5-13-2013

User-Defined Key Pair Protocol

Omar Hassan

User-Defined Key Pair Protocol

By

Omar Hassan

Rochester Institute of Technology

B. Thomas Golisano College
of
Computing and Information Sciences

Department of
Networking, Security, and Systems Administration

Master of Science in
Networking and System Administration

May 13th, 2013

Committee:
Charles Border (Chair)
Yin Pan
Matt Lidestri

Abstract

E-commerce applications have flourished on the Internet because of their ability to perform secure transactions in which the identities of the two parties could be verified and the communications between them encrypted. The Transport Layer Security (TLS) protocol is implemented to make secure transactions possible by creating a secure tunnel between the user's browser and the server with the help of Certificate Authorities (CAs). CAs are a third party that can be trusted by both the user's browser and the server and are responsible for establishing secured communication between them. The major limitation of this model is the use of CAs as single points of trust that can introduce severe security breaches globally. In my thesis, I provide a high-level design for a new protocol in the application layer of the TCP/IP suite that will build a secure tunnel between the user's browser and the server without the involvement of any third party. My proposed protocol is called User-Defined Key Pair (UDKP), and its objective is to build a secure tunnel between the user's browser and the server using a public/private key pair generated for the user on the fly inside the user's browser based on the user credential information. This key pair will be used by the protocol instead of the server certificate as the starting point for creating the secure tunnel.

Contents

Introduction

The average person has now become more dependent on the Internet in their daily life, allowing a wealth of information to be derived from Internet traffic. This information motivates malicious users to develop new techniques to steal that information; therefore, Internet traffic requires a mechanism to prevent eavesdropping, tampering and spoofing attacks. The Transport Layer Security (TLS) protocol provides this type of Internet protection by creating an encrypted tunnel based on the public key infrastructure (PKI) technology. PKI allows two parties without a previous relationship to use Certificate Authorities (CAs) to share the security parameters required to create a secured tunnel between them [1]. A CA is the organization that issues and manages the security credentials and certificates required for verifying the identities of the involved parties. Browser vendors usually embed a copy of the root certificates for a set of CAs that they decide to trust, which helps the end user in deciding whether to trust a particular certificate. When a browser receives a website certificate that is signed by one of those CAs, it will automatically trust the website on behalf of the user. As one final check, browsers can use the information in both the website certificate and the certificate of the CA to verify that the certificate has not been revoked by the CA.

Although PKI has been widely deployed and is used in most secured websites, security professionals have expressed numerous concerns regarding the nature of CAs as a single point of trust. Specifically, if the CAs were compromised, severe security damages could occur and many users would be at risk. Incidents in the past have revealed multiple types of attacks upon CAs: those related to the failure of domain validation resulting in the issuance of domain certificates to individuals who are not the domain owner and those related to the malicious control of the computing resources maintained by CAs. As a result of these breaches, attackers were able to fraudulently acquire trusted certificates for different domains and services, allowing the attackers to transparently execute a man-in-the-middle (MITM) attack to view and manipulate the data being exchanged [2].

In my research, I will provide a high-level design for a new protocol in the application layer of the TCP/IP suite that will create a secured tunnel between the user's browser and the server. My protocol targets the issue of using the CA as a single point of trust. Instead of starting the process by receiving a certificate from the server that needs to be verified by the CA, I will start the process from the user, who will send the server a message signed with his private key generated on the fly from his credential information. The signed message only needs to be verified by the server with the corresponding public key, after which the session key will be generated by the server, encrypted with the user's public key and communicated to the user.

PKI Related Issues

Although PKI has been widely deployed and is used in most secured websites, security professionals have expressed many concerns regarding the nature of CAs as a single

point of trust. Various incidents that have occurred reveal different types of vulnerabilities in the PKI security model. Some of these incidents are related to a lack of domain validation, resulting in the issuance of domain certificates to individuals who are not the domain owner. Other incidents are caused by failing to validate the certificate, which results in the acceptance of a malicious certificate as a valid one. Still other incidents are related to the malicious control of the computing resources maintained by CAs. As a result of these breaches, attackers were able to fraudulently acquire trusted certificates for different domains and services, allowing them to transparently execute a man-in the-middle (MITM) attack to view and manipulate the data being exchanged [2].

Domain Validation Issues

Domain-validated certificates validate the ownership and control of the domain. They can be issued from different CAs, sometimes with long life times. If the ownership of the domain moves from one owner to other, it is possible that the previous owner has a domain-validated certificate for the domain from a different CA. This certificate could be used in conjunction with either a phishing attack or a DNS spoofing attack to masquerade as a legitimate site and bypass the protection afforded by TLS.

Certain certificates can function as wildcard certificates that are issued for *.x.com, in which case the certificate will be valid for me.x.com and you.x.com. A related issue with this feature is that the CA only verifies the ownership and control of the x.com domain, so a malicious user may issue a certificate for amazon.x.com and use it as a basis for phishing attacks on amazon users.

One of the techniques used to verify the ownership of a domain is the use of an email challenge-response mechanism to verify that the SSL certificate subscriber owns or controls the domain that he asks to include in the certificate. This verification is achieved by allowing the subscriber to select an address from a predetermined list including the admin, root, and administrator addresses. If this list is not carefully made—for example, if it includes email addresses that could be created for normal website users—other users could obtain a domain-validated certificate [12].

Certificate Validation Issues

There are also issues related to the validation of the certificate at the client side. One popularized issue is the null prefix attack that was publicly disclosed in 2009 and impacted a number of applications, such as popular web browsers, instant messaging applications, and email clients. In this attack, the attacker tricks the impacted application into validating his website certificate as if it were the certificate of the target website. For example, if we want to transparently run an MITM attack against paypal.com, we would need a valid certificate for “paypal.com”, which we cannot obtain because we do not own the paypal.com domain. However, we could obtain a valid certificate for the URL “paypal.com\0.attackersite.com” if we own the domain “attackersite.com”. Now the attacker, being MITM, will send the certificate that has been issued by a trusted CA to the browser, and the browser will compare the destination URL and Subject field of the

Certificate to authenticate the server. Vulnerable browsers will use the code in (figure 1) for comparison. The strcmp function will check every character of these values till it reaches the end of the string denoted by nocharacter "\0". In other words, the browser will consider "paypal.com\0.attackersite.com" equal to "paypal.com", and it will accordingly accept the attacker certificate as if it were a paypal.com certificate [13].

```
char *destination = getDomainWeAreConnectingTo();

char *commonName = getCommonNameFromCertificte();

bool everythingIsOk = (strcmp(destination, commonName) == 0);
```

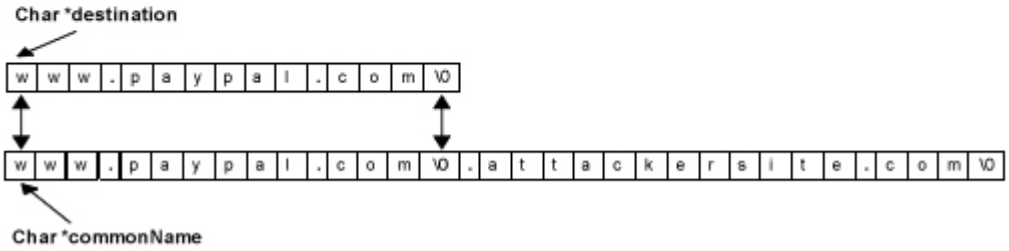

Figure 1: Vulnerable Certificate validation code "Source: http://palpapers.plynt.com/issues/2010Feb/null-prefix-attack/"

CA Computing Resources Attacks

Certificate authorities and registration authorities are the main source of trust in PKI; they are the groups that are responsible for investigating the legitimacy of and endorsing certificate requesters. This endorsement gives the requester full control over the traffic related to the domain given in the subject field of the certificate and thus the ability to capture sensitive data. Despite existing security measurements and security hardening, attackers were able to enter the CAs infrastructure and issue themselves valid TLS certificates for a series of famous domains that they did not control, such as Google, Yahoo, and Mozilla.

Attacks on DigiNotar and Comodo are well-known examples of the malicious control of CA computing resources to acquire valid certificates for high-value domains. In the case of Comodo, the hacker, calling himself "Ich Sun," stated that he broke into Comodo Italy using a very common database attack known as SQL injection, allowing him to execute commands on the backend database server that are supposed to be prohibited. He then took advantage of another flaw to obtain remote access to the system and eventually found a password hard-coded into a file on one of the systems that ultimately allowed him to issue the digital certificates [15].

User Reaction to Un-trusted Certificates

People are always considered to be the weakest point in the security system. Although PKI is publicly used, browsers still need to accept certificates that are not trusted and deliver a warning to the user to warn him of the risks associated with visiting websites with untrusted certificates. These warnings are common for many employees who use intranet services provided by their organizations that do not pay yearly fees to protect their intranet services, oftentimes because they believe that these services are already protected by other in-place security measurements such as firewalls, intrusion detection, intrusion prevention, and anti-viruses. When people become accustomed to ignoring warnings generated from untrusted certificates, they will be more likely to ignore warnings generated from real MITM attack.

Related Work

There has been an increased rate of security breaches against CAs, which has prompted browser vendors, CAs and independent groups to start working on innovative solutions and workarounds to mitigate the risks associated with the use of CAs. Significant efforts have been made in this area, which I will describe in this section of the proposal.

Convergence

A new model for authentication, known as Convergence, was released by Moxie Marlinspike in August 2011 and is based on previous work from Carnegie Mellon University called the Perspectives Project [3]. Rather than trusting a hard-coded list of CAs, Convergence allows users to configure a dynamic set of Notary Servers to validate the client connection. When a client receives an https certificate from a site, it will contact the Notaries and provide the host name, after which the Notaries will contact the site, receive its certificate and forward it back to the client. If the client receives a certificate that is different from everywhere else, it is likely that the certificate is fraudulent. Convergence provides a trust agility in which any browser could easily ship a default set of Notaries and then update them as needed without affecting the functionality, which cannot be performed with the current PKI implementation. For example, even if there is an issue with Comodo certificates, Comodo cannot be disabled because a quarter of the web would also be disabled; on the other hand, the Notaries can easily be replaced without affecting functionality [4]. However, Adam Langley, a Google security researcher, stated that he does not think Convergence will be added to Chrome because 99.99% of users would never change the default Convergence settings, which would lead to a huge amount of traffic to the default Notaries [5].

Certificate Patrol

Certificate Patrol is a plug-in implemented in Mozilla Firefox [6] that utilizes the fact that website certificates change infrequently, with normal lifetimes of three to five years. The key idea of Certificate Patrol is to build a dynamic mapping within the browser or in an external database that maps TLS certificates to websites. A warning alarm is then raised whenever the browser receives a different certificate for one of the websites in the map. This solution does not require any change to the PKI because it is a plug-in that monitors all SSL connections and determines if the certificate in question has been changed [6]. Although Certificate Patrol is good in concept, enterprise servers such as Google change the certificate frequently, which would confuse the user with frequent warning messages and cause the user to ignore those warnings.

Origin Bound Certificate (OBC)

Origin Bound Certificate (OBC) is a newly submitted RFC [7]; four researchers at Google presented a new extension to the TLS protocol by using a client certificate created on the fly by the browser without any user interaction. OBC does not include any information about the user and is sent to the server during the TLS handshake phase. The server binds the OBC to the authentication cookie later in the procedure. (See figure 2.)

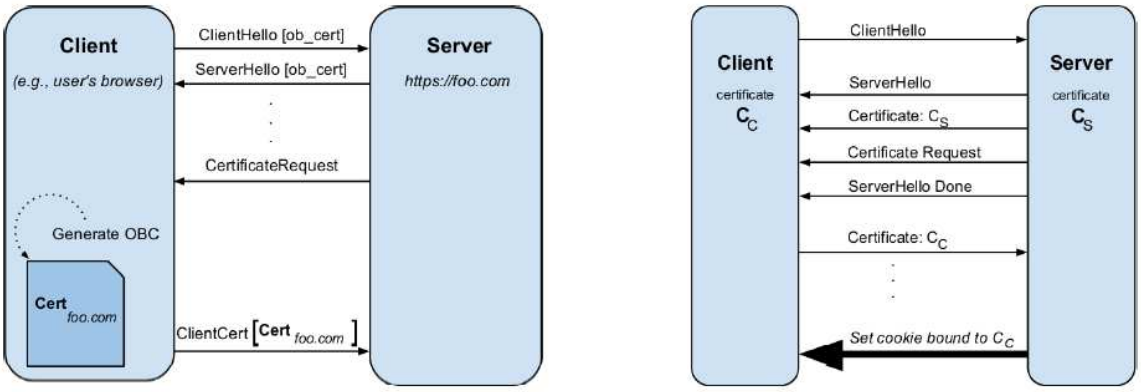

Figure 2: TLS-OBC handshake "Source:
https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final162.pdf**"**

On subsequent visits to the website, the browser will send the existing OBC to the server, which must verify that the public key in the certificate corresponds to the key used to authenticate the client in the handshake. Thus, the MITM will have to send a new OBC because he lacks the ability to forge the client's OBC, and the new OBC will be detected when the MITM forwards the user cookies to the server. (See figure 3.)

This approach cannot protect the first time the user accesses the website or the first visit after the browser cookies are cleared, either by the legitimate user or by the MITM, but

the browser can generate an alarm for the user at that time stating that the connection is not trusted.

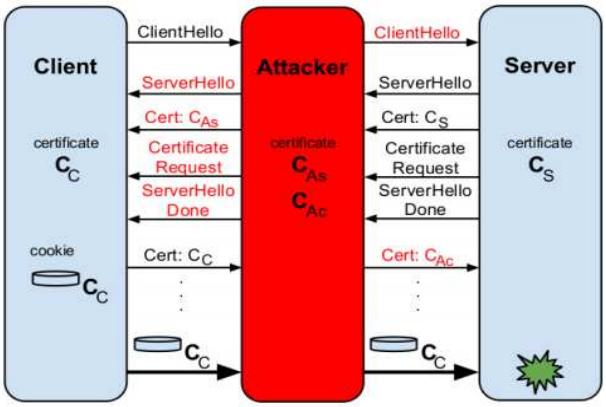

Figure 3: Using OBCs and bound cookies to protect against MITM. The server recognizes a mismatch between the OBC to which the cookie is bound and the certificate of the client (attacker) with whom it is communicating. "Source: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final162.pdf**"**

The next set of approaches uses DNS records to retrieve either the official certificate itself or the official Certificate Authority that issued the certificate for the website. DNS records are not digitally signed, which allows malicious users to easily manipulate DNS records and poison the DNS cache. Thus, these approaches need to use the DNS Security extension (DNS-SEC).

Certificate Policy Framework (CPF)

Certificate Policy Framework (CPF) was proposed in research published by Matthew Lidestri in January 2012 [8]. CPF offers a mechanism for the service operators to control which certificates are authorized to authenticate their services, with the help of a new DNS record (CPF). CPF gives the service operator the flexibility to define an access control list for each host name that declares which certificates should be passed and which certificates merit alerts to the user or a blocked connection. Thus, any CPF-compatible application can query DNS for CPF records to verify the integrity of the certificate.

These actions are represented by the following qualifiers:

(+) Pass: Permit the connection. This qualifier can be omitted because it is the default qualifier.

(-) Fail: Block the connection and do not offer the user with a means to override.

(˜) SoftFail: Warn the user but allow them to override the error at their discretion.

A sample domain zone file for hostname “example.com” is shown in figure 4.

```
www.example.com.        IN CPF      "v=1
hash_sha1:6a0e9a60583c365eedafad7f4010965515dc014a -
hash_sha1:42ac0d3e30198c893a1f301939ace903019355ec ~all"

mail.example.com.       IN CPF      "v=1
hash_sha1:938ca8e9a284355ce1a7ff7621c1d2d876ab2543 -all"
```

Figure 4: sample domain zone file for the domain example.com "Source: https://ritdml.rit.edu/bitstream/handle/1850/15220/MLidestriThesis2-28-2012.pdf?sequence=1"

In this example, the domain name www.example.com will accept certificates with an SHA-1 hash of 6a0e9a60583c365eedafad7f4010965515dc014a, fail certificates with an SHA-1 hash of 42ac0d3e30198c893a1f301939ace903019355ec and soft-fail any other certificate representing the domain.

Certification Authority Authorization (CAA)

One of the main issues with the current PKI implementation is that the compromise of any browsers’ trusted CAs or their children could be used to issue fake certificates to any https-protected site because browsers will blindly trust certificates signed by any of the trusted CAs. Certification Authority Authorization (CAA) DNS Resource Record is a new technique to reduce this risk by allowing the website operators to specify which public CA can issue certificates for their domains [9]. Although this will significantly reduce the risk of being dependent on the security of all browser-trusted CAs by depending only on the security of the authorized CA, the authorized CA can still be compromised, which would place all https websites authenticated by that authorized CA at risk.

DNS-based Authentication of Named Entities (DANE)

DNS-based Authentication of Named Entities (DANE) is a working group that is developing a protocol that would allow certificates to be bound to DNS names using Domain Name System Security Extensions (DNSSEC). DANE allows website operators to store a copy of the SSL certificate directly in the DNS record for their websites, so browsers can validate whether the SSL certificate presented to them is the same

certificate that the website operator provisioned on the operator's website. DANE is similar to the CAA solution because the security dependency is reduced to single entity, but it differs from CAA in that it relies on the signature of the parent domain. For example, the keys for "example.com" can only be signed by the keys for "com", and the keys for "com" can only be signed by the DNS root [10].

Although these DNS-based approaches prevent any untrustworthy signer from compromising anyone's keys except those in their own subdomains, the security of these approaches is based on the security of the DNSSEC, so the integrity of the DNS keys may be corrupted if the registrar for that domain is compromised. Thus, registrars still theoretically have the power to abuse their position because they are responsible for communication with the root servers.

Overview

The proposed protocol addresses the main TLS issue, the inclusion of a third party (CA) in the process of authenticating and securing the traffic between the user's browser and the server, by making the process the responsibility of the user and server.

The consequences of using a CA as a third party responsible for securing a user's communication with the server is that the client encrypts the required security parameters using a certificate received from some entity claiming to be the server. This claim is supported by the attestation of the CA, which may lead to many types of MITM. For example, another entity may manage to obtain a claim that it is the server and send a fake certificate to the user's browser instead of the real server certificate, thus taking advantage of the fact that browsers will automatically trust any certificate signed by a trusted CA or relying on the user to simply ignore browser warnings about untrusted certificates. As a result, the user's browser will create the secured tunnel based on this fake certificate, exposing all of the traffic to the fake certificate owner.

In the proposed protocol, when the user tries to access a website, the browser will ask the user to either go through a browser plug-in to be authenticated or to register the user credential information if it is the user's first visit to the website. Based on the credential information provided by the user, the browser will generate a public/private key pair for that user. Using that key pair, the secured tunnel is then created between the user's browser and the server.

In the next two sections, I will summarize the flow of messages between the user's browser and the server during the user registration and authentication.

User Registration

A user visiting the website for the first time will be asked to provide a username, password, password confirmation, and the answer to one of ten available security questions. This information will be provided to a browser plug-in and will be used to generate the public/private key pair for the user, and the browser will use the generated private key to sign the current timestamp of the server, which will act as a token. The generated public key, username, and token will be sent to the server, which will in turn validate the token using the received public key. The server will then generate the session key, encrypt it with the user public key that it received from the user's browser and send it back to the browser. If the browser has the corresponding private key, it will be able to read the session key and start communicating securely with the server. After the successful setup of the secured tunnel between the user's browser and the server, the user will be redirected to the website registration page to complete registration at the website. (See figure 5).

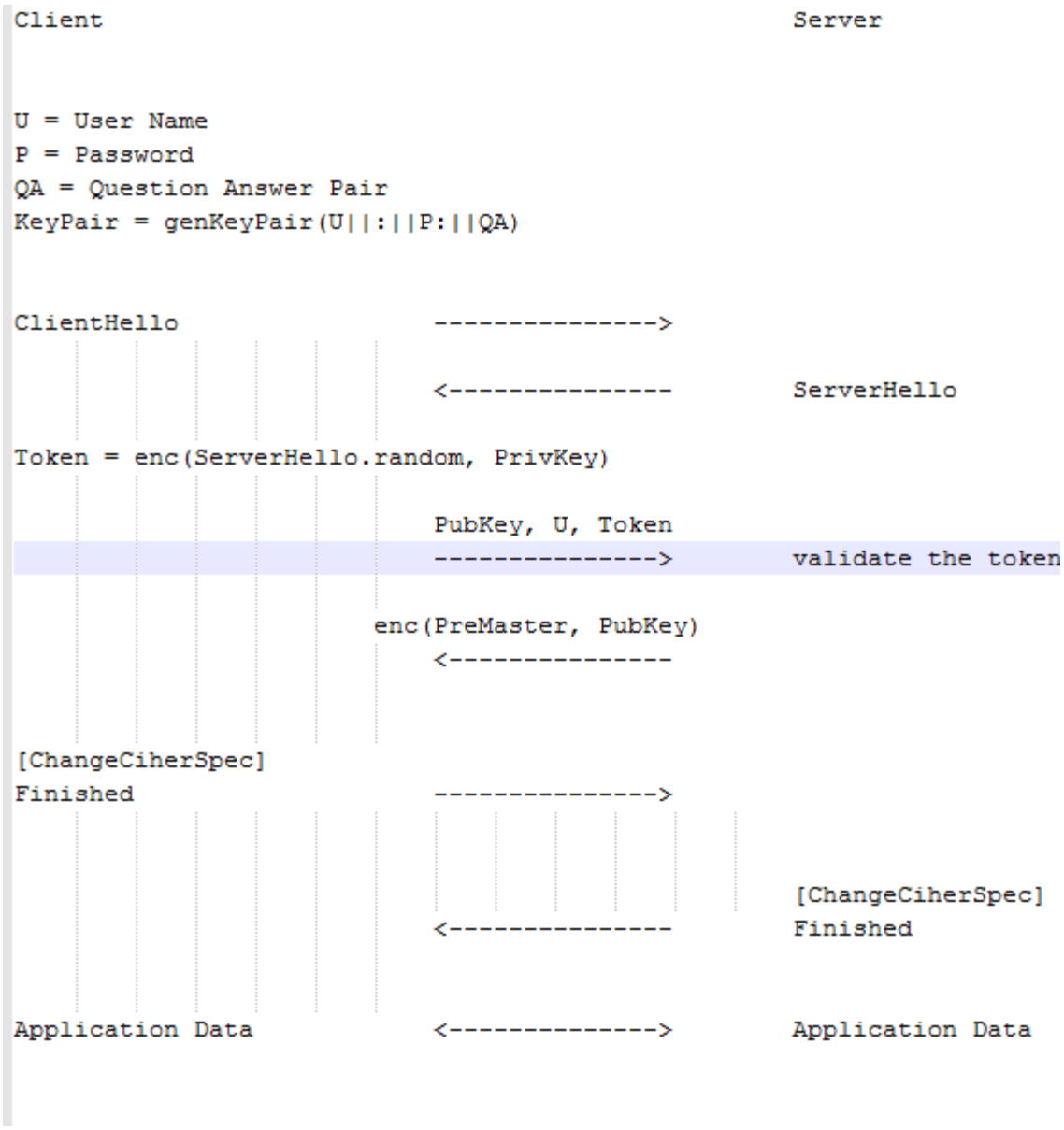

Figure 5: User Registration

User Authentication

When the user visits the website later, he must be authenticated through the browser plug-in, where the user will be asked to provide the username, password, and the security question/answer pair chosen during registration to generate the same key pair. The authentication will succeed only if the user provides the same information given during registration. Otherwise, an invalid public/private key pair will be generated inside the browser so that when the token, the signing of the current timestamp of the server with

the public/private key pair, is sent to the server with the username, the server will fail to validate the token using the registered public key and immediately terminate the connection with a fetal BAD_CERTIFICATE alert. (See figure 6).

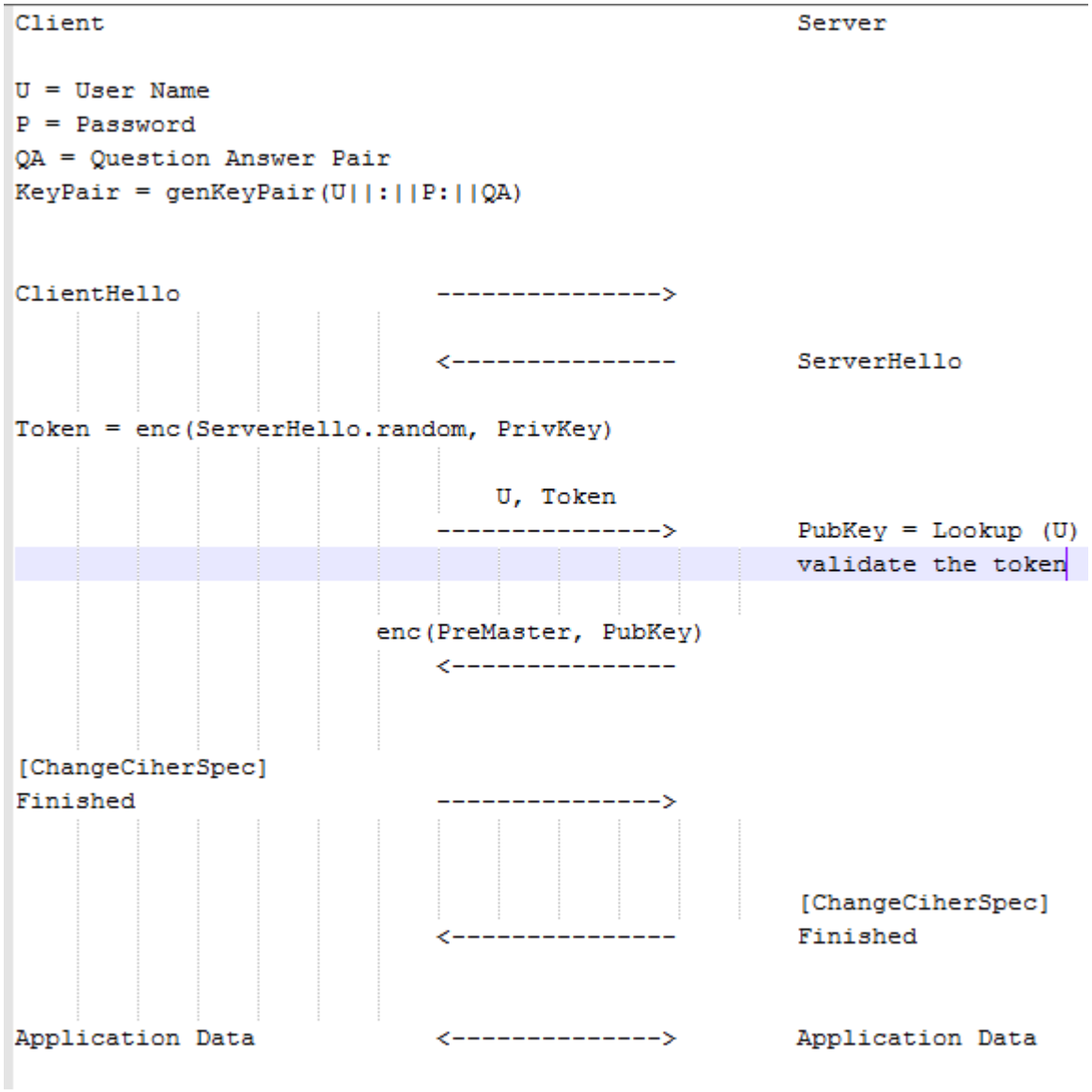

Figure 6: User Authentication

TLS-Based Implementation

TLS protocol is the current method used for PKI security and has been subjected to different types of attacks in the past. Both TLS and its predecessor SSL have gone through various security enhancements designed to reduce the risks inherited from the nature of the PKI, which makes it an attractive starting point for developing any new protocol that is related to the concept of creating a secured tunnel between two parties. Although UDKP is completely independent of TLS, an investigation of the technical details of the UDKP protocol reveals that UDKP is a modified version of the TLS protocol that takes advantage of the security practices implemented in TLS after years of practical experience. This similarity between TLS and UDKP will make the implementation of UDKP much easier because it uses a similar message structure as that in TLS protocol. UDKP uses fewer handshake messages than TLS, but the rest of the UDKP sub-protocols have comparable sub-protocols in TLS, including SSL Change Cipher Spec Protocol, The SSL Alert Protocol, and SSL Application Data Protocol.

In this section, we will give an overview of the TLS protocol details before describing the technical details of the UDKP protocol. TLS is usually implemented on top of Transport Layer protocols. It has been historically used primarily with Transmission Control Protocol (TCP); however, it has also been implemented with datagram-oriented transport protocols, such as the User Datagram Protocol (UDP) and the Datagram Congestion Control Protocol (DCCP). The TLS protocol is placed between the transport and application layers (see figure 7), and it has two sub-layers. The lower layer is based on either TCP or UDP protocol and essentially comprises the SSL Record Protocol that is used for the encapsulation of higher-layer protocol data. The higher layer consists of the following sub-protocols [14]:

SSL Handshake Protocol: responsible for negotiating the security parameters needed to establish the secured tunnel between the client and the server

SSL Change Cipher Spec Protocol: allows the communication parties to put into production and begin using the negotiated parameters.

The SSL Alert Protocol: used to exchange messages between the communication parties to indicate any potential problems.

SSL Application Data Protocol: takes application layer data and feeds it into the SSL Record Protocol for cryptographic protection and secure transmission.

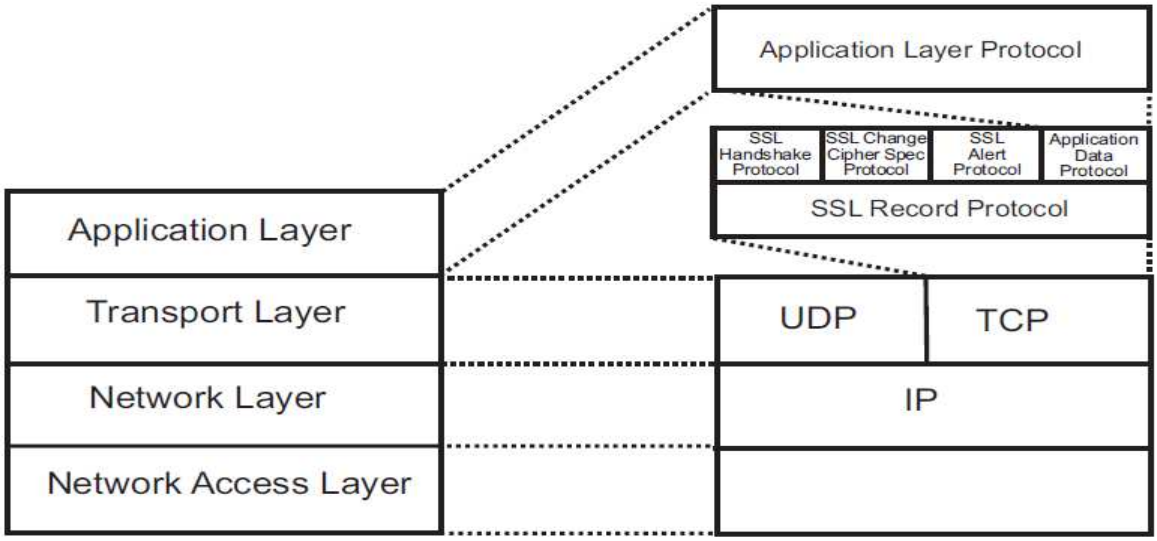

Figure 7: TLS Logical placement "Source: Rolf Oppliger, Ph.D. (2009). Ssl and Tls: Theory and Practice, Artech House, Boston, London"

TLS is a layered protocol that is used to create a secured tunnel between the client and the server to provide data privacy and data integrity; these goals are achieved through a series of messages exchanged between the client and the server to authenticate each other and to exchange the required keys for the security operations (see figure 8).

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
```

* Indicates optional or situation-dependent messages that are not always sent.

Figure 8: Message flow for a full handshake "Source: http://www.ietf.org/rfc/rfc5246.txt **"**

The message flow starts from the client, who sends a ClientHello message and waits for a ServerHello response message from the server. These two messages help the client and the server agree on the Protocol Version, Session ID, Cipher Suite, and Compression Method. After the ServerHello message is sent, the server sends its certificate followed by a ServerKeyExchange message if they are required according to the ServerHello message (e.g., the server might have no certificate, or it might have a certificate only for signing and so needs a ServerKeyExchange message for encryption). At that point, the server has sent all the data required from its side to establish the secured tunnel. The server might now request a certificate from the client for client authentication, or it will send the ServerHelloDone message. The server will then wait for a client response, which will be the client certificate if the server has requested it. If the server did not request a client certificate, the client will send the ClientKeyExchange message that may contain either nothing or the PreMasterSecret (Once again, this depends on the selected cipher in the ServerHello message). The PreMasterSecret is encrypted using the server public key and communicated to the server. If the client has sent a certificate with signing ability, it will send a digitally signed CertificateVerify message to prove his possession of the private key in the certificate.

Finally, a ChangeCipherSpec message is sent by the client followed by a Finished message, indicating that further communication will be protected using the new algorithms, keys, and secrets exchanged in the handshake. In response, the server will send its own ChangeCipherSpec message followed by a Finished message for the same goal. At this point, the handshake is over, and the client and the server may begin to exchange application layer data [11].

UDKP protocol differs from TLS protocol in the manner that it performs the handshake; specifically, it uses a reduced number of handshake messages. However, the remainder of the UDKP sub-protocols has parallels to TLS sub-protocols, including SSL Change Cipher Spec Protocol, The SSL Alert Protocol, SSL Application Data Protocol, and SSL Record Protocol.

Technical Details

The purpose of UDKP protocol is to provide a secured tunnel between the user's browser and the server using a set of messages. As I mentioned before, the messages exchanged between the user's browser and the server will be similar to the messages used in TLS 1.2 protocol to simplify the implementation of UDKP and to benefit from the best practices and countermeasures that are utilized in that popular protocol.

Public Key Locator Handler

Although UDKP is similar to TLS in the structure of the messages, changes I made in the handshake protocol prevent its implementation between the transport layer and the application layer. UDKP protocol now needs to interact with the application layer to

locate the public key for a specific user. UDKP is not only responsible for encapsulating packets coming from the application layer, which will move UDKP up to the application layer, it must also interact with the authentication server to authenticate the user. The server will receive the username from the browser in the authentication message, and depending on the implementation decision, the received username will go from the transport layer to the application layer to be used to locate the related public key. In my Proof-Of-Concept (POC) implementation, I use a pre-configured public key locator handler that knows how to locate the public key for a specific user, and I pass this handler to the transport layer. In the POC, I have defined only one type of handler that knows how to locate the user public key from any type of relational database, but in a real implementation, there might be various types of handlers that are able to locate the user public key from different types of storage (see figure 9).

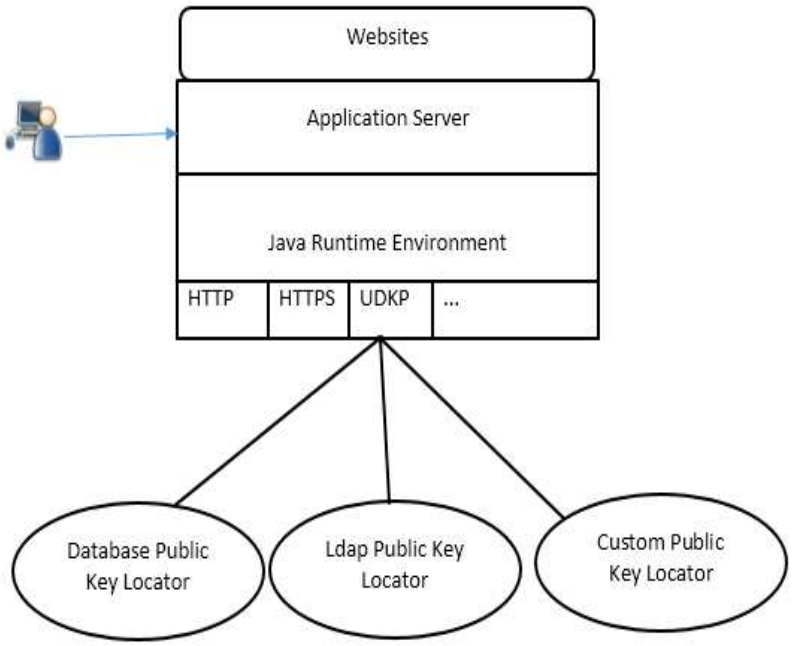

Figure 9: UDKP public key locator handler

For the UDKP to be able to complete the handshake phase, the handler needs to be configured in the server configuration file and the website deployment engineer needs to both specify the handler type, which will be a database handler in our case, and provide the required information to locate the user public key from the database (see figure 10).

```
<UDKP className = "org.apache.catalina.realm.UDKPRealm"
      driverName = "com.mysql.jdbc.Driver"
      connectionUrl="jdbc:mysql://DB_HOST:PORT/DB_NAME?user=&password="
      userTable="users"
      userNameCol="username"
      userCredCol="udc_key_col"
      registerUrl="http://localhost:8443/THE_INSERT_PAGE_NAME">
</UDKP>
```

Figure 10: UDKP database public key locator handler configuration

Database handlers require the following information:

className: points to the implementation of the handler, which is a database locator handler in this case. The development team has the option to define their custom locator handler implementation, or they can use the default implementation provided by the server.

driverName: indicates the database vendor that is used by the webserver to store the user information; the database is mysql in this case.

connectionURL: contains the location of the database and the credential information needed to connect to the database.

userTable: the table that contains the user information.

userNameCol: the column that contains the username.

userCredCol: the column that contains the public key for specific user.

registerURL: contains the URL to which the user will be redirected to complete the registration into the website.

The complete scenario is given as follows: when the webserver starts, it creates the public key locator handler based on the server configuration and then waits for incoming connections on the UDKP port. When it receives a connection from a client, it will pass the public key locator handler to UDKP. Using this handler, UDKP will know how to authenticate the user when it receives the Authentication message from the user's browser.

Message Structure

The message flow begins when the user uses a UDKP-supported browser to access a website that is protected with UDKP protocol on the UDKP-specific port. Before initiating the message flow, the browser must ask the user to provide his credential information to the UDKP plug-in (see figure 11).

Figure 11: Information message asking the user to access the website using the plug-in, and in the lower part how to get access to UDKP plug-in

The UDKP plug-in has two modes of operation: the registration mode, for new users, and the login mode, for existing users. In the registration mode, a new user will provide a username, password, password confirmation, and select a security question to answer. The general rule of thumb is that the password will never leave the user browser; instead, the provided information will be used by the browser plug-in to generate a public/private key pair (see figure 12).

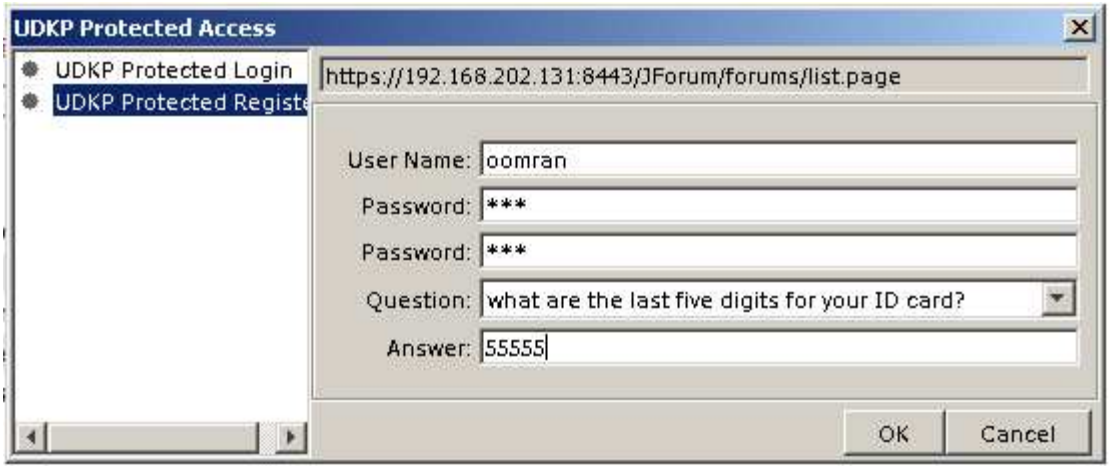

Figure 12: UDKP plug-in Register Mode

If the user is already registered in the website, he will go through the plug-in login mode to provide his username and password, select the same security question that he selected during registration and provide the same answer. This information will be used to generate the same public/private key pair that was generated during registration (see figure 13).

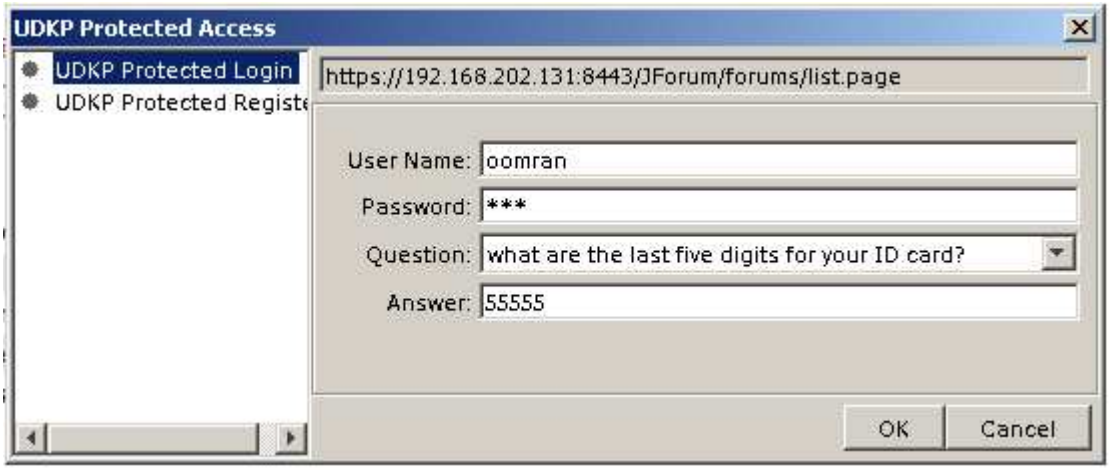

Figure 13: UDKP plug-in Login Mode

The browser is now ready to initiate communication with the server by sending the first handshake message, the ClientHello message, after which it waits for the ServerHello message response. Both of these messages are identical to their peers in TLS protocol.

Unlike TLS, UDKP will not send the server certificate or the ServerKeyExchange message, as the security of the tunnel is no longer based on the server certificate. Additionally, the client authentication will be through a message signed with the user

private key that was generated inside the browser plug-in, so there is no need for the server to ask for or receive the client certificate.

When the browser receives the ServerHello message, it will respond with the UserAuthentication message. This is a new message that has been introduced in UDKP and contains four fields: user_token, user_name, user_public_key, and auth_mode. The user_name is the username for the user that is logging in. The user_token is constructed by signing the ServerHello.random value, which contains the server timestamp; this will protect against replay attacks. The user_public_key is the public key generated for the user inside the user's browser based on the user credential information. The final field is auth_mode, which will contain a zero if the user is using the plug-in registration mode and a one if the user is using the plug-in authentication mode. The user public key is mandatory only during user registration – i.e., if auth_mode contains a zero – because during authentication the public key would have been already stored into the server

```
struct{
        UDKP_private-key-signed ServerHello.random user_token;
        UserName user_name;
        UserPublicKey user_public_key;
       AuthenticationMode auth_mode;
  } UserAuthentication;
```

Once the server receives the UserAuthentication message, it will check if auth_mode is zero, indicating a registration attempt. In that case, the public key must be in the message, or the connection will be terminated with a fetal no_certificate_RESERVED alert. If user_public_key exists, the server will use it to recover its ServerHello.random value from user_token using the user public key. If auth_mode was one, indicating that the user is trying to be authenticated, the server will assume that the user is already registered and will accordingly try to locate the user_public_key using the pre-configured public key locator handler, which knows how to locate the user_public_key for the incoming username. If user_public_key was found, the server will use it to recover its ServerHello.random value from user_token using user_public_key.

If the server failed to recover its ServerHello.random value from user_token using the user_public_key because of a missing or invalid user_public_key, the connection must be terminated with a fatal BAD_CERTIFICATE alert. If the value is recovered, the server will proceed to the next step by generating the pre-master secret that will be used to generate all the required session keys that would be used during the session. The method used to generate session keys is identical to that in TLS protocol. The PreMasterSecret message is also identical to its peer in TLS protocol and is communicated to the user after being encrypted with his public key so that no one can read it except the user himself, who has the related private key.

```
 struct {
     UDKP_public-key-encrypted PreMasterSecret pre_master_secret;
  } server_to_client_key_exchange;
```

When the browser receives the server_to_client_key_exchange message that contains the encrypted PreMasterSecret, it will recover the PreMasterSecret using the user private key. At this point, it will derive session keys from the PreMasterSecret, as specified in the TLS protocol, and then send a ChangeCipherSpec message to notify the server that all subsequent communications will be protected using the newly negotiated algorithms and keys. Finally, the client sends the Finished message, which is the first message protected with the new algorithms and keys. The Finished message contains a hash for all the handshake messages received from the server so that the server can verify that the security parameters the client is using are the same as those they both agreed to use and demonstrate no manipulation. When the server receives the Finished message, it responds with its ChangeCipherSpec message, indicating that it will move to the protected mode of communication. Lastly, the server sends its protected Finished message. The ChangeCipherSpec and Finished messages are identical to their peers in the TLS protocol.

At this point, the handshake is complete and both parties can communicate with each other with confidence that their communication is authenticated and protected.

Implementation Considerations

There are some considerations that must be taken into consideration during the implementation of UDKP to ensure the best possible security.

Firstly, the username that the user provides during registration must be the same as the username in the registered profile because the key pair generated inside the browser, which is sent to the server, is based on the profile username. If the user has the ability to change his username from the website, the public key that was sent from the browser to the server would be invalid because it would be based on the old username. Additionally, the public key cannot be regenerated by the server because the other components that are required to generate the key pair must not leave the user's browser. My POC implements the URL for the website's registration page as part of the server configuration; then, once the secured tunnel is created, the server redirects new users to the registration page, where the page reads the username from the incoming request and displays it to the user as a read-only field.

The second important consideration is that the username used in the plug-in login mode must be the same username used to authenticate the user to the website. Therefore, once the user is authenticated through the browser plug-in, he must be redirected to his home page and not to a normal login page. If a user were redirected to a normal login page, a MITM attacker would be able to set up a phishing attack against the user to steal his credential information.

Evaluation

In this section, various scenarios and use cases are studied and discussed to evaluate and ensure the robustness of the protocol against various types of attacks. Our security evaluation is based on a predefined threat model with a fairly broad scope that I believe to be a real-world threat model. The other part of the evaluation uses the proof-of-concept implementation to evaluate the performance of UDKP in comparison to TLS protocol.

Threat Model

In my threat model I will focus on the following types of attacks:

Registration Attack: an MITM attacker is able to insert himself as an MITM when the user is registering his profile for the first time in the website.

Login Attack without Public Key: an MITM attacker is able to insert himself as an MITM during the login operation without previous knowledge of the user public key.

Login Attack with Public Key: an MITM attacker is able to insert himself as an MITM during the login operation and knows the user public key in advance; this knowledge may be from a database attack or from the registration phase, where the user public key is transferred in the clear.

Phishing Attack: the attacker manages to install his phishing website as a proxy between the user and the real website in the hopes of stealing valuable information from the user, and that attacker has previous knowledge of the user public key.

Registration Attack

This attack is related to the user registration phase. For the purpose of simplification, I will consider the facebook.com website as an example, and I will assume that facebook is protected using the UDKP protocol. To register in facebook, I will go through the browser plug-in and provide my username, password, and the security question/answer pair, after which I will connect to the facebook. At that time, the public/private key pair will be generated inside the browser based on the provided information and the following message will be sent from the browser to the facebook server for the sake of registration:

```
struct{
        UDKP_private-key-signed Random user_token;
        UserName user_name;
        UserPublicKey user_public_key;
       AuthenticationMode auth_mode;
  } UserAuthentication;
```

If the MITM attacker delivers this message to the server, the server will respond with the session key encrypted by my public key. Because my private key did not leave the

browser, the MITM attacker does not have it and thus will not be able to read the session key information or monitor the traffic (see Figure 14).

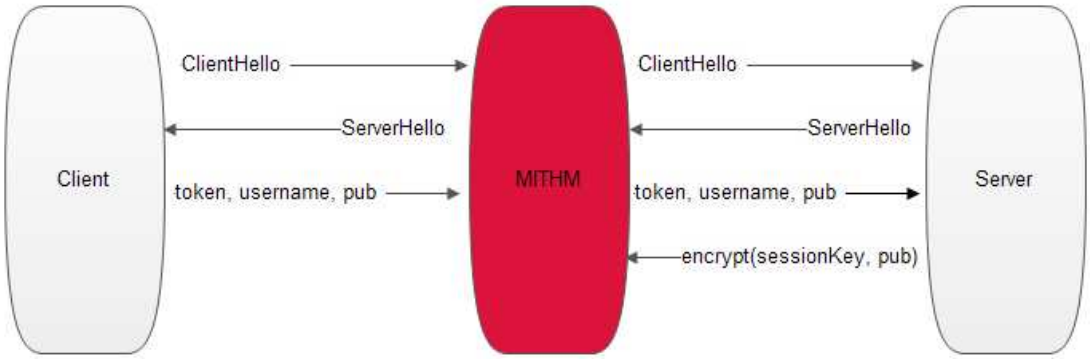

Figure 14: MITM attacker trying to perform a passive attack during user registration

The only chance for the attacker is to generate a different key pair and to send the newly generated public key to the facebook server in place of mine. Then, when the facebook server sends the session key premaster to me, it will be encrypted using the MITM attacker's public key. This allows the attacker to be able to read the session key premaster and use my public key to re-encrypt the session key premaster and send it back to me. At that time, the traffic will be exposed to the MITM attacker. However, the attacker is the one who has registered at facebook and not me, so I will be using the attacker's account. Once I try to access facebook from a clean connection, I will not be able to access it because the public key stored in the facebook is not my key, but the attacker's key (see Figure 15). This type of attack could be mitigated by first opening a server-authenticated connection, then renegotiate a UDKP-authenticated connection with the handshake protected by the first connection.

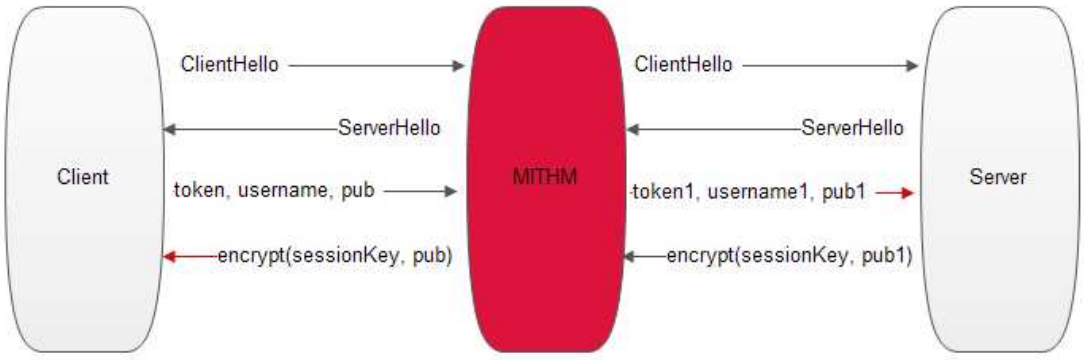

Figure 15: MITM attacker trying to perform an active attack during user registration by sending his information instead of the victim information

Login Attack without Public Key

As stated above, the public key is only mandatory during the registration; thus, if the user was able to securely register in the website, he would have avoided the threat with the most risk to his privacy. Let us consider again the facebook example. When the user provides the username, password, and security question/answer pair to the browser plug-in login mode and after the public/private key pair is generated by the plug-in for the user, the public key has been already stored in the facebook server. The server then needs to fetch the public key based on the received username to validate the user_token that contains the ServerHello.random value signed using the user private key.

```
struct{
        UDKP_private-key-signed Random user_token;
        UserName user_name;
       AuthenticationMode auth_mode;
  } UserAuthentication;
```

If the attacker delivered the UserAuthentication message to the facebook server, the server will try to read and validate the token using the user public key. If it succeeds, it will respond with the session key premaster encrypted by the user public key. Because the user private key did not leave the browser, the attacker will not be able to read the session key and hence will not be able to monitor the traffic (see Figure 16).

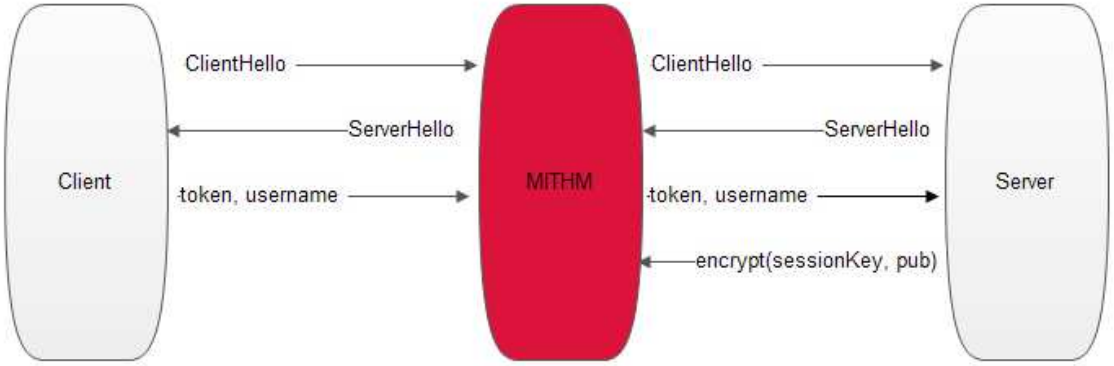

Figure 16: MITM attacker trying to perform a passive attack during the user Login

The attacker may only be successful by already being a registered user in facebook and to send his own Authentication message to the facebook server, which will then respond with the session key premaster encrypted with the attacker public key. However, the victim will be waiting for the session key premaster encrypted with his own public key, and because the MITM attacker does not know the user public key, he will not be able to communicate the session key premaster to the victim. Thus, the attack will be detected and the connection will be terminated (see Figure 17).

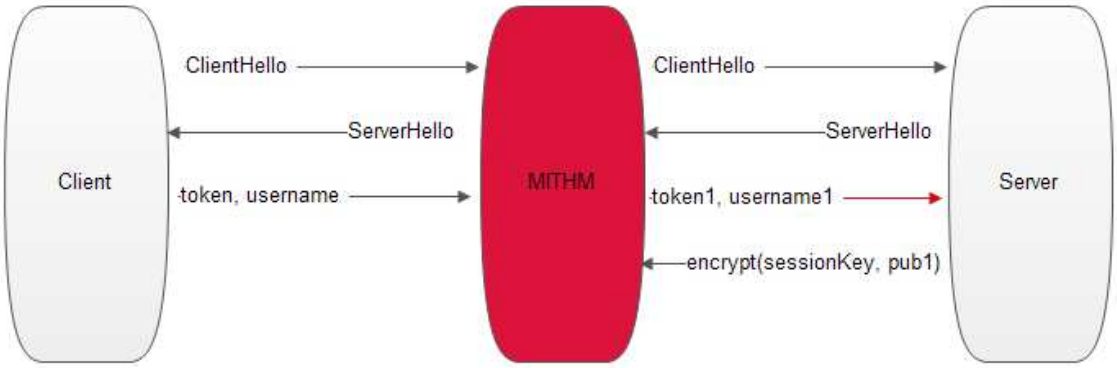

Figure 17: MITM attacker trying to perform an active attack during the user login by using his username instead of the victim username, where the attacker does not know the victim public key

Login Attack with Public Key

This class of attack is similar to the previous type of attack, but the MITM attacker knows the victim public key either through a database violation or from when the user registered in the website.

When the victim sends his Authentication message, the attacker does not pass it to the facebook server. Instead, the MITM attacker sends his own Authentication message to the facebook server, assuming that he is already a registered user at facebook, after which the server responds with the session key premaster encrypted with the attacker public key. The attacker then reads the session key premaster and re-encrypts it using the known victim public key. However, once the handshake is over, as mentioned in the implementation considerations, the username used in the plug-in login mode will be the same user used to login the user into his account home page. Thus, the victim will actually see the attacker facebook account, not the victim account, and he will be able to detect the anomaly (see Figure 18).

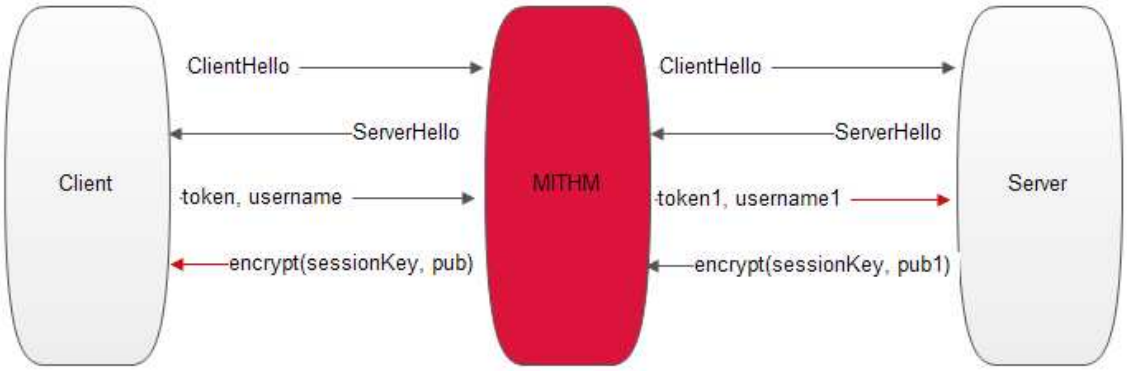

Figure 18: MITM attacker trying to perform an active attack during the user login by replacing his username instead of the victim username, where the attacker knows the victim public key

UDKP and Phishing Attacks

The main objective of phishing attacks is to steal the victim credential information (username, password, and security question/answer pair) so that the attacker can have access to the victim account later. In our case, this information will fortunately never leave the user's browser. Thus, phishing attacks with UDKP protocol will be for the sake of traffic monitoring and they will follow the same patterns discussed previously.

Proof of Concept (POC)

For the sake of both demonstrating the functionality and proving the concept behind UDKP protocol, I have made a simple implementation for the proposed protocol. In this implementation, I have modified TLS protocol to reflect the new changes in the handshake messages. I made the implementation to be as full-functioning as possible to reflect the accurate performance comparison results. My POC environment is a virtual machine that can work in four different modes: TLS Server, TLS Client, UDKP Server, and UDKP Client.

In TLS Server mode, the machine runs the original versions of Tomcat 7 application server, OpenJDK java runtime environment, and a forum web application.

In TLS Client mode, the machine runs the original version of Lobo web browser, and OpenJDK java runtime environment.

In UDKP Server mode, the machine runs the modified versions of Tomcat 7 application server, OpenJDK java runtime environment, and a forum web application.

In UDKP Client mode, the machine runs the modified version of Lobo web browser, and OpenJDK java runtime environment.

I then made two copies of this model virtual machine: one can work as either a TLS Server or a UDKP Server, and the other can work as either a TLS Client or a UDKP Client. This yields two identical environments to prove the concept behind UDKP protocol and to measure the performance comparison of the TLS vs. UDKP protocols.

I have not written any application from scratch for this POC. Instead, I have modified some of the existing open source applications. The following are the main components of the POC:

1. Tomcat 7 application server (open source written in java).

2. Lobo web browser (open source written in java).

3. OpenJDK Java Run Time Environment 6 (open source written in java).

4. JForum web application (open source forum written in java).

Java runtime environment already contains the TLS protocol implementation; it acts as the infrastructure upon which other java applications can operate. Lobo browser is the client that initiates the TLS connection with the Tomcat application server using the TLS implementation in the java runtime environment for preventing the interception or monitoring of the traffic to the forum application. To simplify the task of implementing a fully functioning UDKP protocol, I have modified the TLS implementation inside the java runtime environment and changed the method that the communication parties (Tomcat Application Server and Lobo Browser) use to communicate with TLS protocol. Finally, I needed to make certain changes to the forum web application to support the UDKP protocol; I will talk about these changes in detail in the next section.

I used the following procedure to evaluate the performance of the UDKP protocol in comparison with the TLS protocol:

1. Run the TLS environment (TLS Client and TLS server virtual machines).

2. Start the Tomcat in the TLS Server virtual machine and run the Lobo browser in the TLS Client virtual machine.

3. From Lobo browser in the TLS Client, request the forum application that is deployed in the TLS Server machine and write down the milliseconds required to handle that request (I modified the Lobo browser to print this value).

4. Issue the same request from the same browser window two more times and record the corresponding milliseconds (for subsequent requests, the session would be cached and the full handshake would not occur).

5. Repeat steps 1 to 4 three times and record the average number of milliseconds required to handle the first, second, and third requests to the forum application.

6. Repeat steps 1 to 5, but run the UDKP environment (UDKP Client and UDKP server virtual machines) instead of the TLS environment.

The results of the above procedure are summarized in the following table, which has two parts: the first part is related to the TLS protocol, and the second part is related to the UDKP protocol. I have three readings for the first request, which contains the full handshake, and I also have three readings for the subsequent requests where the session is retrieved from the cache.

Environment		First Request	Second Request	Third Request
TLS		7719	47	32
		2593	31	16
		2562	47	16
	Average	4291.33	41.66	21.33
UDKP		5812	31	15
		3141	47	47
		3141	31	47
	Average	4031.33	36.33	36.33

Table 1: UDKP and TLS performance comparison

As we can see in the above table, the average values for UDKP requests are less than the average values for TLS requests, but they are approximately the same. This is reasonable because, although the server no longer sends its certificate, it still needs to verify the user Authentication message through a database connection; thus, the cost is approximately the same. However, one of the factors that control these measures is the number of users in the database: if there are thousands of users inside the database, the authentication will definitely take more time to complete. If we take into consideration that the handshake of the UDKP not only secures the traffic to the server but also represents the authentication operation for normal websites, we can conclude that the UDKP handshake has a better performance than TLS.

Limitations

UDKP aims to completely replace the TLS protocol; however, the main limitation of UDKP in comparison to TLS is the protocol layer. TLS protocol works in the transport layer, meaning that TLS usage is transparent to the application. Thus, existing applications can migrate easily from http to https without any change to the application code. Unfortunately, this is not the case with UDKP, which works in the application layer. To migrate existing applications to UDKP, it would require changes to the application code. In this section, I will talk about the main limitations of UDKP.

Website Login Changes: according to the definition of UDKP, the application code must not perform the login because the user must not provide the password to the website. Once the user receives its website home page, he would have been already authenticated by UDKP; otherwise, if the authentication failed in the handshake, the server must terminate the connection with the user. If the user was successfully authenticated to the server through UDKP protocol, the application should receive the username in the request, and load the home page information for the user from the database using the username received in the request instead of using the username and the password (see Figures 19, 20).

```
Username = <get username from the login screen>
Password= <get password from the login screen>

SELECT <User Information>
FROM <User Table>
WHERE username = <Username> AND password = <Password>
```

Figure 19: The original code of JForum application (pseudocode). The user information is loaded based on the username and the password provided from the website login screen

```
Username = <get username from the protocol throw request>

SELECT <User Information>
FROM <User Table>
WHERE username = <Username>
```

Figure 20: JForum application code after modification (pseudocode). The user information is loaded based on the username coming from the server with the request

There is one more required change for the UDKP authentication to work, which was discussed previously. This change is the manner of telling the server where to locate the user public key to be able to authenticate the user during the UDKP handshake. In my POC, I have implemented this change through the concept of handlers, where the server comes with a set of predefined handlers to handle different types of storage. We can have database handlers to locate the public key in databases, LDAP handlers, or even custom handlers that the website development team can define. I have implemented a database handler to use in my POC; the deployment engineer needs to configure the server with this handler in the server configuration file (See figure 21).

```
<UDKP className = "org.apache.catalina.realm.UDKPRealm"
      driverName = "com.mysql.jdbc.Driver"
      connectionUrl="jdbc:mysql://DB_HOST:PORT/DB_NAME?user=&password="
      userTable="users"
      userNameCol="username"
      userCredCol="udc_key_col"
      registerUrl="http://localhost:8443/THE_INSERT_PAGE_NAME">
</UDKP>
```

Figure 21: UDKP Server Configuration

Website Registration Changes: The registerURL is recommended for redirecting the user to the registration page when a new user tries to register through the registration mode of the plug-in. For security reasons, the user must register with the same username used in the plug-in. This is because if the user changes the username in the website, the password and security question/answer pair must also be provided to the website to regenerate the public key. Providing the password and question/answer pair to the website is prohibited according to UDKP protocol, so the web application must receive the public key and the username from the browser plug-in and save them in the user database. In the registration

page, the password fields must be removed as they are not required and the username must be displayed as a read-only field (See figures 22, 23).

Figure 22: The original Forum registration page

Figure 23: The modified Forum registration page

Clear Username: The username is getting transferred in clear during UDKP handshake, as it will be used to fetch the user key that is required to create the secure tunnel. This risk could be mitigated by first opening a server-authenticated connection, then renegotiate a UDKP-authenticated connection with the handshake protected by the first connection.

User profile is required: For websites like search engines where the registration of the user profile is not required to be able to use the website, UDKP will not be able to protect the users' traffic to such websites.

Future Work

There are certain modifications that could be made to enhance UDKP security, performance, and functionality. In this section, I give a list of those enhancements.

Offline Brute Force Attack

UDKP is vulnerable to offline brute force attacks against the server_to_client_key_exchange message that contains the session key premaster encrypted with the user public key. This message is only readable using the user private key. Although an MITM attacker does not have the user private key, he knows that the private key has been generated based on the username, password, and security question/answer pair using a known formula. Because the username is known to the MITM as it is transferred in the clear, the MITM attacker can try all possible passwords with all possible answers to the ten security questions to regenerate the public/private key pair. The correct password would be the one that generates the correct key pair. Each generated key pair would need to be tested by trying to decrypt the token that was signed with the user private key. This is a very serious vulnerability and must be mitigated with one of the following measures:

1. Forcing passwords to have a minimum complexity to make this offline attack impractical. This complexity could be achieved by enforcing a minimum length of eight characters, with different cases and at least one number or special character. Additionally, the security questions must be chosen carefully so that each question may have thousands of possible answers.

2. Another mitigation that could be implemented is to complicate the process of generating the public/private key pair by making it time-consuming. This would make a brute force attack impractical. For example, if the process of generating a key pair consumed an additional second, it would not significantly affect the user login operation but would make a brute force attack by generating all possible key pairs impractical.

3. Finally, you can give the cautious users the option to replace the security question/answer pair with a randomly generated key pair and to store that key pair on a USB drive or smart card after encrypting the private key with the user's password. That will add a significant amount of protection against a brute force attack, and will achieve two factor authentication without modifying the application code.

Unified Secured Key Pair Generation

An important part of the protocol is to generate the public/private key pair from the username, the password, and the security question/answer pair. This generation must be very secure and unified across all protocol implementations so that all browsers can generate the same key pair from the given username and password. That means that all the implementations must use the same security questions defined by the protocol.

These are the set of limitations and enhancements that need to be considered for the real implementation of UDKP to achieve the best security and functionality expected from the protocol.

UDKP and SSL Termination

This is a new feature that allows the termination of secure traffic and its conversion to unencrypted form at the load balancer or dedicated devices. This feature has many benefits, mainly for increasing the site and web application performance by offloading the handshake and cryptography work to dedicated hardware. One additional benefit of the SSL termination feature is to centralize the related measurements and protection against the common SSL attacks in one place. This also allows the application firewalls to validate and check the incoming requests for application-level attacks such as SQL injection and cross-site scripting.

SSL termination occurs by uploading your SSL certificate to the SSL termination device or load balancer, at which point it terminates a user's SSL browsing directly. In UDKP protocol, there is no server certificate to be uploaded. Instead, the server will negotiate the encrypted connection based on the user public key that is stored in the database server. Unfortunately, the SSL termination devices are not involved in the user authentication, so they will not have access to the user public key.

The proposed protocol could be modified such that the public key could be sent to the server during the login authentication mode. The load balancer could use this public key to encrypt the session key premaster secret and send it to the user. If the user is able to read the session key premaster, he knows the related private key. However, the user could generate any key pair and use it to authenticate any username, so as a final check, the load balancer must send the username and the public key to the webserver to validate that they are related to each other (See figure 24).

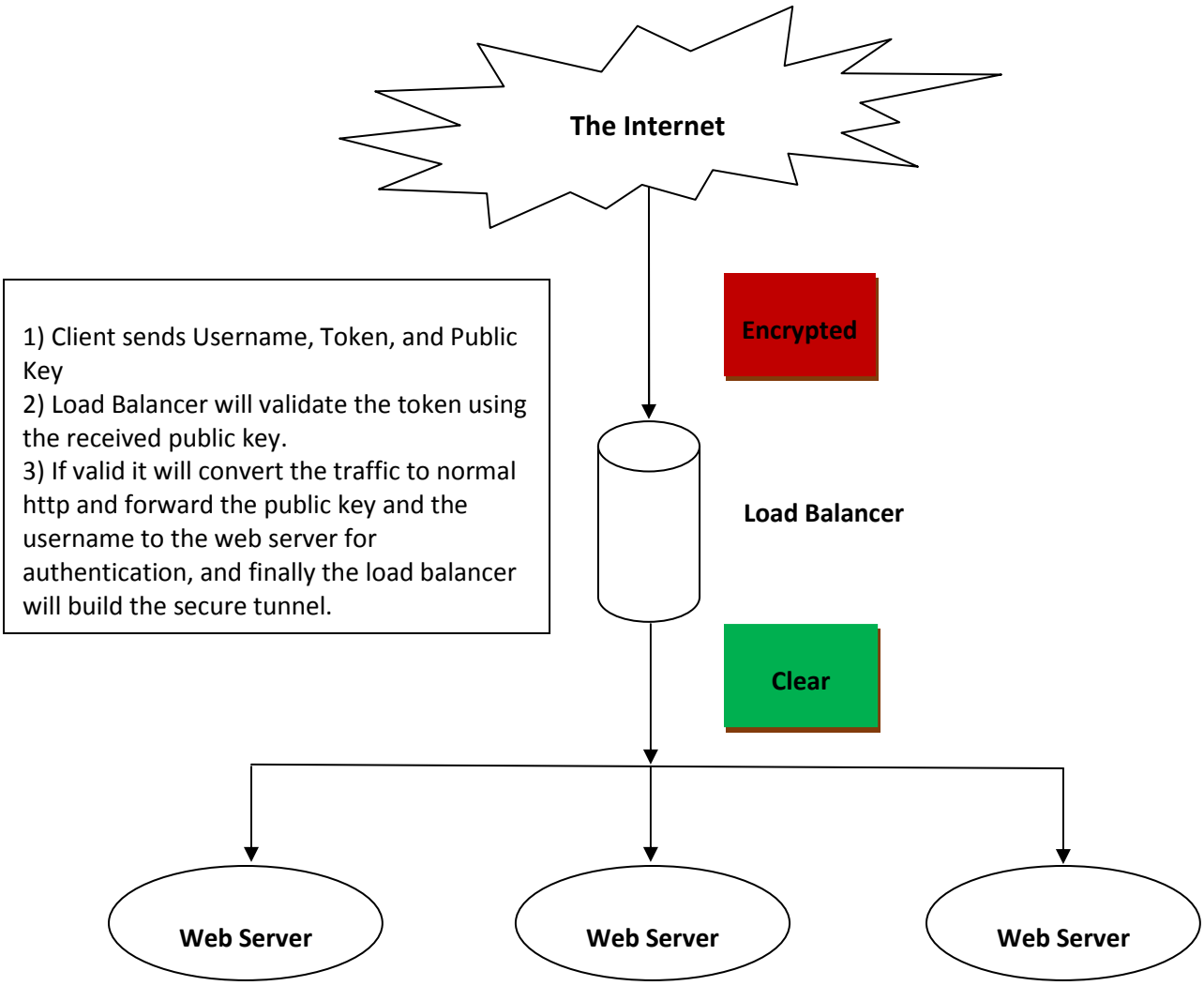

Figure 23: UDKP Termination

Password Change

According to UDKP, the password must not be provided to the website, it must be provided through the plug-in. However, we must consider the case if the user needs to change his password or he forgot his password. Then, the user must perform the related operations through the plug-in to be in compliance with the UDKP protocol.

In the case where the user wants to change his password, it is a simple process. The user provides the old credential information, which includes the username, the password and the security question/answer pair, and he will also need to provide the new credential information. Then, the encrypted tunnel is established using the old credential information, and the new public key is sent to the server to override the old key. The next time the user accesses the website, he will be required to provide the new credential information.

There are different methods to handle the case where the user forgot his password. One method is to simply send a token to the email, or some other websites send this token to a mobile phone that you had registered in advance with your account. Some online banks require you to call the bank and answer some security questions, after which the password will be reset to your visa pin code. The common behavior in all these options is that you will receive a token in some manner, and you will use this token to prove that you are the owner of your account and that you simply forgot your password. For UDKP protocol, that token could be used as the seed for a temporary key pair that would be used to prove that you are the legitimate owner of your website account, after which you can create a permanent key pair.

UDKP and OpenId

The OpenId technology makes your internet navigation simpler by having only one username and password to remember. The goal of OpenId protocol is to allow users to sign on to different services with a single identifier, where the authentication itself is performed by the OpenId provider. The OpenId provider will provide the user with an authentication URL that the user can use at any website that supports this technology. The supporting websites redirect the user to be authenticated to the OpenId provider on behalf of the website. When you visit a website that supports an OpenId login, look for a text box with an OpenId icon, type in your OpenId provider authentication URL, and you will be redirected to the OpenId provider to verify your identity using your OpenId provider credential.

One of the main security concerns related to the use of OpenId technology is phishing attacks that trick users into giving away their OpenId authentication credentials. The attacker does not have to attack the OpenId provider directly, but he can set up a malicious website that will redirect the user to a phishing OpenId provider URL under the control of the attacker. As a result, the user online identity could be impersonated. This risk could be reduced by using the UDKP protocol, where your identity would be proved to your OpenId provider using a message signed with your private key through the browser plug-in. Then, the OpenId provider, who is responsible for confirming your identity to other websites, only needs to keep your public key to validate your signature. Thus, there is no need for any password to be transferred across the internet, preventing any type of phishing attacks.

Conclusion

The use of Public Key Infrastructure that is provided by commercial CAs has protected the information that flows over the Internet from being compromised, and it is a key solution for e-commerce applications to protect their customers. However, this model is under increasing pressure to adapt to market realities, increasingly sophisticated users and higher expectations of security on the public Internet. The proposed solution aims to build traffic security without the need to be dependent on a third party to achieve this protection.

References

[1] Stephen Wilson, (Dec 2005). The importance of PKI today, *China Communications*. Retrieved December 1, 2012, from http://www.china-cic.org.cn/english/digital%20library/200512/3.pdf

[2] Zetter, K. (March 2011). Hack Obtains 9 Bogus Certificates for Prominent Websites; Traced to Iran. Wired.com. Retrieved December 1, 2012, from http://www.wired.com/threatlevel/2011/03/comodo-compromise/

[3] Dan Wendlandt David G. Andersen Adrian Perrig, Carnegie Mellon University. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. Retrieved December 1, 2012, from http://perspectivessecurity.files.wordpress.com/2011/07/perspectives_usenix08.pdf

[4] Marlinspike, M. (August 2011). SSL and the Future of Authenticity [Video file]. Retrieved December 1, 2012, from http://www.youtube.com/watch?v=Z7Wl2FW2TcA

[5] Adam Langley (September 2011). Why not Convergence? Retrieved December 1, 2012, from http://www.imperialviolet.org/2011/09/07/convergence.html

[6] Mozilla Firefox, Certificate Patrol add-ons. Retrieved December 1, 2012, from https://addons.mozilla.org/en-US/firefox/addon/certificate-patrol/

[7] D. Balfanz, Ed., D. Smetters, M. Upadhyay, A. Barth. Google Inc.(November 2011) TLS Origin-Bound Certificates. Retrieved December 1, 2012, from http://tools.ietf.org/html/draft-balfanz-tls-obc-01

[8] Matthew Lidestri (February 2012). Providing Public Key Certificate Authorization and Policy with DNS. Retrieved December 1, 2012, from https://ritdml.rit.edu/bitstream/handle/1850/15220/MLidestriThesis2-28-2012.pdf?sequence=1

[9] Hallam-Baker, P., Stradling, R. (August 2012). DNS Certification Authority Authorization (CAA) Resource Record. Retrieved December 1, 2012, from http://tools.ietf.org/html/draft-ietf-pkix-caa-13

[10] Hoffman, P., & Schlyter, J. (June 2012). The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol. Retrieved December 1, 2012, from http://tools.ietf.org/pdf/draft-ietf-dane-protocol-23.pdf

[11] T. Dierks Independent, E. Rescorla RTFM, Inc. (August 2008). The Transport Layer Security (TLS) Protocol Version 1.2. Retrieved January 29th, 2013, from **https://tools.ietf.org/html/rfc5246**

[12] Mozilla. CA:Problematic_Practices. Retrieved January 30th, 2013, from https://wiki.mozilla.org/CA:Problematic_Practices

[13] Palizine magazine. Defeat SSL Using Null Prefix Attack (February 2010). Retrieved January 30th, 2013, from http://palpapers.plynt.com/issues/2010Feb/null-prefix-attack/

[14] Rolf Oppliger, Ph.D. (2009). Ssl and Tls: Theory and Practice, Artech House, Boston, London.

[15] Robert McMillan, PC World magazine. Comodo hacker claims another certificate authority (March 2011). Retrieved January 31th, 2013, from http://www.pcworld.idg.com.au/article/381611/comodo_hacker_claims_another_certificate_authority/

[16] D. Taylor Independent, T. Wu Cisco, N. Mavrogiannopoulos, T. Perrin Independent (November 2007). Using the Secure Remote Password (SRP) Protocol for TLS Authentication. Retrieved March 6th, 2013, from https://tools.ietf.org/html/rfc5054