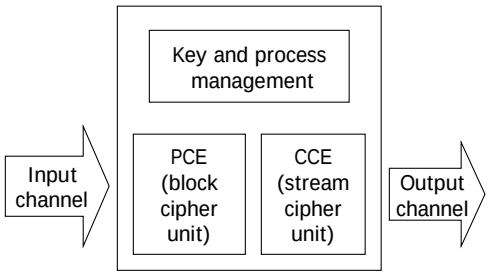
Formal Models of AIM

John Launchbury
Oregon Graduate Institute

March 14, 2001

1. AIM Overview

The AIM chip is a crypto-processor produced by Motorola. AIM can encrypt and decrypt data on multiple interleaved channels, using independent algorithms and keys for each channel. Internally, AIM contains two custom cryptography units: the PCE designed for block ciphers, and the CCE for stream ciphers. In addition, there is a general-purpose processor that handles channel setup and key management. AIM also contains extensive protection and anti-tamper features. One exciting feature of the AIM design is its mathematically assured separation kernel (MASK). The design of MASK is based upon a mathematical model, which, it can be proved, ensures that separate cryptography processes cannot observe or influence one another, and hence that different channels are truly independent.



The PCE block cipher unit contains four parallel execution components: a permutation unit; a non-linear function unit; a linear function unit; and a general-purpose arithmetic-logic unit. These are programmed directly in a microcode in which pipeline delays and unit parallelism are written explicitly. Our goal in this AIM-modeling project is to explore methods for introducing abstraction and additional mathematical insight to the structure of microcode programs that are destined both for the PCE and CCE. This short note is designed to provide an overview of the project.

2. AIM Microcode

We have built a semantic model of the PCE. The model provides a precise yet abstract description of the behavior of the AIM microcode. Building the model involved gathering information (verbally and in documentation form) about the AIM chip and about the PCE unit in particular. After a detailed analysis of the PCE unit's operational behavior we expressed the model as an executable interpreter simulating the core functionality of the PCE unit. As the project proceeds, we plan to modify the interpreter to refine the design of the semantics and to model more of the functionality of the PCE unit.

The permutation unit of the PCE is a wide multi-way switch where each output bit can be driven by any input bit. However, programming the permutation unit by directly specifying each output bit one-by-one loses all information regarding *why* the particular permutation is correct. We designed a small domain-specific language for specifying the structure of permutations. The language enables clean and brief descriptions to be written that are amenable to peer review. The language also has a formal semantics that is used to calculate the appropriate configuration of the permutation unit that meets the description. This configuration is written to a file in a form expected by the AIM development environment.

Most recently, we have gone further, and studied the nature of interactions between the permutation unit and the non-linear function unit. This is particularly relevant for constructing implementations for S-boxes. Again we are capturing the semantics of these interactions at a higher level than previously, and now are able to describe S-box functionality almost as compactly as the permutations themselves.

3. CCE Semantic Model

The AIM chip contains a separate programmable component, the CCE, designed for stream ciphers. The CCE architecture is based on programmable decision trees, yet the cryptoalgorithms are expressed in terms of boolean functions provided by the user. The mapping between these boolean functions and the decision trees are far from obvious. We plan to express the boolean functions using BDDs and then, by applying heuristics, attempt to find close to optimal variable orderings. We have a hypothesis that variable orderings which reduce the size of a BDD representation of a boolean function also reduce the size of decision trees required to implement the function. If valid, this technique will provide the core of a compilation technique for generating configurations for the CCE unit.

4. Verification of Microcode Implementations

We plan to go beyond description of the AIM PCE and CCE units, and explore microcode generation and verification. We have co-developed *Cryptol*, a domain-specific language for describing cryptoalgorithms, which we use to provide specifications of crypto-algorithms. Our plan now is to compare crypto algorithms implemented as PCE microcode against the specifications of these algorithms. Our verification experiments will be based around a combination of theorem proving and model checking, the latter exploiting BDDs and other SAT algorithms.

The Cryptol DSL is able to generate C-code implementations of crypto algorithms. It is interesting to consider what is involved in writing a code generator for Cryptol that produces AIM PCE microcode. Many parts of this are straightforward. One challenge, however, is to find effective ways to exploit the permutation unit in conjunction with the linear and non-linear function units, as these implementation details are inappropriate in very high-level algorithm specifications.