CS808: Computer Security Fundamentals

3.5 Article: Cryptographic randomness and one-time pads

Done: View

Randomness

Randomness is an important aspect of cryptography, more specifically random numbers. To be truly random means there is no way to predict an event, such as a number in a series. This means there are no discernible patterns. To identify a truly random number is to pick a number from a set, where each number is equally probable.

Real world examples of randomness include flipping a coin, or the national lottery. In general true randomness is extrapolated from natural phenomena which is unpredictable such as lava lamp movements or atmospheric noise.

In cryptography, pseudo-random numbers are also used. Pseudo random number generators (PRNG) are algorithms which produce a series of numbers which appear random. To do so, they make use of an initial seed value, a number which starts the generation of subsequent values. This seed should only be shared with those who require the ability to replicate the series, and where used in cryptography the seed is normally truly random. This is helpful in situations where you may be required to provide the sequence at another time. For example, PRNGs are used in simulations of viruses. Where PSNRs are used in cryptography you may see this as being referred to as Cryptographically Secure Pseudo Random Number Generator (CSPRNG). CSPRNGs have additional requirements, the full detail is beyond our scope, but for our purposes it is sufficient to note they should be indistinguishable from true randomness.

CSPRNGs can be sufficiently random to prevent attacks, but their deterministic nature means it can be used for encryption or other situations where you may be required to duplicate the values. It is also more efficient that trying to get a suitable list of truly random numbers.

One Time Pad

A one-time pad is where a truly random key stream is used to encrypt the message. Each character (e.g. letter or bit or byte) is combined with a character from the key stream. For example, consider a simple shift cipher where letters are shifted along by the key value in the corresponding position, looping back to the start of the alphabet as we have seen with the Caesar cipher. Let us assume the key provided below is truly random.

Key: lxtyo

Plaintext: hello

Ciphertext: tcfld (e.g. l (=12) + h(=8) -> t (= 20)

To be classified as a one-time pad, a key must meet the following requirements:

- It is truly random
- Only two copies should exist – one for encryption, one for decryption
- It is one use only, and is destroyed after use
- It is at least as long as the plaintext

One time pads have been around for some time. You can see details of how OTPs were used in the cold war on the cryptomuseum website Provided they are properly implemented, OTPs provide an unbreakable cipher. This is called “perfect secrecy”. No matter how much computational power you have, you can do no better than guessing randomly what the original message is.

Of course, practically this is challenging. Efficiently obtaining a sufficiently long truly random key is difficult, and sharing the key is also a challenge.

No matter how much computational power you have, you can do no better than guessing randomly as to what the original message was. Of course with very short examples such as the one detailed above, you would be able to guess all possibilities fairly easily, so there are always limitations.

For example, the middle square method, which was developed by John Von Neumann in 1949, is the first pseudo random number generator, or PRNG.

The process for this is to take the seed, squared it, line it up in the middle, and extract the middle part. That gives you the next value, which can then be squared, and repeat this process. If it's not an even number that's been outputted then you can pad this with zeros to get an even number. Eventually, this will repeat, but it could take a very long time. The point at which numbers begin to repeat in a PRNG is called the period. We have some PRNG where the period is so long that it does not have any practical implications.

Why not try to find the PRNGs available for your preferred programming language and generate a few numbers?

Last modified: Thursday, 8 September 2022, 11:17 AM

◄ 3.4 Lab: OpenSSL for RSA

Jump to...

3.6: Video: Digital Signatures (10:47) ►