SecureDart is a programming language, designed to secure the server-side component of web applications against attacks from clients. In SecureDart, both client- and server-side programs are written in Dart and annotated with SecureDart annotations. This language combines static and dynamic approaches to verify integrity and confidentiality of information.

The dynamic approach is implemented via replay. When the client generates a request, it records the execution trace that produces, and sends the trace together with the request to the server, packaged in a certificate. The server then re-executes the client code using the trace and verifies that the re-execution produces the same result as the request. For efficiency, SecureDart replays only partial client executions, based on manual annotations in classes and methods in the code. The client-side code is divided by the developer in two components: Trusted and Normal. The first is composed by the code that may generate security critical requests on the server, and the second by that does not generate such requests. Validation, via replay, only needs to be performed on requests generated from the first component. It uses an information flow type system to ensure that potentially harmful requests have certificates and are validated via replay. SecureDart was evaluated on four web application, incurring acceptable extra runtime overhead.

-The appendix "Server lattice Structure", suggest that there are three lattices for the server-side types, but in the Figure 9 are presented only two, and not is referenced the third.

-The formal syntax of the language is not exposed.

-It says nothing about the proofs of the language.

-In the experiments, it does not say anything about the amount of code that was reply.

-In Chapter 6 is said that: for each program, the sum of its LoC(app) and LoC(lib) gives the total number lines of compiler checked Dart code. They should not be added too LoC in RPC decl???

- In Chapter 6 is said that In Chapter 6 is said that: The value of the expression can be assigned to another memory location as long as the label type of the expression is higher than or equal to that of the destination memory location. The label type of the destination variable x must be lower than or equal to that of the program counter to constrain implicit flow. Should not it be the other way around??