All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

Crypto Crash Course

XOR

Most of you are probably familiar with the XOR construct. This bitwise operator performs this operation: given two bits (denoted A and B), it outputs a 1 if either **A or B** are 1, but **not** both.

0 ^ 0 == 0
0 ^ 1 == 1
1 ^ 0 == 1
1 ^ 1 == 0

XOR has a property that makes it a massively important part of crypto: The ability to reserve itself.

Given D = A ^ B:
A == D ^ B
B == D ^ A

So we can produce a simple, absolutely perfect cryptography scheme: Generate a key of N bits of true random data. XOR each of the bits of the key with N of plain text. You get a perfectly encrypted cipher text blob of N bits. Decryption is just XORing against the same keystream.

ONE-TIME PAD

Some of you will recognize this as a one-time pad construct. If you use a given key only once – never repeating it or using the same key for multiple messages – it's absolutely perfect.
That's the 'one-time' part of it. But obviously, we don't want to have to give out massive pre-generated keys that can only be used once. And that's why we don't use OTP for day-to-day operations.

TRILLION DOLLAR QUESTION

Due to the fact that we don't really want to pass around massive keys to everyone we want to communicate securely with, the questions becomes:
How can we, with the highest assurance of safety and the least amount of time, share keys with all the relevant parties?
The entire field of cryptography exists to answer that question. Then we poke holes in the answers.

TYPES OF CIPHERS

We can break down the types of ciphers in use today into two families with their underlying sub types, all with their own purpose and flaws.

- Symmetric – Both sides share the same key
  - Stream – Encrypts data byte-by-byte
  - Block – Encrypts data block-by-block
- Asymmetric – Each side has their own private key and exchange public keys

STREAM CIPHERS

Stream ciphers encrypt byte-by-byte. The most common one you'll see is RC4, which is often used in SSL.
The basic construction is essentially a random number generator seeded with your key, which generates bytes that are XORed with each byte of plaintext for encryption. Decryption is simply XORing the cipher text instead. This means that both operations are identical.

Given this definition, we can construct a simpler stream cipher by choosing a given PRNG and seeding it with our key on both sides.
The strength of the algorithm is then solely dependent on the quality of the randomness. The better the quality of the random output, the stronger the encryption.

BLOCK CIPHERS

Block ciphers are perhaps more familiar to most people. AES (Rjindael), DES, 3DES, Twofish, and other common ciphers are all block ciphers.
In a block cipher, you split your data into N-byte blocks and encrypt those separately. Because we can't assume that all data is a multiple of N-bytes long, we have to pad data, introducing complexity. In addition, the encryption and decryption are not the same.

Because block ciphers encrypt block-by-block, many modes of operation exist to make them more secure and serve various goals. We'll talk about the most common two briefly.

EAC MODE

Electronic CodeBook mode is the simplest mode of operation for a block cipher. Each plain text block is encrypted independently to produce a cipher text block.
This means that if you see two blocks with the same cipher text, you know that they must have had the same plain text.

This may not seem like such a big deal, but take a look at what happens to an image of Tux when you encrypt him with ECB and a block size of 8 bytes (DES):

CBC MODE

Cipher-Block Chaining is perhaps the most common form you will see. With CBC, each plain text block is XORed with the cipher text of the previous block before encryption; the opposite is performed for decryption. The first block is XORed with the IV, or Initialization Vector.

Because of the fact that blocks are chained in this way, flipping one bit of cipher text in block 0 will flip the same bit (position) of plain text in block 1 upon decryption.
This property make for several interesting bugs that you'll see later.

ASYMMETRIC CIPHERS

With asymmetric ciphers, each party of the communication has a public and private key component. RSA is exemplary of this class of ciphers.
Asymmetric ciphers are used for both encryption and signing (a process that allows one party to validate the source of a message).

Generally, asymmetric ciphers are not used for encrypting data directly due to performance concerns and complexity. Rather, you use them to securely transmit a symmetric key.
If Alice want to send a secure message to Bob, she can encrypt the message with a symmetric key, encrypt that key with Bob's public key, and then send the cipher text and encrypted key to Bob. He then decrypts the key with his private key and uses that to decrypt the message.

HASHES

Hashes are construct that take in arbitrary blob of data and generate a fixed-size output, generally 128-512 bits. MD5, SHA1, SHA2 , and others are all hash functions.
Due to the fact that they take data of any size and produce a fixed-size output, all hash algorithms produce collisions – multiple inputs that produce the same output. The strength of a hash algorithm is in how hard it is to find such collision.

A hash on its own is really only useful for a single thing in cryptography: determining the integrity of data. If you are given a blob of data and its hash, it is trivial to determine if the data has been tampered with a transit.
It does not, however, ensure that the hash itself is what was intended, nor does it authenticate a sender in any way.

MACS

Message Authentication Codes are generally based on hashes, but allow for – as the name implies – message authentication.
What this means is that given a MAC, you can ensure that the data has not been tampered with, as well as validating that the MAC itself has not been manipulated.

This is because with a MAC, you have a shared key that is used for construction and validation of the MAC. Without it, you cannot create a valid MAC.

HMAC

The most well-know MAC is called HMAC. It is based around a hash algorithm of your choosing, and is a fairly simple construct.
HMAC(key,message) = hash(key + hash(key + message))
The keys are padded separately in each run of the hash algorithm, but this is a minor simplification.