Leaking Secrets

< Dripping Tap Graphic?>

Often a competent and experienced software engineer is tasked to implement a cryptographic algorithm. Unfortunately they may be unaware of what we call side-channels attacks.

For example normally a programmer couldn’t care less about the power consumption profile of their running program. However in many situations where cryptography is deployed an attacker is in a position to monitor power consumption. And the power consumption profile may leak information about a secret cryptographic key. And power consumption is just one example of what we call a side-channel. A list of some of the side channels we need to worry about is given here - https://en.wikipedia.org/wiki/Side-channel_attack

In short the problem is that the running algorithm may be inadvertently spilling out this side-channel information, which may completely undermine its security.

A classic example of this kind of attack is described in this recent piece of cryptanalysis by Bernstein and his co-workers https://eprint.iacr.org/2017/627

So how hard is it to write code that doesn’t spill side-channel information? Well in one sense its impossible, because there may well be side-channels that we haven’t thought of yet! But the good news is that there are a number of simple things a programmer can do to minimize the risk.

One idea is to write the code such that it runs in constant time (or takes a time strictly proportional only to the bulk amount of data being processed). Ideally the result is a straight-line program. It helps if the algorithm being implemented is exception-free, that is there are no data-dependent special cases that need to be handled. Fortunately such exception free algorithms usually exist, although be warned - they may not be the ones featured in the text-books.

The power consumption profile depends on the sequence of instructions being executed. If it is always the same sequence of instructions, independent of the data, and if the processor executes all of those instructions in a constant time, then the power consumption profile should not give too much away.

This assumes a processor that executes all of the necessary instructions in a fixed number of clock cycles, independent of the data. Which depends on the processor, but these days modern processors (for reasons that have nothing to do with a concern for security, but rather for pipeline efficiency) do execute most instructions in constant time. Except for integer division/remainder instructions, which most often take a number of clock cycles dependent on the data.

But of course we are also assuming that all memory accesses take equal time, and that will depend on the way in which cache memory is organised. OK, so maybe its all getting a little complicated. Let me try and make it a bit easier by suggesting some simple commandments. Thou shalt...

1. Not branch on secret data (or data derived from secrets).
2. Not derive an array index from secret data (to avoid cache attacks).
3. If you really feel you have to violate 1 or 2, look at ways of masking the secret with random data.
4. Try and eliminate all **if** statements. Loop unrolling may help.
5. Try and ensure that all loops iterate a constant number of times.
6. Avoid integer division.

7. Only use exception-free algorithms.
8. Be prepared to sacrifice some performance in order to conform to the above commandments.

OK, I didn’t quite make it to 10. But perhaps the reader might suggest a couple more.

Now programming is difficult enough at the best of times, and cryptographic algorithms are often complex. Clearly programming with extra constraints makes the job somewhat harder. But to avoid future embarrassment, it really is worth making the extra effort. We have used these ideas in our own cryptographic libraries to try and minimize side-channel leakage. See https://github.com/milagro-crypto/amcl

A final appeal. Computer language designers! Want a feature to set your language apart from the crowd? Then help us out here! How about allowing variables and data structures to be marked as “secret”, and a compiler that kicks out a warning like “Line 234 - Warning 171: branching on secret data”? That sort of help from a compiler would be much appreciated.