The Diffie-Hellman Exchange in Embedded Cryptography

Originalartikel

Backup

<html> Ready to design more secure devices but not sure where to start? Learn about Diffie-Hellman cryptography in this technical article. Our world has 7.55 billion people on it. While most of the planet’s inhabitants are good, a small number of bad actors can adversely affect everyone else’s digital lives. Hackers can drain our bank accounts, intercept our emails, stop our pace-makers, destabilize our power grids, or interfere with our elections. All from the comfort of their home. For years, chip manufacturers have left security/encryption to their customers to figure out. Their customers are electrical engineers who are pushing devices to market and have largely left security to the end user. Unfortunately, end users are usually wholly unequipped to implement or audit encrypted communication, so they ignore it. The effect of all of this passing-of-the-buck is that our internet and intranets are populated with Internet-of-Things (IoT) devices that have poorly implemented security or no security implementation whatsoever, which makes those devices prime targets for hackers to incorporate into their botnets. As an engineer, you are responsible for the safe operation of the devices you design. You should be encrypting internodal communications regardless of the physical transmission medium.   This article series will build up some basics of cryptography. Later, projects will demonstrate how to secure internodal communications. If you'd like a quick primer on cryptography before moving forward, check out my intro to cryptography concepts piece.

<h3>Shared Secret</h3> The safest way to encrypt a message between two parties is to combine it with a shared secret that is generated from random noise. The message can be combined with the shared secret via the XOR operation. A bad actor who sees the message after encryption would simply see a different noise profile, just as random as before, the message would be completely obfuscated and irretrievable.

The message recipient can read the message only by recombining the received data with the exact same shared secret via the XOR operator.

The result is the original message. Unfortunately, the shared secret can never be used again. If it was reused, a bad actor could look for similarities in the “noise” profile, and use those similarities to deduce the original messages.

   Anytime the same random number is reused to

encrypt multiple messages, similarities will exist that might allow the shared secret to be determined. In the image above, the same random number was used to encrypt the messages “All About Circuits” and “Setec Astronomy” (from the movie “Sneakers”). Even though both messages are encrypted with random numbers, they are encrypted with the same random number, which allows the messages to be deduced. To learn more search for “initialization vector” or “nonce” (number used once). The message sender and receiver need a different shared secret for each message, and they need a way to share the secrets so that a bad actor/eavesdropper cannot easily deduce the message they are sharing.

<h3>Diffie-Hellman</h3> Whitfield Diffie and Martin Hellman came up with a way for the sender and receiver of private messages to generate their shared secret by transmitting numbers in plain sight using modular arithmetic and exponentiation.

<h4>Modular Arithmetic</h4>

<h4 class=„c4“>$$p\;mod\;m=n\text{ or }mod(p,m)=n$$</h4>

The modulus operator determines the remainder left after division. Imagine cutting 5-foot lengths of rope from a 103-foot length until you’ve run out of rope. The modulus operator determines how long the last piece of rope is—in the case of the example above, 103-feet mod 5-feet = 3-feet. Each piece of cut rope is identical to the last, and if you were handed the left-over rope, it would be difficult to impossible to determine the number of pieces of rope that preceded it.

<h4>Exponentiation</h4> Exponentiation is the process of multiplying a number by itself a given number of times. The products of exponentiation become large quickly (2^{64}=18,446,744,073,709,551,616) which can consume available memory quickly. Fortunately, the process can be broken up into small problems that better fit into the memory of a microprocessor. The modulus operator takes care of overruns.   <h4 class=„c4“>$$2^{64}=2^{16}•2^{16}•2^{16}•2^{16}$$</h4>

<h4>Combining Ideas</h4> Diffie & Hellman realized that it was possible to combine the ideas of modular arithmetic and exponentiation to create a shared secret on two different systems. Both the sender and receiver participate in the generation of the secret and share public data freely. At the end of the key generation process, both the good guys know what the secret key is, and the bad guy is none the wiser. Here’s how good guys Alice and Bob can come up with a secret without bad guy Eve the eavesdropper figuring it out: First, Alice picks two prime numbers and transmits them to Bob – anyone listening in can see the numbers.

Next, Alice and Bob each choose a secret random number (a, b). Using exponentiation and modular arithmetic, Alice and Bob calculate two new numbers (A, B).

At this point, Alice and Bob have each calculated half of the secret message and they transmit their results to one another—Eve can see the results as well, but they are useless to her. To finish generating the secret, each person takes the other’s previous result and repeats the exponentiation and modular arithmetic to arrive at the same secret key.

It turns out that the laws of math allow the exponentiation and modulus operations to happen out of order and on different machines and still arrive at the same result. Each participant in the key generation generates part of the key and uses their partner’s result to calculate the rest. The order of calculation does not matter, both users arrive at the same result.

<h4 class=„c4“>$$((g^{a}\;mod\;p)^{b}\;mod\;p) = g^{ab}\;mod\;p$$</h4> <h4 class=„c4“>$$((g^{b}\;mod\;p)^{a}\;mod\;p) = g^{ab}\;mod\;p$$</h4>

You might be thinking to yourself that Eve can figure out the unknown quantities from the information provided. That is true. But the numbers used in the example were relatively small—actual implementations use much larger numbers. The large numbers make it extremely difficult to determine or guess the secret numbers (a, b) chosen by one or both participants when the only thing that the bad guy can do is look at the results of the intermediate calculations (A, B). It can be done but is computationally expensive. More examples of this process are shown below.

The final result is the shared secret that can be used to encrypt and decrypt data sent between two computers or devices. The shared secret is combined with the data via the XOR operation to obfuscate the data before transmission. After receiving the encrypted data, the XOR operation is used again to decrypt the data.

<h3>Summary</h3> The Diffie-Hellman exchange can be used to create secrets between two parties without revealing the secret to someone else. Computers can break the encryption if the secrets are too short—so very long keys must be used to make the task time-consuming. The next article will introduce Elliptic Curves used in cryptography.

 </html>