Implementation and Evaluation of ZK-PoK

Boran Car

K. U. Leuven

January 18, 2012

Outline

The Problem

- Zero Knowledge Proofs of Knowledge
  - Prove knowledge of something without actually disclosing it
- A base for
  - E-petition
  - E-voting
  - E-cash
  - Anonymous credentials (e.g. driver license)

Anonimity

- Current electronic methods require log-in
  - The user has no choice but to give his personal information
  - "Big-Brother" can track you
- Current paper methods too slow
  - Waste paper
  - Waste man-hours
    - Counting
    - Processing

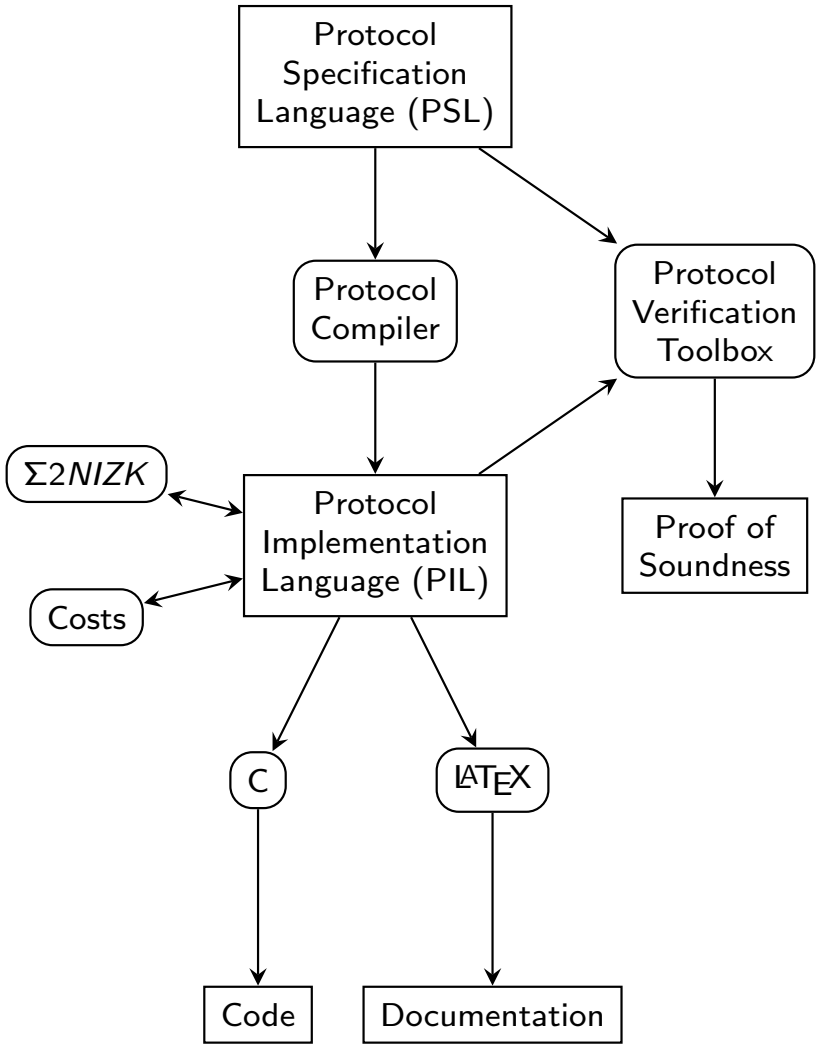
The Goal

- Proving something without disclosing it is not very simple
  - Luckily, there exist some frameworks
- The initial goals were:
  1. To evaluate existing ZK-PoK frameworks
  2. To implement DAA - Direct Anonymous Attestation
- Later this has changed to:
  1. Evaluate existing ZK-PoK frameworks
  2. Implement a custom framework
  3. Implement DAA on top of the custom framework

Why go to custom framework?

- Avaiable frameworks not fit for small embedded devices
  - CACE - Generated C code polluted with debugging info not easily strippable
  - ZKPDL - Uses C++ which is not available for all small devices, interpreter framework
- DFG difficult to extract from existing frameworks
  - DFG needed for conversion to HDL
  - Easier optimizations (e.g. intermediate result caching)
- Domain specific language fit for cryptographers

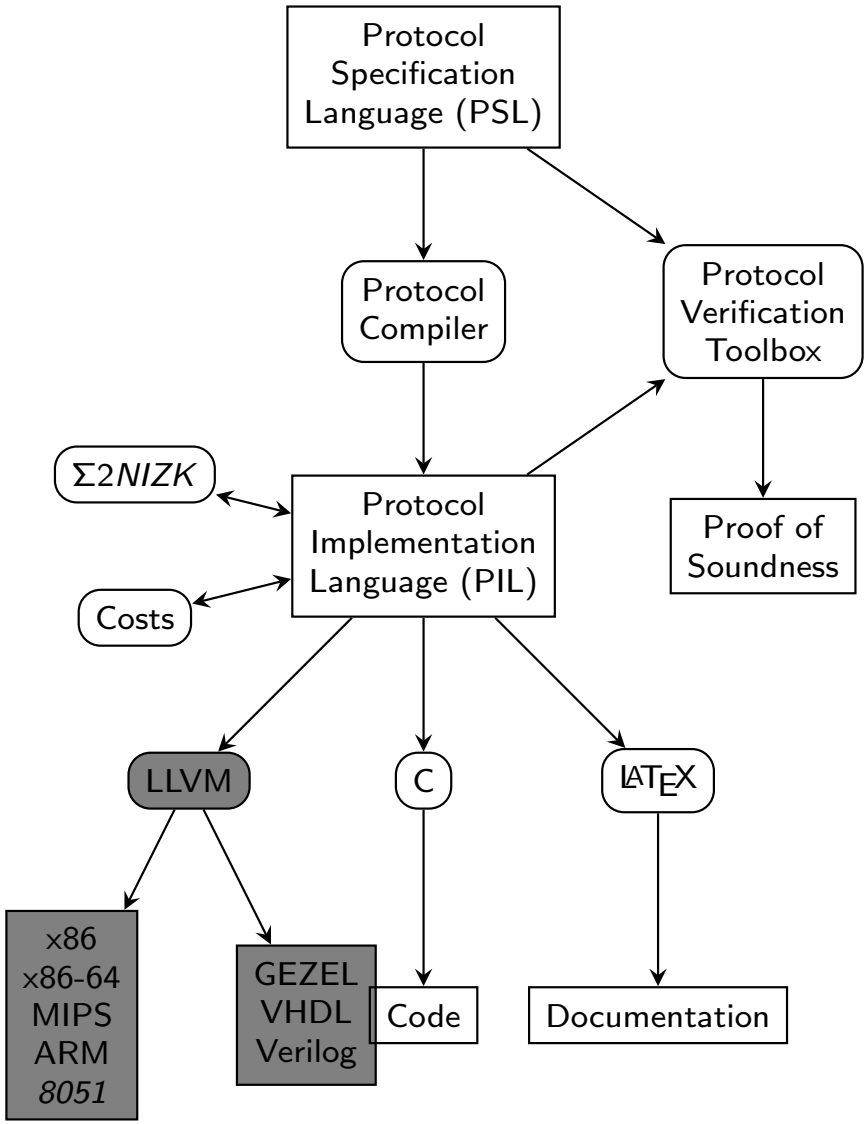
CACE workflow



Custom framework

- Extends CACE (Computer Aided Cryptography Engineering), an European project
- Uses LLVM - Low Level Virtual Machine
  - A VM Load-Store RISC architecture
    - ★ Infinite SSA registers
- Allows for
  - Interpreted
  - Compiled
  - JIT Compiled

Custom framework workflow



LLVM

- A proven compiler framework
- Already used by SVA (Secure Virtual Architectures)
- Uses in many other fields (e.g. GPU, CPU dynamic translation)
  - Guarantees maintenance and optimizations (Linus' Law: "given enough eyeballs, all bugs are shallow")
- Backends for different architectures (e.g. x86, x86-64, ARM, MIPS)
- Uses SSA registers
  - DFG easy to extract (HW translation possible)

GEZEL

- A cycle-based HDL (Hardware Description Language)
  - Finite-State-Machine + Datapath (FSMD) model
- Cosimulation with embedded cores (ARM, 8051, AVR)
- Code generator for VHDL and Verilog
- Open source

Planning $\approx$ 3 weeks remaining

1. Theoretical study $\approx$ 3 days remaining
   1. ZK PoK
   2. DAA $\approx$ 3 days remaining
2. Evaluate existing frameworks
   1. Evaluate a sample protocol (Schnorr's)
   2. Add extensions and couple
      - ★ Add terminal functionality to CACE
      - ★ Add terminal functionality to GEZEL
      - ★ Implement a dummy prover/verifier in GEZEL
3. Implement custom framework $\approx$ 1 week remaining
   1. Implement parser to AST (Abstract Syntax Trees)
   2. Implement code-generation to LLVM $\approx$ 1 day remaining
   3. Implement a JIT compiler $\approx$ 1 day remaining
   4. Implement an LLVM backend for 8051 $\approx$ 5 days remaining
4. Implement DAA on top of custom framework $\approx$ 3 days remaining
5. Thesis text and presentation $\approx$ 2 weeks remaining

Custom framework status

Arrays	Not started
Assignment	Done
Conditional	Parser done
Iteration	Not started
Function call	Done
Function defintion	Done
Global variables	Done
Local data-flow	Done
Return values	Mostly done
Type inference	Basic

The End

- Thank you
- Questions?