phone or by mail. The time base could be established off a website or by phone if no physical presence is possible. This would eliminate the need for multiple devices and is no less secure than losing a key chain full of tokens.

The device itself would be P.I.N. locked by the user as the default state. The P.I.N. chosen by the user (and user changeable). More importantly the data would be encrypted using the P.I.N. as the key. This would prevent compromising the data when the token is lost and physically compromised.

To facilitate loss prevention devices could be registered with a central company that only stores public credentials. When a user calls the company they would in turn then contact all the services related to the token, and have the token invalidated from their servers. This call center could use ANI and trivial public queries to reduce the possibility of denial of service attacks.

When a user selects a particular service with the token it would show two strings. The first string is the string they should provide the service, the second

string is what the service should provide them. The strings could be 16-digit hexadecimal numbers which are updated every 120 seconds (with a sufficient grace period). Ideally, the client should be told to verify the string they receive from the service provider (over a secured medium), to ensure its identity.

While the strings are longer than a SecurID token string, they also have more entropy (128-bits displayed instead of 26-bits for 8-digit strings) and the additional longer period should be more than enough to read and type 32 characters.

To facilitate adoption, the devices should use public domain algorithms such as the NIST standard one-way hashes and symmetric ciphers. This would avoid any single company having a monopoly on the device or its usage.

To make the medium more secure against eavesdroppers, the session should incorporate both strings in the key negotiation protocol. This would require changes to the protocol (e.g. SSL or TLS), which are less than trivial. However, a more clever approach is to do away with SSL altogether, at least for web browsing. Combined with the two strings on both sides of the communication, a Javascript implementation of a fast symmetric cipher (and authentication algorithm) would be more than adequate to encrypt or decrypt and verify document objects (DOM).

The usage of the device could be quite simple. A client visits a service website and proceeds to enter some public credential such as a user name. The bank replies with an encrypted payload (using the key derived from the two 16-digit strings the token would show), with a small unencrypted form and Javascript action attached to it.

The user would enter their two strings and password then hit the submit button which would activate the Javascript to start a session, decrypt the payload and present the client with the server 16-digit string.

This immediately cuts out passive attackers but does nothing against an active adversary who could just replace the Javascript submit action with a GET

request to an attacker website with the two strings.

A trivial solution to this problem would be to have a local HTML page which would be a *'loader'* for other websites. It would store services the client uses in a local HTML cookie, perform the cryptographic operations all locally, and display the decrypted text in a sub-frame.

To use this loader, the client would open the HTML file (say on their desktop), choose the service (or update the list of services) they want. When they select a service type, the Javascript in the loader would prompt for the username, password and the two strings, then immediately go to the encrypted login of the bank.

The login page could be loaded by submitting a GET (or XML) request with the username in the clear and the password encrypted (and authenticated with a MAC), using the key derived from the token. An adversary in the middle would see the login attempt and the encrypted password but nothing more. The server would have to track logins and prevent further logins within the same grace period.

The next line of attack would be to insert objects into the HTML returned by the server. In particular, an *'onload'* event could trigger if the page uses a GET or POST request for a page that would then attack the loader by modifying its DOM. One of many solutions would be to use an XMLHttpRequest component⁸ to fetch non-live content and then proceed to verify and decrypt it.

Care would have to be taken to ensure the encrypted protocol is secure against forgeries, eavesdropping and replay attacks. A simple message authentication algorithm (MAC) such as AES-CCM could handle both privacy and authenticity while an IV counter stored in the loader (w.r.t. this session) could prevent replay attacks.

Ideally a platform independent protocol for full duplex encrypted and authenticated communication could be developed much along the lines of the functionality of an SSL session. The Javascript

implementation would be an intermediate solution to help increase the adoption rate of the new technology. From this, essentially all applications that enjoy the use of SSL for security, could use this new two-factor form of authentication.

Trust is a problem that this solution addresses better than SSL. Where SSL uses certificates bound to a domain name signed by an authority for *'trust'*, this solution uses a shared secret the client sets up with the service directly.

In the case of SSL users rarely check the details of their connection. They look for the padlock and assume that means they are safe. However, various cross-site scripting and DNS poisoning attacks with frames can fool users into loading pages that are SSL protected but not controlled by the intended server.

With this solution an eavesdropper cannot meddle with the data going in either direction since the data will be authenticated with a shared secret key derived from the strings that the token displays.

Two or even three factor authentication protocols are fairly rare in the end user world where millions of people are today. They sign into their e-mail, banks and other services with little more than a user name and catchy password they thought of.

The current popular solutions such as the SecurID tokens are practical upgrades but can be used incorrectly, are proprietary and not scalable.

The public (and private sector) require a standard form of remote login that addresses current security problems and protects both the clients and servers from attackers. Privately held trade secret algorithms and patents are partly to blame for the slow adoption rate of two-factor authenticators. Cost is another factor, but easy to justify against the amount of fraud that occurs because of technically simple attacks such as phishing and malware.

While the technology and cryptography required to implement a scheme similar to the one discussed in this paper exists, the consensus amongst security experts is not there. In fact, there are few if many

⁸ Credit to Simon Johnson for the idea.

open discussions related to this field at all. For the most part it has been abandoned. For example, public key algorithms have been around since the 1970s, yet we are still seeing e-mails being sent as plaintext even when they contain sensitive information. Rarely do people even bother thinking about signing their e-mails and even when they do, they do so with public keys that nobody can vouch for.

A publicly accepted and practiced standard for two-factor authentication could bridge the gap between what is possible with science and what is required for the world we live in.

Secure Science Corporation would like to send an open invitation to other cryptographers and web developers to collaborate on the concept of an improved login token and the related client/server software. In particular, we look towards having a software emulated token in the near future and back end API library to work against. A public binary specification would follow shortly afterwards. Interested parties should contact SSC at securetoken@seurescience.net.