SIENAcollege
Computer Science

Computer Science 180
Web Design
Siena College
Fall 2011

Topic Notes: Web Security

Web security is the problem of ensuring that the data transmitted to or from web servers is legitimate and is protected from unauthorized access. It is a subset of the larger and more general problems of *computer security* and *network security*.

We have already seen one example: password protected content on a web server. We will now consider other aspects of this problem.

Detecting Bots

We just considered web search, and noticed that modern web search engines use robots/web crawlers ("bots") to retrieve copies of content from web servers periodically to build their index from which search results are obtained.

The idea that a computer can essentially "pretend" to be a human browsing using a web client is very convenient in this case. It would not be possible for modern search engines to be as thorough as they are without this.

We saw that even such legitimate use of a crawler is sometimes undesirable. Reputable search engine crawlers will honor a server's "robots.txt" file and will not crawl or index portions of a site that the server's administrators do not wish to be indexed.

Unfortunately, web robots can also be used for malicious purposes. It should be obvious that a bot can make requests from a server much more quickly than any human sitting behind a web browser. This idea can form the basis of a *denial of service attack* – where one, or more likely, many, bots all request pages from a web server as quickly as possible. The web server likely will not be able to keep up, and legitimate requests would not have a chance to be serviced.

Bots can also attempt to fill out forms for a variety of purposes such as systematic attempts to try to guess passwords, to register for accounts on sites like Google, Yahoo!, Facebook, Twitter, or blog sites, to vote repeatedly in online polls, or to attempt to gain an advantage in sites like eBay or Ticketmaster.

So it is an important problem to determine whether the entity interacting with a computer is a real person or a bot. To see more about this problem and a common solution, see the video linked below:

On the web: Luis von Ahn TEDxCMU talk including CAPTCHA and reCAPTCHA at
`http://www.youtube.com/watch?v=cQl6jUjFjp4`

HTTP Cookies

Cookies are a small piece of data (a text string) that can be stored on a computer's hard drive by a web page so it can later be retrieved.

The cookie mechanism was introduced early in the web's history as a way to remember information from page to page and across browser sessions. For example, if you had typed in a user name, a web site could store that in a cookie and already know your user name the next time you visit.

From our perspective, it is like having a JavaScript variable that will still have its value each time we visit the page.

Each cookie is stored as a name-value pair. All of the cookies associated with a site are sent from the browser to the server as part of a server request.

Many sites use cookies for a variety of purposes. We can see the cookies stored in our browsers and can guess the purpose of many of them.

There are many mechanisms that allow cookies to be set and retrieved. Since we have the most experience with JavaScript, we will look at how we can manage cookies with JavaScript.

See Example: simplecookie.html

The specifics are hidden in the functions `getCookie` and `setCookie`, which are borrowed from here.

We can see that the `document` object we know well has an attribute named `cookie` that contains our list of cookies. The `getCookie` function looks for a cookie in that list with the given name and returns its value, if found. The `setCookie` function takes a cookie name, a cookie value, and a number of days in the future after which the cookie should be removed from the browser.

The action in this example is in `visitCookie`, which is called when the page loads by the `onload` attribute of the `<body>` tag. It starts by attempting to retrieve a cookie named `lastVisit`. If it does not exist, we assume that this browser has never visited the site before and display an appropriate message. If there is a cookie, its value should be a date string set on the most recent visit. So that date is displayed. In either case, the cookie is set to contain the current date string, with an expiration date of 1 year into the future.

We can expand this to store more cookies with more information:

See Example: threecookies.html

Here, there are three cookies. One remembers the number of times someone has visited the site, one remembers the time of the first visit, and the third remembers the most recent visit.

Secure Web Servers

Recall that Internet traffic of all kinds is broken down into packets, and sent through a series of routers from source to destination.

Each router along the way and possibly other computers on the same networks as the source and destination will be able to see these packets of information. We must assume that any packet that gets sent onto the network will not only be readable by the destination computer but by pretty much

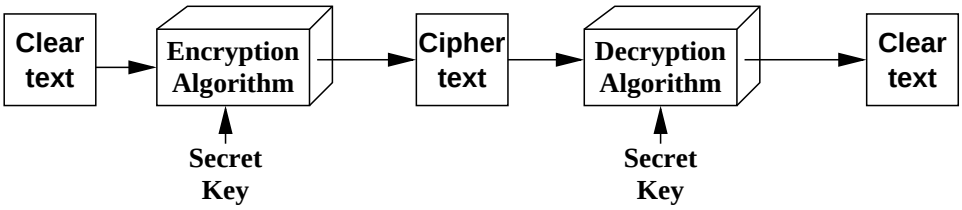
anyone. Tools such as *packet sniffers* can turn any computer into a packet spy on the network.

Combine this fact with the fact that we send information over the Internet all the time that we do not want made public, and it's clear that something must be done to keep our Internet traffic a bit more private.

Any data that should not be exposed to the public needs to be *encrypted*.

Encryption

Encryption allows some data, which we will call *clear text* to be modified into encrypted and seemingly unintelligible *cipher text*. This cipher text can then be sent across a network or stored on a device, and if anyone sees it, they will be unable to ascertain its meaning. Of course, to be useful, we will also need some way to convert the cipher text back to the original clear text (the process of *decryption*).



The cipher text is a function of the clear text, the encryption algorithm, and the secret key. The algorithm is public! Or at least a good scheme should not rely on the secrecy of the algorithm. It's just the key that is kept secret.

The clear text is a function of the cipher text, the decryption algorithm, and the same secret key. Again, the algorithm is public. The decryption returns the original clear text.

For the encryption to be strong enough, it must be very difficult to figure out the secret key, even given a bunch of cipher texts and the algorithm. Two approaches that an adversary may use are *cryptanalysis*, where properties of the clear text and the nature of the algorithm are examined to deduce the secret key, and a *brute-force attack*, where every possible key is tried until one works. For an n-bit key, this means up to 2^n keys must be tried, making brute-force attacks expensive. But modern hardware can break a 56-bit key in just a few hours.

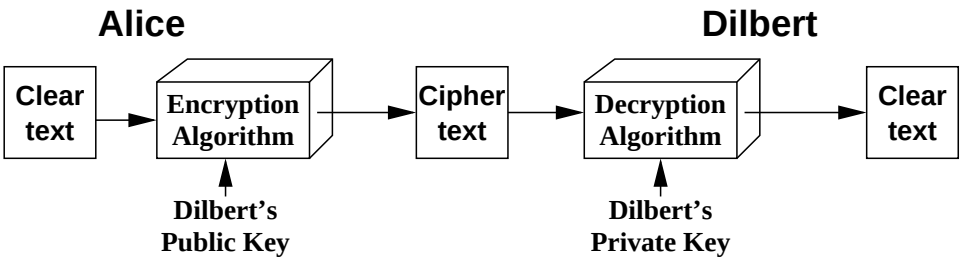
Examples:

- **The Data Encryption Standard (DES)** – 56-bit secret key. Selected by US Gov't in 1977. Broken in 1998.
- **Advanced Encrpytion Standard (AES)** – 168-bit key – competition was held, and **Rijndael** was selected in 2000 as the new standard.

Problem: how do we tell the intended recipient of our messages what our secret key is, without telling all the world what our secret key is? Perhaps this can be sent securely by some other means,

but perhaps the only communication channel is the one we do not trust that led us to employ encryption in the first place.

Public-Key Encryption

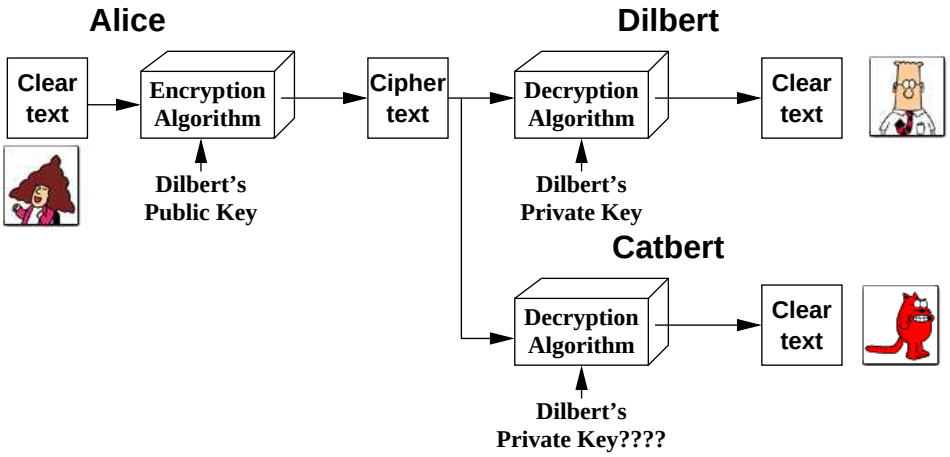


Instead of a single key, we have a *public key* and a *private key*. The public key is, well, public – anyone who wants to have it, can have it. But the private key is never shared. This idea was proposed in 1976 by Diffie and Hellman.

To transmit a message securely from Alice to Dilbert:

1. Alice and Dilbert each compute their public key-private key pair. Each publishes its public key for all the world to see.
2. Alice looks up Dilbert's public key, and uses it to encrypt the message.
3. Dilbert receives the message, and uses his private key to decrypt.

But what if Catbert intercepts the message?



Everything is just fine – even though Catbert, the evil director of Human Resources, has the cipher text and he can have Dilbert's public key, he does not have Dilbert's private key. So he has no way to decrypt the message.

There's still a potential problem with the distribution of public keys. Suppose Alice decides to send Dilbert a message for the first time, so she needs his public key. When she makes that request, maybe Dilbert is out of the office, but Catbert pretends to be Dilbert ("spoofs" his address) and sends his own public key instead. Since Alice didn't know it came from Catbert instead of Dilbert, she gladly encrypts messages intended for Dilbert using the bad public key, and Catbert sits in his office decrypting, soon to fire Alice for what she said about him.. This is known as a *man in the middle attack*.

We need a way for everyone involved to ensure that the others with whom they are communicating are who they say they are.

The mechanism most commonly used involves *digital signatures* or *certificates of authority*. A secure certificate authority (a popular one is called Verisign) gives a "certificate" to a computer that it can present to any other computer to prove that it is who it says it is. Sort of an electronic identication card.

When starting a connection with a server, we would request its certificate and compare to the certificate for that server from the certificate authority. If they match, we can be confident that we're exchanging information with the computer we think we are exchanging information with. Once we've done that, we can exchange public keys and continue on and have a safely encrypted conversation.

Your web browser does this all the time. If you are going to a web site where you are going to pass sensitive information across the network, you will likely connecting using `https`, the *secure http*, instead of standard `http`.

When you connect to a web server using `https`, your browser will make sure that the server is who it says it is by making sure the certificate matches one we have seen from that server in the past, or by checking with the certificate authority if we have never visited that server before. It can then use the *secure socket layer (SSL)* connection to communicate securely using a random encryption key (good for just the one session).

Note that modern browsers indicate that a connection is using `https` by putting a little padlock on the screen. Any time you are going to send information over the Internet that you would not want to be seen by everyone in the world, make sure it is going over an `https` encrypted connection!