Cryptography
Quick overview

This is a very quick overview of cryptography, as it applies to the Web

If you are taking a cryptography course, or know a lot about cryptography, please ignore these notes

If anything in these notes conflicts with anything real cryptographers say, believe the cryptographers
Perfection

The sequence the story takes is:

1. perfect encryption
2. why you can't always use perfect encryption
3. what is currently done instead
4. what happens on the Web
5. how quantum issues may change the future
Meet Alice and Bob

It is traditional in cryptography to use Alice and Bob as examples instead of A and B (any resemblance to real people is entirely coincidental!)

Alice and Bob want to communicate with encryption
The key concept in cryptography is the *key*

Let's say Alice and Bob share a secret key **BEAD**

Alice wants to send a message **FLEE** to Bob

Alice encrypts by 'adding' the key, Bob 'subtracts'

<table>
<thead>
<tr>
<th>FLEE</th>
<th>HQFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAD (+)</td>
<td>BEAD (-)</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>HQFI</td>
<td>FLEE</td>
</tr>
</tbody>
</table>

Addition is letter-at a time, with +A meaning add 1, +B add two, and so on
Modulo arithmetic

The arithmetic is one letter at a time, with no carries or carries.

If we take A=1, B=2 etc., then E+A means "E plus one" which is F.

F+B means "F plus two" which is H.

And we wrap round when we reach the end of the characters (Z wraps to A if there are only upper case letters).
Binary modulo arithmetic

In practice, the message and key are represented as bits, and binary modulo arithmetic is used.

It is not essential to switch to binary, but is more efficient, and \((-\)\) is the same as \((+\)\); they are both \((^\wedge\)\), i.e. 'exclusive or'.
Perfect encryption

If the key has these properties:

a. the key is truly random
b. the key is used once only
c. the key is a shared secret

Then the encryption is *perfect* and *cannot* be cracked

The technique is called *one-time pad* encryption (from the days of paper-based spy communication)

What do these three things mean in more detail?
Think of the key BEAD as the code which closes or opens a combination lock

It is clear that the best you can do to secure the lock is to *spin each wheel by a random amount*

So the key should be 'random'
In cryptography, 'random' means 'containing no pattern' and 'not guessable'.

If you use a pseudo-random number generator to produce lots of bits, an enemy can easily guess or find out what formula has been used.

So the only secret part is the starting point, the seed, say 32 or 64 bits, which is not enough.

So, hardware or entropy generators are used (in your own program, use a function from a crypto library).
b) using keys once

If you use a key twice, it is easy to crack: here it is used to encrypt two messages, FLEE and STOP:

<table>
<thead>
<tr>
<th>FLEE</th>
<th>STOP</th>
<th>HQFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAD (+)</td>
<td>BEAD (+)</td>
<td>UYPT (-)</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>HQFI</td>
<td>UYPT</td>
<td>MRPO</td>
</tr>
<tr>
<td>FLEE(-)STOP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The cracker subtracts the encrypted messages, the key cancels, and the result is two combined clear messages.

Unpicking the two messages just requires crossword-puzzle-type skills.
If you always avoid reusing key bits, then one key bit is needed for every bit of data which is encrypted.

Long messages, or lots of messages, or communication with lots of people, mean that lots of key bits are needed.

We will see later that generating shared secret bits from nothing is expensive.
Block ciphers

Block ciphers (also called symmetric encryption algorithms) improve efficiency

They are called symmetric (a) because the same key is used at both ends and (b) to distinguish them from asymmetric techniques, coming later

They have names like DES, Triple-DES, AES, IDEA, RC4, RC5, Blowfish, Twofish, Serpent

And modes like ECB, CBC, GCM, EAX, OCB
One way to think of a block cipher is that it is like a pseudo-random number generator.

It allows a few true random bits to be used as a 'seed' to generate a longer sequence of bits.

It is designed to be reasonably safe if used for a reasonable amount of time.
c) shared secrets

A big problem on the Web is: how can Alice and Bob, who have never met, have a shared secret?

It is a circular problem, because they need encryption for one of them to send the other a key.

This is called the key distribution problem.
Public key cryptography

The current solution is public key cryptography (also called asymmetric cryptography)

Each person/program generates a random pair of keys

A message encrypted with one must be decrypted with the other, one is kept private, and the other made public

Knowing the public one isn't enough to work out the private one, even though they are linked

Algorithms are called RSA, ECC (ECC is a speciality of the department's crypto research group)
Public key encryption

If Alice encrypts something using Bob's public key, Bob is the only one who can decrypt it.

Problem: the process is too inefficient for bulk encryption, so the system is only used to authenticate, negotiate, and transmit conventional block cipher keys.

Problem: there is still the issue of safely exchanging public keys to get started, i.e. authenticating the public keys.
Alice runs a web site //alice, and Bob visits it with his browser.

They send each other their public keys, then use them to negotiate a secure connection.

The authentication problem can be illustrated by the possibility of a man-in-the-middle attack.
Meet Chuck

The shadowy man in the middle is Chuck:

Chuck intercepts all traffic between Alice and Bob, and passes on every message (reading or changing it at will)

So how can Alice and Bob defeat Chuck's evil plan?
Public key cryptography can be used to solve the authentication problem.

Normally, Alice uses Bob's public key to send messages.

If Alice encrypts a message using her own private key, Bob (and anyone else) can decrypt it, but if decryption succeeds, it must have come from Alice.

The message is not secret, but it is effectively signed.
Authenticating Alice

It is often vital for Bob to know that he is genuinely talking to Alice.

So when Bob first gets sent a public key from Alice's site, his browser must be sure that it is genuinely Alice's public key, and that Chuck hasn't replaced it on the way.

This is where Faith comes in (make your own image!)

Faith is a well known and well trusted authority figure.
Suppose Alice sends Bob a message signed by Faith

The message says "Alice owns web site //alice and here is her public key"

This signed message is a certificate

Faith's public key is well-known, and built into Bob's browser

The message can't be faked by Chuck

So, Bob can be sure that the public key is OK, and after that messages from Bob can only be read by Alice
There is yet one more problem

How does Faith authenticate Alice, so that the certificate is valid?

In the past, Alice sent a request to Faith, and Faith manually did various background checks to make sure Alice was authentic

This was expensive and time consuming
Free certificates

More recently, the process of getting a certificate has been automated and made free.

The trick is for Faith's software to verify that Alice owns (or has admin privileges for) the //alice web site.

Faith sends Alice a unique ID (it doesn't have to be secret!)

Alice publishes it somewhere on the //alice site.

Then Faith visits and verifies it - it must have been published by Alice (somehow).
Let's Encrypt

The Let's Encrypt service (see Wikipedia) was the first automatic authority, taking on the role of Faith

How reliable is the service?

As with everything to do with security, this needs to be kept under continual review

But it is probably more reliable than anything involving people!
Authenticating Bob

Alice doesn't usually care whether she is talking to Bob or Chuck – a visitor is a visitor.

But if she does care, there are slow techniques available.

The commonest is to ask Bob for an email address, then send a secret to that email address.

If Bob then visits Alice's site and presents the secret, that 'proves' that Bob owns the email address (or it might if email was secure!)

It is rare for Alice to want anything better.
The block ciphers, public key algorithms, and other details used in HTTPS are not fixed, they evolve over time.

That's because techniques for cracking them slowly emerge, and the inventors have to stay ahead of the crackers.

So the algorithms actually used are negotiated between the two ends.
This means Alice has an extra duty

That is to avoid using obsolete algorithms where the security has lessened

That means upgrading the web site software frequently (at least yearly), and whenever specific problems arise for which upgrades are published

For example, Node and its SSL/TLS library used by HTTPS need to be upgraded regularly for Node-based servers (check at ssllabs)
So what are the things Alice has to do?

- make sure the site has no security loopholes
- create a public/private key pair
- make sure the private key stays very private
- use the public key to get a certificate
- upgrade the web server software frequently
Quantum computation is coming

A public/private key pair involves two large random prime numbers

The private key is related to the two numbers, and the public key is related to their product

Factorizing the product would break the security

In conventional computing, factorization is probably 'intermediate', but in quantum computing, factorization is 'easy'
Quantum cryptography

But quantum cryptography is also coming

It may become possible to detect any attempt to eavesdrop on or tamper with a message

So Alice could repeatedly send batches of random bits to Bob until one gets through untouched, and use that as a key

The hope is that practical quantum cryptography arrives before practical quantum computation (and that it won't be misused by criminals)