Achieving security in functional programming

Functional programming is a programming paradigm in which we try to bind everything in pure mathematical functions style. It is a declarative type of programming style. Its main focus is on “what to solve”. It uses expressions instead od statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables. Those functions have some special features discussed below.

**Functional Programming is based on Lambda Calculus:**   
Lambda calculus is a framework developed by Alonzo Church to study computations with functions. It can be called as the smallest programming language in the world. It gives the definition of what is computable. Anything that can be computed by lambda calculus is computable. It is equivalent to Turing machine in its ability to compute. It provides a theoretical framework for describing functions and their evaluation. It forms the basis of almost all current functional programming languages.   
*Fact: Alan Turing was a student of Alonzo Church who created Turing machine which laid the foundation of imperative programming style.*

**Concepts of functional programming:**

* **Pure functions**
* **Recursion**
* **Referential transparency**
* **Functions are First-Class and can be Higher-Order**
* **Variables are Immutable**
* **Data structure**

**Pure functions:**These functions have two main properties. First, they always produce the same output for same arguments irrespective of anything else.   
Secondly, they have no side-effects i.e. they do not modify any arguments or local/global variables or input/output streams.   
Later property is called immutability. The pure function’s only result is the value it returns. They are deterministic.   
Programs done using functional programming are easy to debug because pure functions have no side effects or hidden I/O. Pure functions also make it easier to write parallel/concurrent applications. When the code is written in this style, a smart compiler can do many things – it can parallelize the instructions, wait to evaluate results when needing them, and memorize the results since the results never change as long as the input doesn’t change.

**Recursion:**There are no “for” or “while” loop in functional languages. Iteration in functional languages is implemented through recursion. Recursive functions repeatedly call themselves, until it reaches the base case.

**Referential transparency:** In functional programs variables once defined do not change their value throughout the program. Functional programs do not have assignment statements. If we have to store some value, we define new variables instead. This eliminates any chances of side effects because any variable can be replaced with its actual value at any point of execution. State of any variable is constant at any instant.

**Functions are First-Class and can be Higher-Order:**First-class functions are treated as first-class variable. The first class variables can be passed to functions as parameter, can be returned from functions or stored in data structures. Higher order functions are the functions that take other functions as arguments and they can also return functions.

**Variables are Immutable:**In functional programming, we can’t modify a variable after it’s been initialized. We can create new variables – but we can’t modify existing variables, and this really helps to maintain state throughout the runtime of a program. Once we create a variable and set its value, we can have full confidence knowing that the value of that variable will never change.

If we follow this concepts and principles we can achieve security of functional programming.