Implementation of ECDSA on Combo6X card

Abstract: The paper describes implementation of cryptographic coprocessor performing operations on elliptic curve points with affine coordinates in GF(2^m). Coprocessor will be implemented in Xilinx Virtex II PRO XC2VP50 on a PCI-64 card Combo6X.

The Combo6X card contains two Virtex chips, one acts as a PCI bridge, the other one is fully customizable. The card comes with Linux drivers and VHDL modules that allow easy implementation of custom hardware on the card. It allows writing respectively reading data to respectively from an address on an inner bus in the customizable FPGA.

The coprocessor has an interchangeable arithmetic unit and is used to compare two possible arithmetic units. The two arithmetic units differ in the used type of basis, the first one uses *polynomial basis*, the second one uses *normal basis*. The polynomial basis arithmetic unit performs multiplication using LSB multiplier and division (inversion) using extended Euclidean algorithm. The normal basis arithmetic unit uses Massey-Omura multiplier with variable digit width for multiplication and Itoh-Teechai-Tsuji algorithm for inversion.

Previous work on this topic used post place-and-route simulation and suggests that where an optimal normal basis exists, the normal basis solution arithmetic performs better. This paper focuses on comparison of the two solutions in physical hardware, taking into account delays of the whole hardware system.

Two time values will be measured. The first one will measure only the processing time in the FPGA, the second one will measure the processing time including transfer on PCI-X. Because the second time has to be measured by software, it will be necessary to compute keys in a batch. Otherwise the error introduced by the time measurement itself would be greater than the measured value.

Two possible approaches to the batch processing have to be considered. The first one is writing all the data into the FPGA, computing all the keys and reading all the results from the FPGA. The second one writes, processes and reads each key individually, thus adding a significant communication overhead.