# Encryption Decrypted

*Brendan Schlaman*

11/28/18

Alice wants to send an encrypted message to Bob that only he can decrypt. There are several ways of doing this:

# Option 1: Classic Encryption

This is the standard and intuitive way of encrypting data. Alice will use the following function to encrypt her message:

$$Alice.encrypt(key, message)\ \{XXX;\ return\ encrypted\_msg;\}$$

Alice sends *encrypted_msg* to Bob, who will then use the following function to decrypt the message:

$$Bob.decrypt(key, encrypted\_msg)\ \{YYY;\ return\ message;\}$$

This method works well enough, but there is a glaring flaw. **Alice and Bob both need to have the same key.** If they try to encrypt the key, the same problem arises recursively. This leaves 2 options.

1) Bob and Alice need to meet in person to agree upon a key.

OR

2) Alice needs to send the key (in addition to *encrypted_msg*) to Bob in a way that she feels is secure enough.

## NOTE: This is the method that BBY Custom Agents use.

The functions look slightly different, but the problem is the exact same.

$$BuildTeam.encrypt(Dev\_pub\_key, Build\_priv\_key, message)\ \{XXX;\ return\ encrypted\_msg;\}$$

$$Application.decrypt(Dev\_pub\_key, Build\_priv\_key, encrypted\_msg)\ \{YYY;\ return\ message;\}$$

Calling these keys public and private is somewhat of a misnomer: they may be public and private to the eyes of the Dev Team, but the combination of *Dev_pub_key* and *Build_priv_key* is the same as the *key* from the more general example above. Someone could intercept these and be able to decrypt the message.

# Option 2: Diffie-Hellman Encryption

How can Alice and Bob agree upon a key without exchanging any data other than the encrypted message? It seems impossible. However, some interesting and somewhat nontrivial mathematics makes it possible. The trick is called **Diffie-Hellman Encryption**. Here's how it works:

1) Alice and Bob agree on a public key
2) Alice and Bob each choose random private keys
3) Alice "mixes" her private key with the public key, and sends this *mix(A_priv, pub)* to Bob
4) Bob "mixes" his private key with the public key, and sends this *mix(B_priv, pub)* to Alice
5) Alice mixes her private key with the message from Bob

$$mix(A\_priv, mix(B\_priv, pub))$$

Bob mixes his private key with the message from Alice

$$mix(B\_priv, mix(A\_priv, pub))$$

Now, Alice and Bob have the same key, since order does not matter in the mixing process.

$$mix(A\_priv, mix(B\_priv, pub)) = mix(B\_priv, mix(A\_priv, pub)) = mix(A\_priv, B\_priv, pub)$$

What exactly is this *mix* function? It needs to be a one-way function, so that someone listening to this exchange at any point cannot undo the function, and Bob and Alice cannot derive the key of the other.

The answer comes through modular arithmetic. If we take the following equation,

$$x = a^b \bmod c$$ {*a* is a **primitive root** of *c*},

the result will clearly be of the form:

$$0 \leq x < c$$

However, since *a* is a **primitive root** of *c*, x has the special property that it distributes *evenly and randomly* among the domain of *c*, given different values of *b.* **This means that, given *x*, *a*, and *c*, the only way to find *b* is to guess and check.** We found our one-way function!

Let's now revisit the Alice and Bob case with our new "mix" function.

$$x = a^b \bmod c$$ {*a* is a **primitive root** of *c*}

1) Alice and Bob agree *a* and *c*. *c* should be very long; the length of c determines the strength of the one-way function since this is the number of values that would need to be checked to find *b*.
2) Alice and Bob each choose random private keys (integers). Let's say Alice chooses *i* and Bob chooses *j*.
3) Alice computes:

   $$x_1 = a^i \bmod c$$

   and sends $x_1$ to Bob.
4) Bob computes:

   $$x_2 = a^j \bmod c$$

   and sends $x_2$ to Alice.
5) ***Now the cool part!*** Alice computes the following:

   $$x_2^i \bmod c = \left(a^j \bmod c\right)^i \bmod c$$

   Bob computes the following:

   $$x_1^j \bmod c = \left(a^i \bmod c\right)^j \bmod c$$

   The following is a property of modular arithmetic for any *p*, *q*, and *r*:

   $$p^q \bmod r = (p \bmod r)^q \bmod r$$

Therefore, Alice and Bob have calculated (respectively):

$$a^{j^i} mod\ c$$

$$a^{i^j} mod\ c$$

Which are equivalent! They now have the same key with which to do **Option 1: Classic Encryption**, but they never publicly shared this key. Any of the sent messages can now be intercepted, but the interceptor would have no way other than brute force to decode the message.

Note: there are other encryption algorithms like RSA Encryption that solve the issues of Classic Encryption while not requiring *n* private keys to receive *n* encrypted messages and multiple transfers of data between Alice and Bob. These are slightly more complicated but rest on the same one-way function principles of modular arithmetic.