Encryption for Mobile Computing

Erik Olson and Woojin Yu

Abstract

This paper presents a comprehensive survey of encryption algorithms and their usage in mobile computing, specifically in the PalmPilot™, which uses Motorola's DragonBall-EZ processor. This processor is architecturally very similar to the 68K processor, which does not provide the power and versatility of current processors. Since it is a very limited processor, it is critical for users to choose the appropriate algorithm. The encryption algorithms surveyed in this paper were RC5™, RC6™, Twofish, and Triple-DES. The analysis of each algorithm revealed that there is a common set of instructions, such as rotational shifts, exclusive-or, and data moves, which are widely used among all the algorithms. As transmission speeds increase, it has become apparent that encryption and decryption in palm devices will become a bottleneck, and that improvements must be made at an instructional or hardware level for effective communication.

1. Introduction

With the explosive growth of networked computing in recent years, issues of data security have become a primary concern. In the past, encryption algorithms such as DES have been sufficient to handle most security needs, but it is increasingly apparent that the time of DES is quickly approaching its end. With the limitations of DES's 56-bit key and the advent of faster computers, DES can no longer be considered a secure algorithm. In recent years, Triple-DES has taken the place of DES in an attempt to create stronger security without compromising currently accepted encryption standards. For many purposes, however, Triple-DES is simply too slow.

With the eminent demise of DES, the National Institute of Standards and Technology (NIST) has issued a request for submissions of stronger encryption algorithms to replace the aging DES. This Advanced Encryption Standard (AES) must conform to several strict requirements: it must be a block cypher with longer key length, larger block size, fast speed, and greater flexibility. After several rounds of submissions and eliminations, the NIST has narrowed the applicant pool down to five finalists.

While such algorithms are simple and efficient to implement on high-performance microprocessors found in current desktop and laptop computers, such may not be the case for the smaller, less powerful processors found in current palm computing devices. With the ever-expanding market for palm computers, and the inevitable networking of palm devices, it appears that encryption on limited processors may warrant closer investigation.

The current market leader in palm devices is clearly the PalmPilot™, holding over 70% of the market share of palm computing. The PalmPilot™ series of devices is built around the Motorola DragonBall-EZ 68K core microprocessor, providing a low power, but severely limited processor.

The primary objective of this study is to analyze the efficiency of various encryption algorithms on the DragonBall-EZ processor in an attempt to determine the ability of the PalmPilot™ series of palm computers to perform encryption at speeds required for effective communication.

2. Algorithms

2.1 DES and Triple-DES

The initial attempts at modern data security came in the form of the Data Encryption Algorithm (DEA), developed in the 1970's and adopted (with some modification) by the United States Government as the Data Encryption Standard (DES) [8]. The terms of the acceptance of DES as the standard require a review of the algorithm every five years. DES was reviewed and re-certified in 1983, and again in 1988, despite concerns that DES was becoming more vulnerable to attacks. Again in 1992, due to the lack of alternative encryption algorithms, DES was re-certified for another five years. Thus despite great concerns over the security of DES in the modern age of high speed computing, it has remained the standard for data encryption.

DES is a block cipher, with a 64-bit block size and a 56-bit key. DES consists of a 16-round series of substitutions and permutations. In each round, data and key bits are shifted, permuted, XORed, and sent through 8 S-boxes, a set of lookup tables that are essential to the DES algorithm. Decryption is essentially the same process, performed in reverse.

In recent years, the worries about DES security have spawned a series of DES implementations that are variations of the stock DES algorithm. The most commonly used variation is Triple-DES. Triple-DES is simply triple encryption using the DES algorithm. A 64-bit block is encrypted once with DES. The

resulting cyphertext is then encrypted again with DES, and that resulting cyphertext is encrypted a third time, again using the DES algorithm. Although this series of encryption offers greater security, it is also three times slower than the single DES implementation.

2.2 RC5

RC5 is a symmetric encryption algorithm whose plaintext and ciphertext are fixed-length bit sequences [6]. RC5's biggest advantage is its simplicity: encryption and decryption can each be implemented with only five lines of C code. In addition, due to the simplicity of RC5, it has extremely low memory requirements. RC5 also offers flexibility and versatility by giving users the option to change the number of rounds performed, key size, and block size. By adjusting these options users can manipulate the trade-off between speed and security. One unique aspect of RC5 is the usage of data-dependent rotational shifts. Data-dependent rotational shifts involve manipulating bits with the shift amount determined by a block of data, instead of a fixed integer value. Each encryption and decryption function accepts two blocks (32 bits for each block) of data, either as ciphertext or plaintext. When the data manipulation occurs, the function outputs two blocks of ciphertext or plaintext, depending on whether the function encrypts or decrypts. Just like other algorithms, RC5 has a set-up process, and the set-up time is dominated by the creation of an expanded key table whose elements are used as a second argument in XOR operations during encryption and decryption. A thirty-two bit word, sixteen round, and four-byte sized key configuration was used in this analysis.

```
Encryption
Input:    Plaintext in two 32-bit variables A and B
          Number r of rounds
          Expanded key table S
Output: Ciphertext stored in A,B

A=A+S[0]
B=B+S[1]
For i=1 to r
          A=((A XOR B) <<< B)+S[2*i]
          B=((B XOR A) <<< A)+S[2*i+1]

Decryption
Input:    Ciphertext in two 32-bit variables A and B
          Number r of rounds
          Expanded key table S
Output: Plaintext stored in A,B

For i=r to 1
          B=((B-S[2*i+1]) >>>A) XOR A
          A=((A-S[2*i]>>>B) XOR B
B=B-S[1];
A=A-S[0];

>>> = rotational right shift.
<<< = rotational left shift.
```

2.3 RC6

RC6 was designed to meet the requirements of the Advanced Encryption Standard (AES), yet retains the simplicity and the speed of RC5 [5]. It makes essential use of data-dependent rotations in encryption and decryption, but modifications were made to use four rather than two 32-bit data blocks to meet the AES 128-bit block size requirement. RC6 also includes integer multiplication as an additional primitive operation. It still makes use of the expanded key table, which is created at the set-up time, but the size of the table is twice the size of the table in RC5 because the algorithm accepts four blocks.

```
Encryption
Input:    Plaintext in four 32-bit variables A,B,C,D
          Number r of rounds
          Expanded key table S[0,..,2r+3]
Output:   Ciphertext stored in A,B,C, and D

B=B+S[0]
D=D+S[1]
For i=1 to r
          t=(B*(2B+1)) <<< lg 32
          u=(D*(2D+1)) <<< lg 32
          A=((A XOR t) <<< u)+ S[2i]
          C=((C XOR u) <<< t) + S[2i+1]
          (A,B,C,D)=(B,C,D,A)
A=A+S[2r+2]
C=C+S[2r+3]

Decryption
Input:    Ciphertext in four 32-bit variables A,B,C,D
          Number r of rounds
          Expanded key table S[0,..2r+3]
Output:   Plaintext stored in A,B,C, and D

C=C-S[2r+3]
A=A-S[2r+2]
For I=r to 1
          (A,B,C,D)=(D,A,B,C)
          u=(D*(2D+1)) <<< lg 32
          t=(B*(2B+1)) <<< lg 32
          C=((C-S[2i+1])>>>t) XOR u
          A=((A-S[2i])>>>u) XOR t
D=D-S[1]
B=B-S[0]

lg operation = log base 2 operation
<<< = rotational shift left
>>> = rotational shift right
```

2.4 Twofish

Twofish is one of the AES finalists, and was designed to meet the NIST's design specifications for AES [7]:

- 128-bit symmetric block cypher

- 128-bit, 192-bit, and 256-bit key lengths
- No weak keys
- Efficient, flexible, and simple.

Furthermore, the designers of Twofish have imposed upon themselves the following design criteria:
- The algorithm must not contain any operations that make it inefficient on 8-bit and 16-bit microprocessors
- The algorithm must be implementable on an 8-bit microprocessor with only 64 bytes of RAM

These additional criteria make Twofish ideal for analysis on low power microprocessors, such as the DragonBall-EZ.

Much like DES, Twofish uses a 16-round encryption process, each round consisting of bit permutations, XORs, logical shifts, and S-box lookups. In addition, each round of Twofish contains matrix multiplication, and Pseudo-Hadamard Transforms (PHT), a simple calculation defined as:

$$a' = a + b \bmod 232$$
$$b' = a + 2b \bmod 232$$

Moreover, each S-box is key dependent, and thus must be recalculated for each round, using a series of table lookups and mathematical calculations.

Despite the requirement for simplicity, it is apparent that the current implementation of Twofish is nonetheless rather complex, requiring a number of calculations, table lookups (memory accesses), and logical shifts for each round of encryption.

3. Processor

Palm Pilot™ uses Motorola's DragonBall-EZ (MC68EZ328) microprocessor. It has a 68K core that implements the standard Motorola 68K instruction set architecture [2]. From an architectural point of view, the 68K processor is extremely outdated and its capabilities are very limited, but its acceptance in mobile computing is not surprising because most mobile devices do not need to perform complex tasks.

These are some of the highlights for DragonBall-EZ processor [4]:
- 2.7 MIPS Performance at 16.58 MHz processor clock
- 32-bit internal address bus
- 24-bit external address bus capable of addressing maximum 4 * 16MB blocks with chip selects CSA, CSB, and 4 * 4 MB blocks with chip selects CSC, CSD
- 16 General-Purpose 32-bit Registers
  - o Data Registers (D7 - D0)
  - o Address Registers (A7 - A0)
- 14 addressing modes and five main data types

4. Method

The choice of processor and algorithms were of primary importance to the investigation of encryption with palm devices. The processor was chosen based on current trends in the palm computing market. With the PalmPilot™ being the most popular palm computer, dominating over 70% of the palm computing market, and with the advent of wireless communications for the PalmVII™ and a wireless modem for the PalmV™, it seems logical to examine Palm's processor, the DragonBall-EZ.

The choice of algorithms proved to be more difficult. There are thousands of encryption algorithms, with source code available for study. The decision to study DES and Triple-DES was based on the status of DES as the currently accepted standard for data encryption, and its nearly ubiquitous use in encryption-offering software (and hardware). RC5 was chosen as a contemporary counterpart to DES. It is a thoroughly analyzed and accepted algorithm, offering powerful encryption with a small key size.

With the eminent demise of DES, to be replaced by the AES suite of algorithms, it seemed only natural to consider the future of encryption technology: the AES. Of the many finalists, RC6 and Twofish were chosen for study based on their similarities to the previously chosen algorithms, RC5 and DES. RC6 is based heavily on RC5, while Twofish shares many of the S-box and permutation methods found in DES.

Source code in C was readily available for each algorithm, found in books and on the Internet. In most cases, only minor modification was necessary for use on the 68K series of processors, mostly due to Little-Endian/Big-Endian issues, as well as some other compiler and architectural concerns.

The source was compiled using Metrowerks CodeWarrior™, and disassembled for analysis. Through hand analysis of the assembly code, loops were identified and marked with a tag specifying the number of times the loop is iterated. Using a cycle counting program based on published cycle times for various instructions and addressing modes [3], the number of total cycles, as well as the number of cycles spent executing every instance of each instruction, was determined for the encryption of a single block of plaintext.

A similar analysis was performed on the setup and key generation code to determine setup overhead.

Closer inspection of the assembly code revealed several places that could benefit from hand-coded optimization. The code for RC5 and RC6 was optimized in this manner and re-analyzed using the cycle counter.

Graphing the results revealed bottlenecks and suggested areas of concentration for further investigation and optimization.

5. Results

Each encryption algorithm was divided into two portions: set-up functions that initialize the key elements that would be used in encryption/decryption and core functions, which repeatedly perform operations on data blocks. Set-up and core functions were analyzed separately because the set-up function is executed only once before encryption or decryption, while core functions are executed continuously until there is no more data to encrypt/decrypt. Figure 1 plots percentages of total cycle time spent executing each instruction type in the set-up code of each algorithm. Although dependent on encryption algorithm, it is evident that move and shift (LSL and LSR) instructions are dominant in every case. Figure 2 represents the percentages of total cycle time spent executing each instruction type in the core functions of each algorithm. A similar trend is observed, with move and shift instructions taking up an even greater percentage of cycle time than observed in the setup code.

6. Analysis

The most apparent trend found in all the algorithms under investigation is the high cycle count for the MOVE instructions. This can be attributed to the DragonBall-EZ hardware and instruction-set architecture. The MOVE type instructions are used for memory accesses as well as register-to-register operations. Due to the limited number of registers in the 68K core, the DragonBall-EZ must make a large number of memory access requests, and, since there is no cache, each request takes several cycles to complete.

The impact of reducing the time required for a MOVE instruction can have a significant effect on the overall efficiency of all the encryption algorithms. Unfortunately, the cycle time required for a MOVE instruction is extremely difficult to alter. The addition of a cache may decrease the impact of MOVE delays by providing faster memory accesses, but the cost in die area as well as implementation of caching hardware may be too great to justify the use of a cache.

Similarly, reducing the *number* of move instructions may increase the efficiency of all the algorithms. This can be accomplished in hardware by adding more registers, eliminating the need to swap data from registers to memory as often. In software, clever coding may reduce the required number of MOVE instructions as well. This software optimization may appear to be the cheapest solution, but it is extremely limited in its ability to increase speedup, since only a very limited number of MOVEs may actually be eliminated.

A second trend seems to appear as a correlation between the types of algorithms. The computation-based algorithms, such as RC5 and RC6, appear to be dominated by MOVEs, and logical shifts, LSR and LSL. The S-box based algorithms, such as DES and Twofish, seem to be dominated by MOVEs, ORs, shifts, and branches.

One factor that contributed to the dominance of moves and logical shifts in the RC5 and RC6 algorithms sprang from the declaration of rotational shifts in the C source code. RC5 and RC6 are highly dependant upon rotational shifts for the implementation of the encryption algorithm. These rotational shifts were declared in the C source as a logical shift left OR'ed with a logical shift right. The DragonBall-EZ processor's implementation of logical shifts has a variable cycle time, dependant on n, where n is the number of places shifted. However, the DragonBall-EZ has its own rotational shift instruction, requiring the same amount of time as a single logical shift.

Analysis of the assembly code revealed a strict translation of the C source rotational shift, using two logical shifts and OR'ing the results. Replacing this code with the rotational shift instruction not only eliminated the need for two logical shifts, but also eliminated the required overhead involving costly MOVE instructions.

Further optimization of the rotational shifts can be done in hardware by replacing the variable cycle rotational shift operation with a fixed cycle shifter that can be implemented easily using multi-layered multiplexers. Based on the current rotational shift cycle count, $8 + 2n$, where n is the number of places to shift, and choosing a conservative cycle time (8 cycles, equal to that of the current shift setup time) a speedup of 2 times can be achieved over the initial, un-optimized source.

Figure 3 Example of rotational right shifter

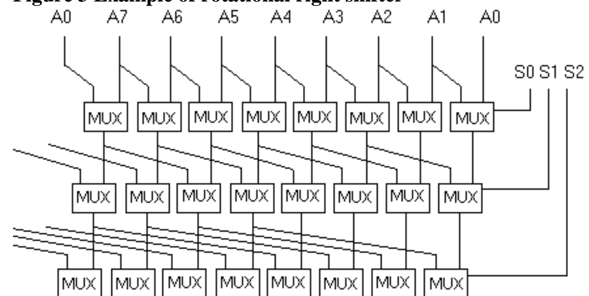


Figure 1

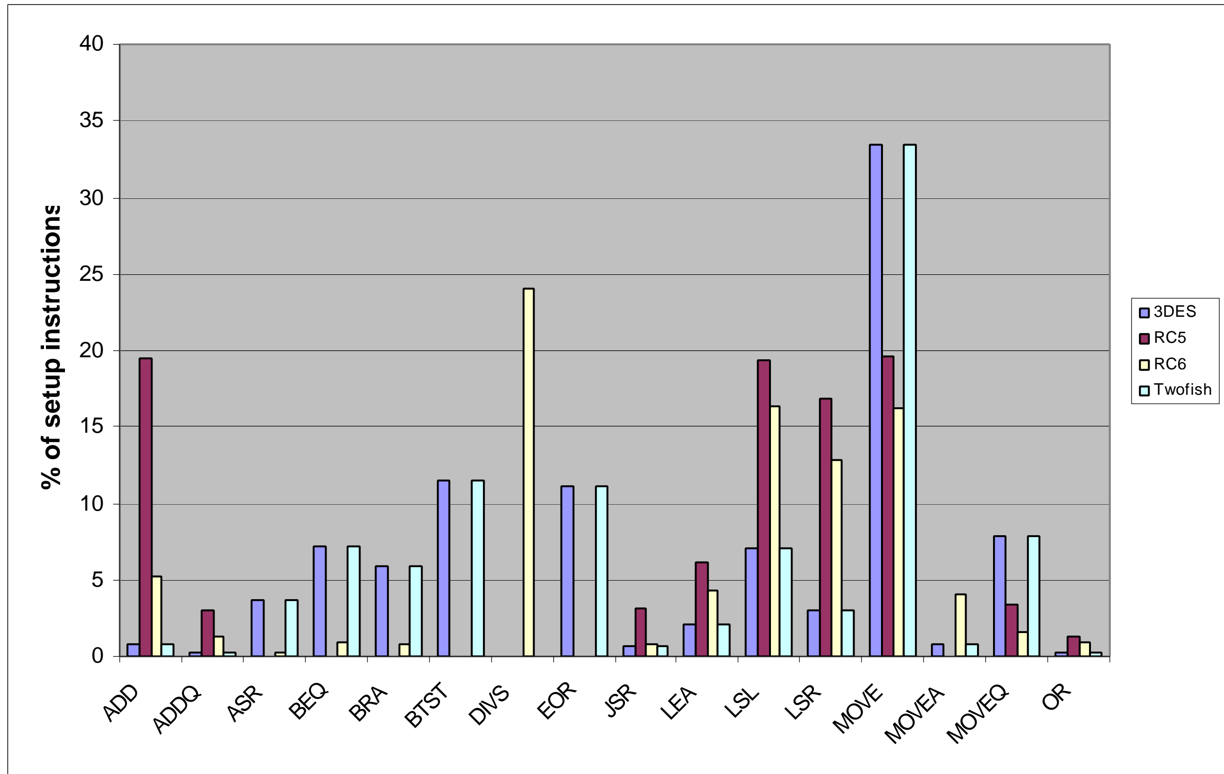
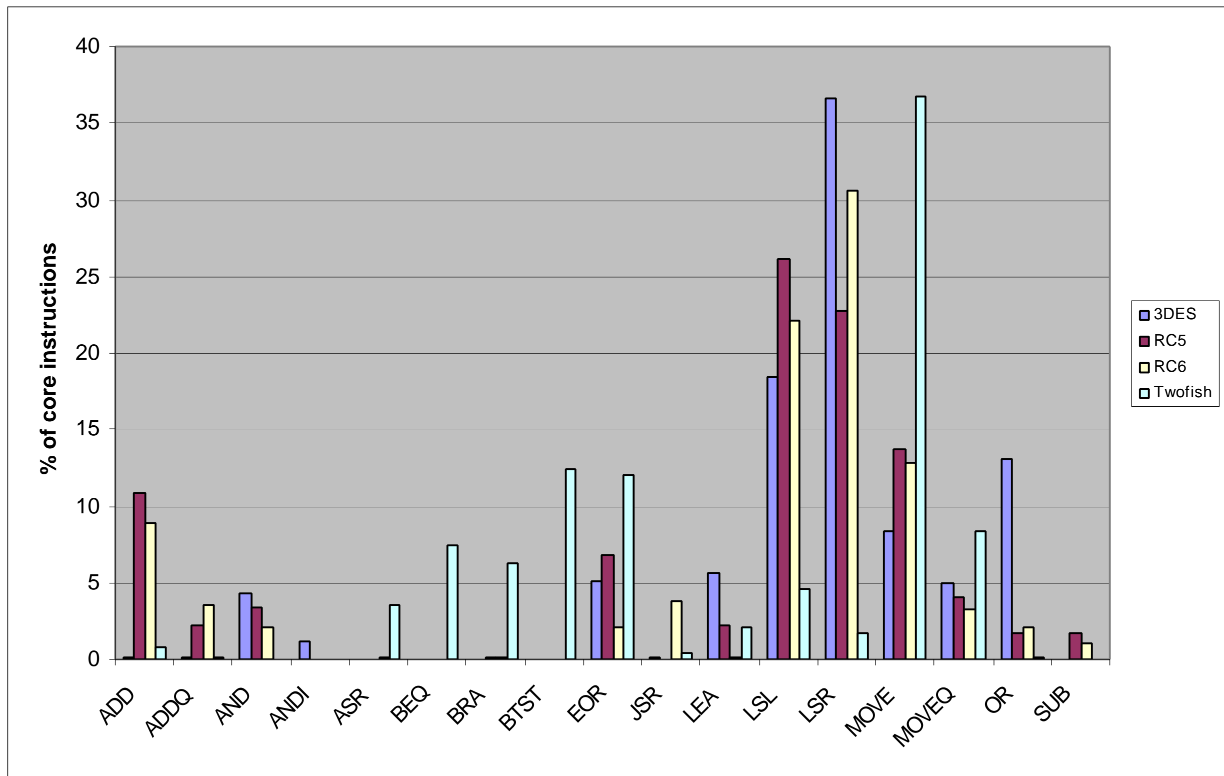


Figure 2



7. Conclusion

Analysis of the total cycle time required for encryption using the various algorithms reveals an interesting trend. The calculation-based algorithms, RC5 and RC6, are much faster than the S-box based algorithms, DES and Twofish, on the DragonBall-EZ processor. Taking only the encryption core into account, and ignoring all Operating System and transmission overhead, the following values were obtained:

• DES	6.1 Kbps
• RC5	121 Kbps
	272 Kbps[1]
• RC6	87 Kbps
	187 Kbps[2]
•Twofish	12.8 Kbps

Even at the relatively slow current transmission speeds of the PalmVII™ wireless modem (14.4Kbps), it is clear that DES and Twofish are simply too slow. RC5 and RC6, however, appear to be sufficiently fast for current transmission speeds. Taking the worst-case scenario using un-optimized RC6 for encryption, the algorithm is still fast enough for transmission at the current speed of 14.4Kbps, even if transmission and OS overhead consume over 83% of the cycle time. However, the future of wireless communication is bound to improve dramatically in the coming years. As transmission rates increase, it is clear that none of the algorithms in this study can be implemented fast enough on a limited processor such as the DragonBall-EZ. Unless drastic improvements are made to the DragonBall-EZ processor itself, it is clear that there is a need for either less complex encryption algorithms designed specifically for use in low power processors, or the integration of specialized, low power encryption hardware.

8. References

1. Motorola. *Motorola M68000 Programmer's Reference.* 1992
2. Motorola*.* *MC68EZ328 Integrated Processor User's Manual.* 1998
3. Motorola. *M68000 8-/16-/32-Bit Microprocessors User's Manual Ninth Edition.* 1993
4. Motorola. *MC68EZ328 Product Brief.* 1998
5. Ronald Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Lin. *The RC6 Block Cipher.*
6. Ronald Rivest. *TheRC5 Encryption Algorithm.*
7. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson. *Twofish: A 128-bit Block Cipher*. 1998
8. Bruce Schineier. *Applied Cryptograph Second Edition*. John Wiley & Sons, Inc. 1996

[1] Optimized using theoretical 8-cycle hardware rotational shifter

[2] Optimized using theoretical 8-cycle hardware rotational shifter