Hardware Implementation of Quasigroup Encryption for SCADA Networks

William Mahoney, Abhishek Parakh
School of Interdisciplinary Informatics
University of Nebraska at Omaha
Omaha, Nebraska 68182
wmahoney / aparakh @unomaha.edu

Matthew Battey
Aspect Software, and Computer Science Department
University of Nebraska at Omaha
Omaha, Nebraska 68182
mbattey@unomaha.edu

Abstract—We present an efficient hardware implementation of a quasigroup block cipher system. Power and resource comparisons are done with AES and we show that the proposed quasigroup system provides an inexpensive alternative for SCADA networks and other low powered, resource constrained devices.

Keywords—FPGA, hardware encryption, critical infrastructure, industrial control systems

I. INTRODUCTION

The widespread deployment of network connected cyber systems, based on embedded computing platforms, requires the development of inexpensive encryption algorithms that are suited for these resource-constrained environments. As the amount of data being gathered, transmitted and received by these devices grow, so does the drain on battery power. At the same time, these embedded systems are often mass-produced, and a result is that manufacturing component costs are a concern; an increase in the cost of one component is multiplied by the potentially millions of devices being built. An example critical area, where embedded systems are used, is Supervisory Control And Data Acquisition (SCADA) systems like the power grid, water purification systems, oil and gas delivery, as well as trains and transportation systems. Often in embedded systems, including SCADA, the hardware requirements and the battery life are such deterrents that data encryption is entirely left out, or manufacturers decide to use a resource-inexpensive, but deprecated, algorithm such as RC4 with small key sizes.

We have developed a quasigroup-based encryption algorithm that holds promise for these low-cost and low power embedded systems applications. Previously we have reported on the software implementation of the algorithm [3], [4]; in this paper we report on our work in progresstowards a single-chip quasigroup device based on Field Programmable Gate Arrays (FPGAs). This design will be a drop-in design for a computer hardware engineer wanting a reasonable throughput but low cost solution for transporting data securely. The algorithms developed using quasigroups use table look-up operations with minimal algorithmic complexity along with shift operations. The algorithm is excellent at destroying the existing structure of the data, making the output close to the output of a random number generator. This paper reports on the work thus far.

Section two lists our motivation and the methods currently in use for encryption in embedded systems. Next, a brief overview of quasigroup encryption will be presented. We have implemented three designs for comparison and the designs and results are in sections four and five respectively. Conclusions and future work follow.

II. EXISTING METHODS AND MOTIVATION

NIST has from time to time standardized cryptosystems beginning with DES and 3DES to the more modern AES cryptosystem. Today AES with key sizes of 128, 192 and 256 bits is commonly used. But AES requires much more computational power than its predecessors. What appears to be a minor cost increase necessitated by the horsepower needed by AES translates into millions of dollars in the world of individual electrical consumers. So over 60% of websites still use RC4 with a 128 bit key for encrypting data, mainly because of the low computational requirements and high throughput of RC4. But recently RC4 has been shown to have vulnerabilities, and meanwhile, due to the increasing popularity of RFID and sensor networks, several lightweight cryptographic algorithms have been developed in recent years. The EU ECRYPT network began an organized effort to develop new stream ciphers for widespread adoption under the eSTREAM project [25]. The project did not, however, look at development of block ciphers. Many of the other lightweight cryptosystems have been modified (stripped down) versions of ECC [27], [28] and DES [26] algorithms. For example, [26] proposes a lightweight version of DES for RFID chips and uses a single s-box repeated eight times in order to improve on storage.

A PRINTcipher was proposed for integrated circuit printing [29] however its cryptanalysis has revealed some weaknesses [30], [31]. Similarly Piccolo [32] was proposed as an ultra-lightweight block ciphers for RFID tags and sensor nodes. Fault analysis of Piccolo is presented in [33], [34] where it is shown that the key candidates can be reduced significantly based on a few correct and faulty cipher texts.

TWINE was proposed [35] as a lightweight cipher for multiple platforms and low-end microcontrollers. Some cryptanalytic attacks have been proposed on it with slight improvements than brute force attacks [36]. PRESENT is another general purpose block cipher developed for low-power consumption and high chip efficiency [37]. It has also been

This research is partially funded by AFOSR FA9550-10-1-0341 - Cybersecurity of Critical Control Networks.
adopted by International Organization for Standardization and International Electrotechnical Commission as a standard algorithm. LED is another recently developed compact block cipher [39]. Some weaknesses have been discovered in LED, as well, that enable attacks on it [40].

These are only a few efforts related to development of lightweight cryptosystems. Their main applications have remained in the RFID and sensor network application domains and not specifically towards SCADA systems or cyber-physical systems (CPS). In SCADA and CPS additional hardening against reverse engineering, cloning and side channel attacks may be required.

Quasigroups can be used for error correction and in the construction of message authentication systems [2]. Also, the Latin square has been researched previously with respect to encryption, where they are referred to by their quasigroup name [3],[4],[8]. Stream ciphers using quasigroups, as well as public key implementations with quasigroups, were investigated by Gligoroski and others [9],[10],[11],[12]. Satti and Kak [22] looked at multi-level stream cipher implementations of quasigroups, where they also combined the implementation with indices and nonces in order to improve the strength of the resulting encryption. An all-or-nothing system was implemented by Marnas et al. [18] but they use quasigroup encryption only to replace the exclusive-or operation in the all-or-nothing system; the actual encryption is done with other cryptosystems.

The quasigroup encryption system is primarily a substitution and permutation system, a type of cryptosystem used frequently in the encryption of speech [7],[19]. Public key systems such as elliptic curve and NTRU systems [6] have lower power consumption relative to computationally expensive secret key systems. In the case of our algorithm the only necessary operations required are table look-up and rotation so we also anticipate low power consumption.

III. OVERVIEW OF QUASIGROUP ENCRYPTION

Quasigroups are similar to Latin squares and can be represented in the form of a matrix. In general a quasigroup matrix consists of elements from 0 to \( n - 1 \) such that no element in a row or a column repeats. A corresponding inverse quasigroup matrix is also defined that is basically a reverse mapping of elements. Interestingly, the use of Latin squares for cryptography is not new [20]; a simple exclusive-or (\( \oplus \)) table is an example of a commonly used Latin square. The (\( \oplus \)) operation is used as the basis for one-time-pads, which is the only cryptosystem providing perfect security. Further, (\( \oplus \)) conforms to the structure where \( v_{ij} = r_i + c_j \mod 2 \); indices \( r_i \) and \( c_j \) starting at zero. This allows for commutative operations, i.e., \( a \oplus b = b \oplus a \). However, in general, quasigroup operations are not necessarily commutative and the lack of this property provides greater security for shorter key lengths, which is desirable for embedded systems. Higher order quasigroups may be viewed as generalization of (\( \oplus \)); an order \( n \) quasigroup, denoted by a \( n \times n \) matrix, can maintain the commutative properties of (\( \oplus \)) and hence be \( v_{ij} = (r_i + c_j) \mod n \).

Conventionally, quasigroup based ciphers have been used as stream ciphers only, where the entire quasigroup was kept secret and a single byte seed was used to initiate the encryption process. The stream cipher system processes input bytes one byte at a time. The initial output \( c_0 = s \times m_0 \), and subsequent bytes are generated by \( c_i = c_{i-1} \times m_i \) where \( c_i \) is the encryption of byte \( m_i \). The method provides a high throughput but is vulnerable to a known plaintext attack.

A. Block Cipher Design

Our quasigroup block cipher [4] was designed based upon the ideas of confusion and diffusion. In order to make the block cipher design consistent with existing encryption practices we used a 256-bit key with a 128-bit block size.

Since we continue to work with one byte at a time, we divide the 256-bit encryption key into 32 one-byte seeds and use a 256x256 quasigroup. Each of these seeds is used in rounds of encryption as follows:

1. Generate a random 256-bit encryption key and divide it into 32 one byte (8 bit) seeds. Each seed will be used once per round of encryption.
2. Divide the source data into 128 bit (16 byte) blocks.
3. For each block do the following:
   - For each 8-bit seed byte in the key do the following:
     - Using the current block as a stream of 16, 8-bit integers, apply the current 8-bit key as the quasigroup cipher seed and encrypt the block.
     - Rotate the currently encrypted 128-bit block left by 1, 3, 5 or 7 bits depending on the index of the current seed byte modulo 4.

Note that although each block is 128 bits long, when applying quasigroup encryption we divide the block into 16, 1-byte sub-blocks. After every round of encryption, all the bits in the sub-blocks are taken together prior to the left rotation.

B. Overview of Software Implementation

We first implemented the quasigroup scheme in C# since we could then compare it against a popular implementation of AES. For the quasigroup table, a \( n \times n \) array is constructed such that row zero contains \([0,1,2,...]\), row one contains \([1,2,3,...]\) and so on. The table is then modified such that the rows are randomly rearranged using Knuth/Fischer-Yates shuffling, and then the columns are randomly rearranged as well.

The Statistical Test Suite (STS) provided by the National Institute of Standards and Technology (NIST) [21] is used to evaluate the quasigroup scheme. A “P value” is the result of each individual test in the NIST-STS suite. This value of P represents the probability that a perfect random number generator would produce a sequence less random than the result of the encryption. Obviously the plaintext does not pass any tests in the suite, but the output of the quasigroup scheme must pass all of the tests in the suite. The test suite includes several tests which can be used with varying block lengths. Upon completion of each test, the P value is given based upon the resulting output from encryption. The data we used included clear text made up of zero bits, clear text made up of one bits, text containing all letter ‘E’ s, and the clear text of the
IV. OVERVIEW OF THREE FPGA IMPLEMENTATIONS

We desire to take the quasigroup scheme and implement it in low-cost hardware for the embedded systems world. Field Programmable Gate Arrays will be used as our design target. Field Programmable Gate Arrays (FPGAs) are components that can be programmed for a specific purpose by the circuit designer, as opposed to custom chips that are programmed at manufacturing time. Internally, an FPGA contains a collection of Common Logic Blocks (CLBs) connected by typically a grid system of wires \[14\],\[16\]. Although FPGAs are slower than custom ICs, and additionally they are more expensive, they can be fingerprinted so that they are specific to a customer \[17\]. And FPGAs can be designed specifically to make reverse engineering difficult; Wollinger provides a good overview \[23\] on the reverse engineering process. Newer information is available in Huffmire \[15\]. The difficulty of reverse engineering makes FPGAs a good candidate for encryption.

In order to research whether quasigroup-based encryption will offer the opportunity to save costs in an embedded environment, we implemented three different algorithms in FPGAs. The initial design is a translation of the C# software into the Verilog hardware description language. For comparison purposes we next test an open-source Verilog realization of AES. The difficulty of this comparison is that the open source AES implementation requires a significant hardware component due to the large number of I/O pins required in the FPGA. Their design is very fast but requires the complete clear text to be available, in parallel, simultaneously; thus a component with a very large “pin count” is required. To aid in comparison then our third implementation is the quasigroup front-end to load the data, with a minimal pin count, but with the parallel AES algorithm inside the FPGA. Each of these designs is detailed.

A. The Quasigroup Implementation

The first design is an FPGA implementation of the basic Quasigroup algorithm. The design goal is to keep costs down at the expense of speed, so every effort has been made to minimize the interconnections to the component since they are equivalent to cost. From an external perspective, this design has a 16-bit address bus, two 8-bit data busses, a 3-bit select logic, and clock and reset connections. Because the encryption requires the 256 × 256 size quasigroup table, an external 64K byte-addressable read-only memory is assumed for now; this accounts for the 16-bit address bus in the design. When the design is reset, it initially reads in the key information, the initialization vector (IV) for the cipher block chaining (CBC), and then successive 16-byte blocks of data. When the data is encrypted, the resulting data is read out on one 8-bit bus while the next clear text is written to the component on the other 8-bit path. The quasigroup algorithm is essentially serial, so once the final byte of the clear text is written the design completes the encryption in 560 clock cycles. This corresponds directly to a translation of the C# implementation of the algorithm.

B. Comparison Design - Parallel AES

In order to compare the quasigroup scheme versus others we selected an open source AES encryption core from opencores.org. This design is a fully parallel implementation, and as a result it requires three 128-bit buses to transfer the encryption key, the clear text, and the result. Because of the high pin count we expect this design to be significantly more expensive than the quasigroup design. The clear text and key are loaded over two of the busses and once loaded the encryption results are ready in 21 clock cycles, a considerable advantage for AES over quasigroup. The result is read out on a 128-bit bus as additional blocks are loaded. Further, the design is pipelined so that successive blocks of clear text can be presented to the design on each clock, and the corresponding cipher text is available 21 clock cycles later; for this reason the throughput of the design is considerably higher than the quasigroup method, but with a corresponding addition in the complexity of the design and interface. Also, the potential for CBC is limited since the results of the previous encryption block cannot be exclusive-or’d with the next block without incurring the penalty of waiting these 21 cycles. For this reason this basic AES implementation does not support CBC.

C. Hybrid Front-end/AES Design

A limiting factor for the implementation of the basic AES design is interconnect required to the FPGA. The three necessary 128-bit busses dictate a very high I/O pin count on the actual selected component, which considerably impacts the cost of that component. We thus desire to mix the AES encryption “back end” which performs the encryption with the

<table>
<thead>
<tr>
<th>Test</th>
<th>AES</th>
<th>Beowulf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E</td>
<td>0xFF</td>
</tr>
<tr>
<td>AE</td>
<td>988</td>
<td>989</td>
</tr>
<tr>
<td>BF</td>
<td>992</td>
<td>990</td>
</tr>
<tr>
<td>CSF</td>
<td>990</td>
<td>993</td>
</tr>
<tr>
<td>CSR</td>
<td>994</td>
<td>989</td>
</tr>
<tr>
<td>FFT</td>
<td>990</td>
<td>988</td>
</tr>
<tr>
<td>FREQ</td>
<td>992</td>
<td>992</td>
</tr>
<tr>
<td>LR</td>
<td>991</td>
<td>987</td>
</tr>
<tr>
<td>Rank</td>
<td>990</td>
<td>995</td>
</tr>
<tr>
<td>Runs</td>
<td>994</td>
<td>988</td>
</tr>
<tr>
<td>Serial 1</td>
<td>990</td>
<td>992</td>
</tr>
<tr>
<td>Serial 2</td>
<td>986</td>
<td>993</td>
</tr>
</tbody>
</table>
quasigroup encryption “front end” that loads the data, so as to compare designs that have similar pin count. In this hybrid design, the AES key, IV, and clear text are presented to the chip in the same manner as the quasigroup method - one byte at a time using the combination of the address and chip select lines from the component. Once the entire clear text is into the chip the design waits the requisite clock cycles, and as in the quasigroup design clocks out the cipher text while clocking in the next clear text. This design and the original quasigroup design are thus similar from the external standpoint, with the exception that not all address connections are necessary since the hybrid design does not require an external quasigroup table ROM. Since the key length for AES is 16 versus 32 bytes the hybrid design can initialize the key in fewer cycles, but initialization is of no importance as it only occurs once.

V. EXPERIMENTAL RESULTS

We desire to determine the power requirements as well as the approximate costs associated with the three designs, in order to determine whether our quasigroup design warrants additional research. Initially our plan was to actually implement the designs using evaluation boards - reference designs - from FPGA vendors. But the software simulations of the FPGAs have such high fidelity that there was no need to test with actual hardware.

We utilized the Altera software package Quartus-II as well as the Modelsim-Altera simulator to perform register transfer level and then gate level simulations of the designs. The package is sufficiently sophisticated that we can obtain exact timing requirements and determine maximum clock speeds as well as power consumption. The Quartus-II software will also select the specific component necessary to hold the design, based on the required pin counts as well as the functionality necessary for the implementation. This turns out to be a great feature as it directly allows us to compare actual costs of the FPGAs. Although the quasigroup encryption is sufficiently small that it could presumably sit in a very minimal FPGA, we decided to conduct an apples-to-apples comparison and stick with the same families of Altera FPGAs, with the precise component selected by the software.

Specifically we tested all three designs with an automatically selected component from Altera’s Cyclone-II product line, and then again with the Cyclone-III product line. In all cases we placed no restrictions on the automatic selection of the component, allowing the software to pick the mapping to I/O pins, for example. In the case of the quasigroup design implementation we assumed that the external ROM necessary for the encryption table was not a factor in the speed calculations, since it will be internal ROM eventually.

Table II describes the results - but with the external ROM omitted for the quasigroup design. Specifically, the number of clocks required per 16-byte block, and the throughput in bytes per clock are presented first. Our thinking is that this allows the reader to estimate what a faster frequency component would gain in throughput, since the number of clocks for the encryption is constant. Next we have the actual clock period measured as per second based upon this clock period.

Clearly based on the preceding table, quasigroup encryption does not look attractive from the perspective of throughput. However SCADA systems typically do not require high throughput anyway. Our driving factors in areas such as “smart grid”, recall, is cost and power. Table III compares the three designs from these perspectives. To estimate the costs we simply took the corresponding Altera component numbers and looked up the price on digikey.com, an electronic component sales site. Digikey is a fast turnaround, any quantity vendor and as such it is expensive. But the reader should examine the figures relative to each other.

Clearly implementing the fully parallel AES scheme has a significant impact on the cost; the hybrid approach minimizes the pin count for the device, but does not save enough to justify AES. It appears that the quasigroup design will utilize less power than the AES implementation. It also appears that if the data throughput is not a primary concern that the quasigroup encryption algorithm has a lower cost.

VI. CONCLUSIONS AND FUTURE WORK

Our objective with the research was to verify whether quasigroup encryption might provide a low-cost solution for embedded systems, relative to AES it shows promise. The Altera FPGAs contain internal memory blocks that can be used in the FPGA design without the penalty of going off-chip. For example, the Cyclone II family provides these in 4K-bit (512 byte) blocks, with the number of blocks dependent on the device. We thus list our obvious future work in the research - to integrate these internal blocks in the FPGA design and check the timing and costs again.

We also note that in work by Good and Benaisa [13] a variety of AES hardware schemes are described, some of which may perform more closely in cost to the quasigroup scheme presented here. It remains to test those implementations that claim the smallest footprint.

Lastly we also note that since the combination of Quartus-II

<table>
<thead>
<tr>
<th>TABLE II: PERFORMANCE COMPARISON FOR DESIGNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocks/128-bit Block</td>
</tr>
<tr>
<td>Quasigroup in Cyclone II</td>
</tr>
<tr>
<td>Quasigroup in Cyclone III</td>
</tr>
<tr>
<td>AES in Cyclone II</td>
</tr>
<tr>
<td>AES in Cyclone III</td>
</tr>
<tr>
<td>Hybrid in Cyclone II</td>
</tr>
<tr>
<td>Hybrid in Cyclone III</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III: COST COMPARISON FOR DESIGNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Selected</td>
</tr>
<tr>
<td>Quasi in Cyclone II</td>
</tr>
<tr>
<td>Quasi in Cyclone III</td>
</tr>
<tr>
<td>AES in Cyclone II</td>
</tr>
<tr>
<td>AES in Cyclone III</td>
</tr>
<tr>
<td>Hybrid in Cyclone II</td>
</tr>
<tr>
<td>Hybrid in Cyclone III</td>
</tr>
</tbody>
</table>
and Modelsim-Altera provides a design simulation with extremely high trustworthiness, we have not found it necessary to actually build anything. In the future we plan on using one of the FPGA development kits to implement a breadboard test setup for the quasigroup encryption.

REFERENCES


