The Effectiveness of Brute Force Attacks on RC4

Nathaniel Couture
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada
1394e@unb.ca

Kenneth B. Kent
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada
ken@unb.ca

Abstract

The security of encryption algorithms depends heavily on the computational infeasibility of exhaustive key-space searches. The RC4 cipher, utilized primarily in the area of data communications, is being used in this paper as a test case for determining the effectiveness of exhaustive key-searches implemented on FPGAs using a Network on Chip (NoC) design architecture. Preliminary results show that a network of ***key-checker*** *units implemented on a Xilinx XC2V1000 FPGA using the Celoxica DK2 design tools can exploit the speed and parallelism of hardware such that the entire key-space of a 40-bit RC4 encryption can be searched in minutes. Furthermore, it has been found that the clock rate of the circuit diminishes as the number of key-checker units increases. Future work is proposed to find a method for predicting an optimal balance between the size of the network (# of key-checker units) and the clock rate in order to maximize performance.*

1. Introduction

Advancements made in e-commerce, wireless internet access, and various other communication technologies have created a tremendous need for secure encryption algorithms. The RC4 cipher, predominantly used in the area of data communications, was developed in 1987 at a time when an Intel 80386 running at 16 MHz was leading edge[1].

The size and sophistication of Field Programmable Gate Arrays (FPGA) and their accompanying Computer Aided Design (CAD) tools are evolving at incredible rates [5], and because of this are more commonly being used to solve computationally intensive problems [6,7,8]. In this paper, a hardware implementation of an RC4 key-search machine, using a network of key-checker units, is used to test the effectiveness of brute force attacks on 40-bit RC4. The objective is to present possibilities for future work on the prediction of the optimal number of checker units to be used in similar hardware designs considering the fact that the clock rates of circuits decrease as the number of checker units increase.

Necessary background information is reviewed in Section 2. The RC4 encryption algorithm is described in Section 3, and the implementation details are presented in Section 4. Results, conclusions and future work are discussed in sections 5 and 6 respectively.

2. Background

This section provides background information for this paper.

2.1. RC4

Ron Rivest of RSA Data Security, Inc developed the RC4 cipher in 1987, the details of which were published in 1996 [11]. RC4 is a public-key encryption system and, as the name suggests, requires the exchange of public keys. Through an authentication handshake process the participants of the communication session use the public keys to generate master keys, which are used to encrypt and decrypt messages transferred during a particular communication session. All messages encrypted with the master keys are considered secure.

2.2. Brute Force Attacks

A brute force attack on encrypted messages, otherwise known as a “known plaintext attack”, consists of decrypting an intercepted message with every possible key and comparing the result to the “known” plaintext. The “known” text is essentially guessed, but is easily deduced from the fact that

[1] RC4 will be used in this paper as a test case for experimenting with exhaustive key-searching.

communication sessions often begin with the same sequence of bytes. For an attack of this kind to be successful, only a small number of "known" bytes are necessary, making the guessing process significantly easier.

2.3. Field Programmable Gate Arrays

FPGAs are two dimensional arrays of uncommitted logic gates grouped into Configurable Logic Blocks (CLBs) with common interconnection resources. Each CLB is capable of implementing a simple logic circuit that consists of up to 4 inputs and 2 outputs. Using a hardware description language (HDL), a custom circuit can be described in a high level language. Similar to the compilation process of software, the HDL code is synthesized, generating a bit file, which essentially maps the described circuit to the CLBs on the FPGA. The bit file is downloaded to the FPGA allowing a developer to quickly program the hardware device into a custom circuit. Taking this approach to implement the RC4 algorithm implies the creation of an instance-specific circuit that implements the RC4 algorithm using a particular pair of ciphertext/plaintext data. The instance-specific approach requires that for each new pair of ciphertext/plaintext data, a new instance of the circuit must be generated. A large portion of the circuit however remains the same between instances and therefore does not need to be modified.

2.4. Network on Chip Architecture

A hardware design solution for this particular problem involves creating several individual components, each designed to perform a specific task. Several components connected together form a network similar to a typical broadband network. The components on a chip can be viewed as a micro-network, communicating with each other using the micro-network stack paradigm which is based on the stack-protocol [1]. The individual components of the RC4 key-search machine use this network architecture as the basis for their communications.

3. RC4 Encryption Algorithm

The RC4 encryption algorithm completes its encryption in 3 steps. The first step involves initializing S and K, both of which are byte arrays containing 256 elements. The initialization of S and K is quite simple: K is repeatedly assigned the values of the key until it is full and S is filled linearly with the sequence of numbers beginning with 0 and ending with 255, which will later be permuted. Pseudo code for this step is shown below (where i and j are integers):

```
S[0..255]= 0 to 255
K[0..255]= key( i mod key_length)
for i = 0 to 255
    j = ( j + S[i] + K[i])
    swap S[i] and S[j]
```

The second step in the encryption process is the pseudo-random byte generation. This requires a small number of simple operations performed on arrays S and K, and is best described by the pseudo code below (where i, j, and t are integers):

```
i = ( i + 1 ) mod 256
j = ( j + S[i] ) mod 256
swap S[i] and S[j]
t = (S[i] + S[j] ) mod 256
random byte = S[t]
```

The third and final step is the creation of the ciphertext, obtained by XORing the pseudo-random byte with the plaintext. For the purpose of this paper we are interested in the decryption process, which similar to encryption, is done by simply XORing the ciphertext with the pseudo-random byte, producing the original plaintext. This decryption process will be the basis for the brute force RC4 cracker.

4. Implementation Details

There are several methods of attempting a brute force attack on RC4; two will be discussed in this paper. The first is a software implementation running on a PC. This approach is scalable and can be extended to a cluster of PCs. The second is a hardware implementation realized on an FPGA using a Network on Chip architecture.

Despite the differences in implementation, their functionality is identical: they apply the decryption algorithm, described above, to decrypt an encrypted message (ciphertext) using every possible key. This means that for a message encrypted with a 40-bit key, this process requires us to check 1.1×10^{12} keys in the worst case, therefore it is essential that the program or device performing the key-search be very efficient and capable of searching a large number of keys per second.

4.1. Software implementation

The RC4 cipher was originally designed for software, therefore this implementation is very

straightforward. The program implements the RC4 algorithm, taking as parameters the ciphertext and plaintext. Starting with the smallest possible key (0x00000) and incrementing by one each iteration, the program eventually reaches the key which properly decrypts the message. This implementation is very scaleable and easily extended to a cluster of PCs by dividing up the key-space among participating computers.

4.2. Hardware Implementation

The implementation was realized on a Xilinx Virtex II *XC2V1000* FPGA using tools from Celoxica [3]. It consisted of two distinct components: the **Key-Checker Unit** and the **Controller** (Figure 1).

The checker-units can be viewed as dedicated integrated circuits, each consisting of a few memory elements, a couple of adders, an XOR gate and a couple of counters. Each checker unit is designed to check a single key independently. Using multiple checkers in a network therefore results in an adjustable level of parallelism easily modified by adding or removing key-checker units from the design. The controller is used to distribute the key space and is also responsible for receiving messages passed to it from the key-checker units indicating the end of the search.

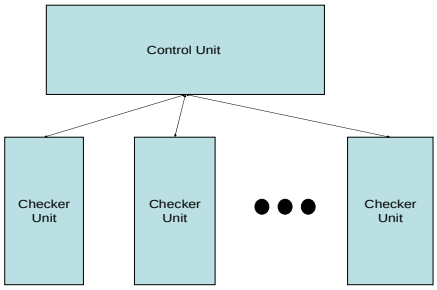

Figure 1: Block Diagram of the Hardware RC4 Key-Search Machine

A key-checker unit contains an implementation of the RC4 decryption algorithm. Each unit has an interface comprised of 2 ports, an input and an output. A single key is sent to the input port and upon completing the decryption process returns the result on the output port. As described in a previous section, the algorithm begins with the initialization of arrays S and K. In terms of hardware, S and K are small memory units. Initialization of S involves assigning the values of S to 0 through 255 sequentially and takes 256 clock cycles due to the restrictions on memory access. Similarly, K is filled with the bytes of the key. As an optimization over the proposed algorithm, the length of K was reduced to the size of the key (40 bytes in this case) and read using modulo addressing rather than repeatedly filling it with redundant data. This was done in order to reduce the number of clock cycles required to check a single key. With S and K initialized, the next step is to permute S based on the values contained in K. Considering that the swap alone takes 4 clock cycles, one to read S[i], one to read S[j] and a clock cycle each to rewrite each of them to the opposite locations, many clock cycles are used simply to perform the initializations. Overall, the whole process of checking a single key takes approximately 1300 clock cycles; 516 clock cycles for initialization of S and K, 770 cycles for the pseudo-random permutation of S and 7 cycles for the generation of the pseudo-random bytes.

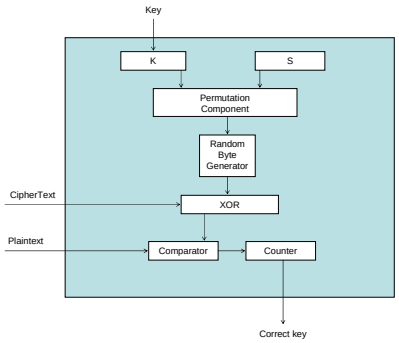

Figure 2: Abstract View of the Key-Checker's Internal Structure

During the design, the number of clock cycles in each step of the decryption must be considered, as it directly affects the performance of the key-search machine. A larger circuit, although it may contain more key-checker-units, will likely affect the clock rate negatively and should be monitored such that an optimal solution can be implemented. Memory dependencies in the hardware also cause certain restrictions that were not issues in the software implementation; this includes limited access to the arrays S and K to one address per clock cycle [3].

Despite the slow clock rates attained by FPGAs in general and the considerably large number of clock cycles required to verify a single key, good performance is still achievable by scaling the implementation [8]. Implementing multiple key-checker units and dividing the key-space equally among them is trivial and makes this approach very effective and at the same time very cost-effective [8]. Increasing both the number of key-checker units and the clock rate increases the performance

over a single key-checker unit dramatically. Though it must be noted that increasing the number of checker units typically results in an overall decrease in clock rate. This is due to the fact that a signal takes more time to propagate through a longer wire.

5. Results

Both implementations were tested using 40 bit keys. 40 bits were chosen due to the fact that it is the maximum key size allowable for an application exported from North America using RC4 [11]. The software implementation was executed on different PCs ranging from a Pentium II 266 MHz to a Pentium IV 2.4 GHz.

The results of the hardware cracker are still preliminary and were extrapolated from the results obtained for an implementation using only a single key-checker unit running at 10 MHz. The implementation did however support multiple key checker units, as well as slightly higher clock rates, roughly 14MHz according to the synthesis tools. It is expected that this can be improved through design optimizations. A summary of the results can be seen in Table 1. Using simple analysis, we quickly discover that on average, only half of the key-space must be searched in order to find the correct key. The estimated times were therefore based on how long it would take, on average, to find the correct key.

Implem.	# of Unts	Keys / second	# of FFs	Estimated time
FPGA Running at 10MHz	1	8,000	2300	800 days
	5	40,000	11000	165 days
	10	80,000	~22000	82 days
	25	200,000	~50000	33 days
	100	800,000	~220000	8 days
	500	4,000,000	~1000000	40 hours
Pentium II 266MHz	1 PC	22,500	NA	310 days
Pentium III 500MHz	1 PC	225,000	NA	28 days
Pentium IV 2.4GHz	1 PC	1,000,000	NA	6.5 days

Table 1: Performance Results

6. Conclusions and Future Work

Examining the results, it becomes quite apparent that even a moderate scaling of these implementations make RC4 quite vulnerable to brute-force attacks. A very large FPGA, with a network of 500 checker-units could likely crack a 40-bit encryption of RC4 within minutes or perhaps even seconds. Yet, the maximum allowable key size for exported applications using RC4 is 40 bits. Wire Equivalent Protection (WEP) uses an even smaller key, at 24 bits. Therefore it isn't difficult to see that RC4 does not provide adequate protection**.**

Although FPGAs are currently large enough to support big networks of checker-units, issues concerning congestion around the controller arise, and make it difficult for the synthesis tools to route the connections to and from each checker-unit, thus reducing the potential size of these networks to much less than what is available on the chip. Also, as the size of the circuit increases, the clock rate slows down. It therefore becomes an issue to find an optimal number of checker-units that will produce the best solution in terms of performance.

References

[1] L. Benini and G. D. Micheli, *Networks on Chip: A New Paradigm for Systems on Chip Design*, Design Automation and Test in Europe 2002, March 2002.

[2] L. Benini and G.D. Micheli, *Networks on Chip: A New SoC Paradigm*, IEEE Computer, Jan. 2002, p 70-78.

[3] Celoxica, HandelC Reference Manual.

[4] D.D. Gajski, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[5] D. Edenfeld, A.B. Kahng, M. Rodgers and Y. Zorian, *2003 Technology Roadmap for Semiconductors*, IEEE Computer, Jan. 2004, pp 47-56.

[6] K.B. Kent and J.C. Rice, *Using Instance-Specific Circuits to Compute Autocorrelation Coefficients*, First Northeast Workshop on Circuits and Systems 2003, June 2003, pp. 61-64.

[7] K.B. Kent and M. Serra, *Using FPGAs to Solve the Hamiltonian Cycle Problem*, International Symposium on Circuits and Systems 2003, 2003, pp. III-228 - III-231.

[8] P. Kundarewich, S. Wilton and A.J. Hu, *A CPLD-based RC-4 Cracking System*. Canadian Conference on Electrical and Computer Engineering 1999, 1999.

[9] W. Patterson, *Mathematical Cryptology for Computer Scientists and Mathematicians*. Rowman & Littlefield, 1987, pp 6-15.

[10] RSA Labratories. RSA Security. http://www.rsasecurity.com/ Last updated [2003], Accessed [October 17, 2003].

[11] B.Schneier, *Applied Cryptography*. John Wiley & Sons, Second Edition, 1996, pp 397-400.