How unhackable code is written

We all know how code writing usually looks like : someone knows what to code , they go ahead and code it , solve obvious bugs and usually test it by writing some specific tests. And the testing shouldn’t be done for every line of code, although tempting for a beginner. That is why writing tests isn’t the best way of ensuring that software works as intended : we would have to think of all the possible cases in order to ensure that everything works( this being a tedious and surely unsuccessful task ) .

The solution to this problem lays in something called formal verification. What this aims to achieve is testing software with the certainty that theorems are proven by mathematicians. Instead of writing tests for specific cases , the program is checked using a proof checker on a mathematical formula(formal specification) that describes the behaviour of the software. The hardest part of this is usually defining the proof in mathematical terms , since in some instances that is quite the challenge. Take for example encryption , where the question may be what it means to keep a secret.

This comes with one major disadvantage though : a programme verified in this way could be even 5 times as long as a traditionally written one. This was one of the main reasons why this type of verification didn’t exactly take off initially . But , while before the internet the bugs were tolerable , once everything became connected , someone’s vulnerability became everybody’s problem. And so , with more manageable goals and more important than ever , formal verification gets its comeback.