Title page

In this tutorial we will give an overview of Jasmin, a programming language designed to write high-assurance high-speed cryptography.

We are Miguel and Santiago from MPI-SP.

The tutorial will be pretty hands-on, so let's begin right away with an example.

[HINT: NEXT FRAME]

memeq

Jasmin is a low-level programming language with C-like syntax.

This is a simple function that checks if two memory regions `p` and `q` coincide in the first `n` quadwords.

Jasmin comes with tools to verify that code is

- correct (relative to a specification);
- safe (e.g., it doesn't divide by zero); and
- secure (e.g., against side-channel attacks).

memeq

It has assignments, whiles, ifs, and functions.

But is is lower-level than C: it really is structured assembly.

Each Jasmin instruction must correspond to an assembly instruction.

[HINT: NEXT FRAME]

memeq 2

Indeed, we can use assembly instructions directly in the Jasmin source file.

[HINT: NEXT FRAME]

Here I'm using the `INC` x86-64 instruction to increment `i`, this is the dashed red arrow.

memeq 2

There are two reasons we need a language at a lower-level than C: efficiency and security.

Regarding efficiency, we can be much more precise and detailed with our optimizations.

Regarding security, we can see many low-level problems at source-level, without the compiler getting in the way.

memeq 2

Well, but then why bother with Jasmin when we have assembly? Same reasons, different emphasis: security and efficiency.

A structured language with a clearly defined semantics avoids tons of the problems of trusting large assembly codebases.

It is also important for efficiency: higher-assurances on our code allows more aggressive optimizations.

memeq 2

Jasmin gives structure to assembly programs with functions, conditionals and loops without compromising efficiency or security.

[HINT: NEXT FRAME]

Their compilation is standard and predictable:

- functions (in dotted green) compile to a label and a return;
- loops (in continuous red) compile to a check and a backward jump; and
- conditionals (in dashed blue) compile to a check and a jump to the else branch.

More online

Now let's write some Jasmin.

[HINT: Have people open the files and setup Docker.]

[HINT: Leave the Formosa slide while people work.]

otp

A bit more on syntax: operators are almost the same as in C, for instance here addition is plus and exclusive or is caret.

Functions marked `export` are the only entry points of the program. They are what is visible from C (i.e., by the linker).

Raw memory addresses use square brackets instead of asterisk, and you can specify the size of the load by prefixing it with `u8`, `u16`, `u32`, `u64`, and so on.

The default load size is register size.

otp_fixed

Array accesses have the same syntax as in C.

gimli: sboxes

Until we're done with sboxes.

safety

Show how to avoid common pitfalls.

gimli: structure

Some time?

side channels

Show the CT checker and the SCT checker.

gimli: finish & AVX