Usable Security through better use of Cryptography
Or: Stop violating Kerckhoffs' 6^{th}!

Guido Witmond
eccentric-authentication.org
Rotterdam, Netherlands
Email: guido@witmond.nl

***Abstract*—Usability and security are not at odds with each other. Bad security and lousy usability are caused by broken protocols that require the end user to make security decisions while the information needed to make a correct decision has never been provided.**

We claim that the burden of verifying security properties does not belong to the user. It's the computer's job.

This paper introduces the concept of an abstract secure channel and describes a way to implement it. We design a protocol that offers mutual authentication between a user's agent and a site. The agent manages the user's identities and performs all validations and verifications.

Lifting this burden off the user into an agent is what makes the protocol both secure and very user friendly.

I. Introduction

Many people in the cryptographic community know about Auguste Kerckhoffs and his six principles for designing cryptosystems. The best-known is his second principle that states that the only secret should be the key, not the algorithm. The sixth principle is just as important: "The system should be easy, neither requiring knowledge of a long list of rules nor involving mental strain."

It's often said that security is at arms with usability, that cryptography is hard to use and that one has optimize for one property against the other.

We believe this to be a myth. Our observation is that the main cause of security and usability failures is due to the fact that we don't do proper mutual authentication on the internet:

1) Users can't verify a site's identity. They don't know how. The computer industry has never taught them. Yet they are supposed to be able to tell their bank's site apart from a scammer.
2) Banks assume that the one who provides the password is their customer. After a single mistake – failing to detect scammers – the user hands the password to the criminals. Later, the bank can't tell the criminals apart from the customer. The only way for the bank to authenticate their customer has been lost.

In this paper, we show how usability and security reinforce each other. We deploy cryptography in a novel way. It allows us to design a user agent that does all the verifications and validations for its user. The agent provides security while making it easy to use, in line with Kerckhoffs' principles.

A. Claims

- Usability and security are not at odds with each other.
- Security problems can lead to usability problems. Usability problems can lead to security problems.
- Proper security can lead to good usability. Good usability is needed for proper security.
- Both ends need to authenticate each other before the channel is secure.
- Mutual authentication can be achieved with one party being anonymous: The user.
- Mutual authentication can be achieved programmatically. Users can rely on their agent to protect them against scammers, phishing and IP-rerouting attacks.

II. Problem description

People who do electronic banking might be familiar with an email message that mentions that a new transaction statement is available. As the email system is not designed to protect messages against disclosure and tampering, banks can't send the financial information via email. The customer is then invited to log in at the bank to retrieve it.

People believe this invitation to log in is the safe way to get the statement to the customer. It's not. Actually both security and usability are lost. Here's the protocol. The customer:

1) doesn't get the information he wants;
2) has to switch to another program: from email reader to web browser;
3) needs to navigate to the bank's website;
4) has to verify the site's server certificate;
5) has to provide the account name and password;
6) has to navigate to get the required information.

We will identify all the security and usability problems with this protocol.

Step 1 - The customer gets an empty message whose sole contents is an invitation to log in at the bank.

Due to lack of security of email, usability loses.

Step 2. The customer has to switch from email reader to web browser. This is a small extra load on the user.

A small loss of usability.

Step 3. The customer has to enter the web address of the bank. To prevent people from giving their password to

criminals, banks have to teach people never to click a link in email messages.

Due to security issues, usability loses again.

Step 4 requires the customer to make sure he has connected to the bank and not to a malicious proxy server run by scammers. Common wisdom says it is sufficient to check for the green address bar. Regrettably it's not, as criminals can acquire valid certificates too.

The customer doesn't even know how to authenticate the bank site properly: it takes a call to customer support to ask for the certificate fingerprint. If all customers would perform this step diligenty, they would detect the scammers and be safe.

The sad truth: This is too much much mental strain for people. The verification step gets skipped entirely. It's a blatant violation of the 6^{th}.

Security is lost here.

Step 5. Secrecy of the password is the only thing that protects the account, yet the user has to provide it verbatim. Hardly any site deploys zero-knowledge proofs that protects the password against disclosure. If the user even remembers the password, or it is not too easy to guess. Or shared among other sites.

Security and usability lose.

Step 6. Just after logging in, the bank doesn't know the intention of the visitor. The user has to navigate through the site to reach the required information. This is again due to the fact that the bank's email message had to be as uninformative as possible to teach people not to click on links in emails.

Here usability loses again.

There is one other aspect that has been lost too: The paper trail. The bank can rewrite the transaction history at any moment, even if only by accident. The remedy is judicious downloading and printing of each electronic statement by the customer. But who's doing that? Again, a usability issue leading to a security failure.

The combination of people not being able to verify the authenticity of the bank site and the fact that passwords get transmitted verbatim is a toxic combination.

III. Ideal

What would be the ideal situation?

The customer and the bank would share a secure channel where they exchange messages without worries about confidentiality, identity theft, message tampering or difficult procedures.

Creating that secure – worry free – channel is the topic of the rest of this paper.

A. Creating a secure channel

Claim: To create a secure channel it is necessary and sufficient that both endpoints have a way to properly authenticate the other party over the medium.

A secure channel exist when this condition is met.

Notice: A secure channel is an abstract concept, independent of actual communications. The channel exist at the very moment that both parties can authenticate each other, whether they communicate or not.

'Proof' by example:

A. Base: When two people meet in person they can communicate securely. Their personalities are their identities. They have to observe the surroundings to make sure that they aren't overheard by others. Effectively they create a secure channel. However, as soon as they part, this channel is gone.

B. Induction: When two people meet in person and agree on a password, they create a secure channel that lasts longer. They can write letters on paper and mention the password to prove authenticity to the other.

The secure channel exists from the moment they agree on the password, even before they sent the first message. The secure channel lasts until either forgets the password or if it gets leaked (by someone opening an envelope at the post office).

Keeping this channel secure requires tamper detection by both endpoints, that's the effort needed to be able to properly authenticate the other end.

Notice: They use a short lived channel from step A to create the longer lived password authenticated channel.

C. Alternative induction: Instead of writing paper letters, they could agree to use ZIP-file encryption via email as the transport method. When decryption is successful, the password proves identity.

The secure channel is lost as soon as the password is compromised. That could be a third party cracking the ZIP-encryption. Detection by the end points is difficult unless the cracker writes messages whose contents are not believable to have come from the expected sender.

D. Alternative induction: Face to face people exchange their PGP key fingerprints. They use their email program do the encryption, decryption and signature validation. All the people need to do is to protect their own private key and trust that the other side submits a revocation when that key is compromised.

E. When people recognize each others' voice, they can exchange their PGP-fingerprints via telephone. This shows the secure channel can be created over a distance. However, it requires that the people know each other a priori to learn each other's voice characteristics.

Aside: The web of trust has been debunked as a way to introduce keys. It requires people to assess if someome has followed the proper procedure of verifying the name on a key against that person's passport before signing the key. Again, this is an example where a user needs to make a security decision without sufficient information.

IV. Authentication at a distance

The common theme of these examples above is that the secure channel starts with face to face contact. In formal words: both endpoints need to mutually authenticate the other.

If mutual authentication is not possible, there is no secure channel.

In many cases there is no face to face contact. Yet, we need a secure channel over a distance.

To create a secure channel means that two endpoints need to authenticate each other. It doesn't require that both endpoints use the same method. We provide two methods that complement one other.

The first method is a way for a person to authenticate a web site. This problem has already been solved. All we need is to deploy it. It's called DNSSEC and DANE.

The second method is a way for a site to recognize return visitors. We deploy well known technology in a novel configuration to achieve these goals.

Together these allow us to create a protocol that offers mutual authentication. In other words: it lets people create a secure channel so desperately needed.

A. DANE: Properly authenticating a web site

As we have seen in step 4 of our problem statement above, people can't authenticate a web site. They have been taught to look for the green address bar. But that is not sufficient as criminals can buy confusing domain names and valid certificates using stolen passports and creditcards. Or the criminals hack into a CA and issue valid certificates for the target's domain name.

Until recently the only way to be sure of connecting to the correct site was to call the help desk of that site, ask them to read out the fingerprint of their CA certificate and compare that to the actual certificate received by the browser. Chances are that the help desk people would consider you a hacker and refuse to cooperate.

This authentication problem has been solved by two developments: DNSSEC and DANE.

DNSSEC offers a validation method to make sure that the data received from a DNS-server is authentic, i.e.: you are sure the data you receive is what the domain owner has placed in the zone.

The reason it works is that every DNSSEC validation chains back to a single Root Key that is administered and guarded by ICANN. Although it's a single point of failure, the whole world is watching it like a hawk. Proof of leakage of that key guarantees world wide headlines in the news. This makes it sufficiently safe for our purpose.

DANE, is a method of specifying who is the CA that has signed their server certificate. The domain owner chooses the CA, acquires a certificate and specifies that CA in the TLSA record. DANE relies on DNSSEC to protect authenticity.

DANE is an example where usability and security go together. The browser does the validations, the human does nothing, except for typing in the domain name.

When people learn of a new bank, either by marketing, reading about it in a newspaper or referral from friends, they have no problem getting the domain name correct.

This is sufficient for a person to identify a site. The browser validates the site's server certificate against the CA specified in the DANE records. When it matches, the browser shows the page, if there is a validation error, the browser shows the error without option to bypass. People can connect to the bank's site knowing they are safe against Man-in-the-Middle attacks.

This lets people authenticate a site by their domain name. The browser does the job that us humans could never do.

So far, DANE solves half of the problem of mutual authentication.

B. Authenticating customers

We deploy DANE method to secure the connection to the bank. Banks might want to stick to the 'trusted' password scheme to authenticate their customers. After all, once the browser has validated the connection, the password gets transmitted safely every time.

Sadly, this isn't secure enough. In fact, we have increased the risk of phishing. Criminals create a lookalike site at a slightly different domain name. The browser shows a green address bar because both the server certificate and the DANE records match. No wonder, both are under criminal control.

So if our user mistakes the scammers' site for the bank, he'll lose his password. The browser is not able to help here, it doesn't know the user is providing the credentials for site X to site Y.

We better help the user here and let the browser do the authentication. We eliminate user managed passwords and replace it with a browser controlled mechanism. Our goal is to let the browser remember the credentials for site X and never provide these to site Y.

Intermezzo: "Here is your captain speaking: We boldy go where no one has gone before." In the next sections we are going to create a new method of client authentication. We introduce a lot of new concepts in a very small space. Feel free to skip to section: "Bank Example Revisited".

C. Local CA

Our bank sets up a Certificate Authority of their own. They use it to sign their server certificates. Then they specify this CA-certificate in DANE.

This is sufficient to create a safe connection between the visitor and the bank.

However, the visitor hasn't provided any identifying information yet and is still anonymous[1].

D. Client certificates

Suppose the anonymous visitor wants to sign up for a bank account. How is the bank going to authenticate her at the next visit.

The visitor needs to identify herself. It's in her best interest to do so in a way that cannot be impersonated by someone else.

[1]Except for IP-address, tracking cookies, etc.

Claim: Client certificates provide all the necessary properties.

These are the steps our prospective client takes:

- She navigates to the bank's site and presses the 'Signup'-button on her browser;
- The web browser creates a new public/private key pair.
- It creates a Certificate Signing Request (CSR) that contains nothing but the public key. No names, no email addresses, no ids. It signs the CSR using the private key to prove ownership of the key
- The browser submits the CSR to the bank via HTTP-POST.
- The bank verifies the signature and immediately signs a client certificate for the public key and returns it in the same HTTP-response.
- The browser stores the certificate in its certificate store (together with the private key).

From this moment on, the bank can recognize this visitor at later visits. Whenever the bank needs to authenticate the client it asks for a certificate login. The browser selects the correct private key and performs the client certificate login procedure[2].

The customer is anonymous, the client certificate contains no personal identifying information, yet the certificate lets the site recognize her at recurring visits.

We have designed a method for the bank to authenticate its visitors. Each visitor creates a private key and submits the public key for certification. Proof of posession of the private key is sufficient to prove identity.

We have fulfilled our requirement of mutual authentication.

Q.E.D.

After the visitor has received the certificate she can start the 'paperwork' needed to open the account, ie: provide a copy of an official identity card or passport, credit history, etc. This lets the bank tie the customers public key to her real life identity. The bank might ask the customer to show up in person at a branch office, that's up to the bank and local laws.

For many other applications, knowing the real life identity of a visitor at a site might not be needed. Examples are blog sites, web shops, newspaper sites, social media sites. Our person has a different private key for each of those. Proving real life identity at the bank does not compromise the anonymity of the other accounts.

E. Improving protections

The client certificates that are signed by the bank contain no identifying information whatsoever. As such they are only valid at the bank. As a rule, each website runs their own CA, each site accepts only their own certificates.

That is an interesting propery: As each site uses their own CA, the browser knows the difference between site X and Y. When criminals create a fake site for our bank at a slightly different and – for users – confusing domain name, they can't obtain a server certificate for it that is signed by the bank's CA. The bank is not stupid, it runs their own CA, they only sign their own server certificates, no exceptions.

The criminals need to set up their own CA. When the fake site requests authentication, the browser won't offer the certificates of the bank as it knows that the site is a different one. In fact, it's new site, the agent will offer to create an account. That should wake up the user! Thus the browser protects against phishers, typosquatters and Man-in-the-Middle attacks.

It also means that our bank can safely provide links in unsecure emails. The bank knows that the browser protects the user. Clicking links is safe again[3].

There is another interesting property:

Both the server certificate and the client certificate are signed using the same CA. We use that property in the browser: Whenever a site requests a client certificate login, the browser selects the private key based upon the CA of the server certificate.

This way we guarantee that when a domain name gets hijacked at a domain registry or an IP address gets rerouted, the browser won't try to log in there. As the hijackers don't posess the private key of the original CA, the browser correctly identifies it as a different site. The browser protects against DNS-manipulations and IP-rerouting attacks.

F. Public Key Exchange

Our protocol lets the user use domain names. At a technical level we go deeper.

The user agent uses the DANE record to learn about the site's CA.

At first contact, the agent verifies that the site's server certificate matches against the DANE record.

At signup, the agent offers a public key of its own. The site's CA signs a client certificate for the agent. This lets the site authenticate recurring visitors.

At later visits, DNSSEC and DANE are used to learn of the site's IP-address and server certificate. The identity remains the same: the site's Root Key.

We have implemented a public key exchange! The site is identified by its Local CA's public key. The user is identified by its own public key. Their private keys proof their identities.

What we have done here, we've created a public key exchange that requires no input from the user. No fingerprint validations, no key signing parties. It is fully automated, yet secure.

We can use these public keys for both authentication as well as message encryption.

After creating the client certificate, the only question for the user is: "Is this site worthy of my personal information?" But that's something each of us has to determine for ourselves.

[2]This implies TLS-renegotiation

[3]Except for malware and such, check out genode.org for a safe operating system.

V. Bank example revisited

In the sections above, we have created a secure channel over a distance. Here are the steps again:

- The customer initially identifies the site by domain name.
- The browser goes deeper and validates the server certificate against the DANE specified CA. This is sufficient to authenticate the site.
- At the user's command, the browser requests and receives a client certificate. The client certificate is signed by the site's own CA.
- When the user logs in using the certificate, the private key proves it's the same user as before.

It is this combination of the client authenticating the server by DANE and the server validating the client by site-signed public key that makes it a secure channel. Both parties have a way to authenticate the other.

That leads us to the original problem. When the bank has prepared a new transaction statement it can be sent to the customer. It is encrypted using the customers public key. Only the customer can decrypt it. Any transport medium suffices. It could be plain old email, if the user gives the mail program access to the private key. Or it could be any other messaging network that runs on the user's computer[4].

This solves all our identified problems from the problem description:

1) The customer gets the expected information: the transaction statement. The bank has signed it using their private key so you can validate its authenticity.
2) The user's computer can take care of the message and decrypt it, no need for the user to go to a different program.
3) No need to remember the banks web site address, you already have the message. The message might contain links to the site. All safe.
4) The browser validates the domain name. After signing up, it also protects against typosquatting, phishing, DNS-coercion and IP-rerouting.
5) No need to type passwords. The browser has all the information it needs to identify to the bank. Just press the 'Login'-button to log in.
6) The message received contains the required information. No need to hunt for it.
7) The bank sends the transaction statements in the messages, just refrain from deleting these to keep a 'paper trail'.

It's the browser that takes care of all the cryptographic verification details. The user just decides whether to sign up, log in and log out. If there is an error, the browser shows the error and refuses to proceed.

As the certificate is void of personal identifying information, the user stays anonymous until he reveals his identity. The question left for people is: "Do I trust this site with my info?"

[4]Sorry, no cloud services.

VI. Conclusion

First we've outlined the myth of security and usability being at odds with each other. Our example of the empty email message from a bank shows how usability and security are both lost.

Then we showed that the most important reason for the lack of usable security is that the user bears the burden of authenticating without having the information needed to make a correct decision. This is in violation of Kerckhoffs' 6^{th} principle.

We showed that using passwords is not a good idea anymore for authentication. It can increase the risk of phishing.

We've shown the importance of mutual authentication for usable security and how that can be achieved starting at a face to face meeting.

We went on to design a protocol to perform mutual authentication when there is no face to face meeting possible. The protocol has these properties:

- It performs mutual authentication;
- The transport is fully encrypted;
- The original trust anchor is the DNSSEC Root Key;
- The user agent is able to do all the verifications for its user;
- The user does not have to perform any security verifications;
- The user agent works together with a Local CA at a site to create an anonymous client certificate;
- The user stays anonymous, there is no personal identifying information in the client certificate;
- The user agent is able to protect its user against Man-in-the-Middle attacks, Phishing, Typosquatting, DNS coecion and IP-rerouting attacks.

With this protocol we've solved the problem of mutual authentication at a distance: People can authenticate websites, and sites can authenticate their users.

We believe this protocol shows that the common belief that security and usability are at odds with each other to be a myth.

We claim that this protocol aligns with Kerckhoffs' principles. All six.

Acknowledgment

The author would like to thank mr. Martin Leiser for his thorough reviewing and proofreading of this paper.