

Case Study: Compiler Comparison for an Embedded Cryptographical Application

E. Barteska², C. Paar¹, J. Pelzl¹, V. Wittelsberger², T. Wollinger¹

presented at "The 2004 International Conference on Embedded Systems and Applications - ESA", June 21-24, 2004, Las Vegas, USA

¹ Communication Security Group (COSY)
Ruhr-Universität Bochum, Germany
Email: {cpaar, pelzl, wollinger}@crypto.rub.de

² HSP Embedded Tools GmbH
Albrecht-Thaer-Straße 22
48147 Münster, Germany

Abstract. In the recent years, communication with embedded computers has become omnipresent. Mobile phones and personal digital assistants (PDAs) are a part of our business and personnel everyday life. Protecting contents with cryptographic primitives is inevitable and demands for highly efficient cryptographic applications in the sense of time and power consumption. Some cryptosystems allow for high efficiency assuming a thorough done implementation. Optimizing implementations on embedded devices often implies the use of the processor language (assembly). However, current compilers for the C programming language can often also produce highly optimized code regarding code size and/ or execution time. In this contribution, we analyze the generated code of different compilers for embedded processors. The time and space consumption of the code — a modern cryptographic implementation, namely a hyperelliptic curve cryptosystem — determines We provide an accurate comparison of several important compilers (DIAB 5.1.2, GNU 3.3, MetaWare 4.5, ARMCC 2.0 and CodeWarrior 2.0) for embedded processors.

Keywords: embedded processors, cryptographical applications, compiler comparison, encryption, hyperelliptic curve cryptosystems.

1 Introduction

It is widely recognized that data security will play a central role in the majority of future IT systems. Many of these future IT applications will be realized as embedded systems. A lot of those applications rely heavily on security mechanisms such as security for wireless phones, faxes, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products as well as digital cinemas. Note that a large share of those embedded applications will be wireless. Wireless applications can be easily eavesdropped, which makes the communication channel especially vulnerable and the need for security even more obvious. In addition, authentication technologies are desired in order to prevent the threat of generating personal profiles when using the stated application. This merging of communications and computation functionality requires data processing in real time, and embedded systems have shown to provide appropriate solutions for many applications, e.g. cellular phones.

All modern security protocols, such as IPsec, SSL, TLS use symmetric-key algorithms as well as public-key (PK) algorithms. Public-key cryptosystems solve in a very elegant way the key distribution problem of symmetric-key schemes. This algorithms are chosen for key establishment and authentication through digital signatures, and then a symmetric-key algorithm is chosen to encrypt the communications and the data transfer, achieving in this way high throughput rates. Public-key cryptography is based on the idea of separating the key used to encrypt a message from the one used

to decrypt it. Anyone that wants to send a message to party A can encrypt that message using A 's *public key* but only A can decrypt the message using her *private key*.

In general, one can divide practical public-key algorithms into three families: algorithms based on the *integer factorization problem*: (e.g. RSA), algorithms based on the *discrete logarithm problem*: (e.g. Digital Signature Algorithm (DSA)), and algorithms based on *Elliptic Curves* (EC) and *Hyperelliptic Curves* (HEC). PK systems have a major disadvantage when compared to private-key schemes. PK algorithms are very arithmetic intensive and — if not properly implemented or if the underlying processor has a poor integer arithmetic performance — this can lead to a poor system performance. Even when properly implemented, all PK schemes proposed to date are several orders of magnitude slower than the best known private-key schemes. Hence, in practice, cryptographic systems are a mixture of symmetric-key and public-key cryptosystems.

It is important to point out that hyperelliptic curve cryptosystems (HECC) seem to be specially promising for the use in embedded environments where memory and speed is constrained. The suitability for constrained systems results from the *short* operand sizes of HECC compared to other public key schemes, e.g. RSA or DL based systems. It is widely accepted that for cryptographic applications based HEC the necessary group order is of size at least $\approx 2^{160}$. Thus, for HECC over \mathbb{F}_q we will need at least $g \cdot \log_2 q \approx 2^{160}$, where g is the genus of the curve. Hence, one needs 40-bits to 80-bit long operands to compute the group operations for these curves. In the case of elliptic curve cryptosystem we have to work with operand lengths of approximately 160 bits. Whereas in the case of RSA, the operands will be approximately 1024 bits in order to achieve the same security.

Providing highly arithmetic-intensive public-key cryptographic primitives in an embedded environment is often difficult due to the computational, memory and power constraints. Due to the time constraints while developing software products, programmers are not able to write assembly code resulting in a higher performance. Hence, one is very depended on the compilers transforming the high level programming language into efficient assembly instructions. With challenging performance goals, cost constraints, and short product cycles, developers increasingly rely on a compiler's intimate knowledge of a processor's instruction set and behavior to produce optimal code.

Our contribution: We implemented one of the most promising public-key algorithm, namely a hyperelliptic curve cryptosystem in the C programming language. The code was compiled for a typical embedded processor, namely the ARM7TDMI (Samsung SNDS100). The generated code of five popular state-of-the-art ARM compilers — DIAB 5.1.2, GNU 3.3, MetaWare 4.5, ARMCC 2.0 and CodeWarrior 2.0 — was compared in the face of time and space consumption. To provide an fair comparison we applied different optimization settings for the compilers. We investigated the resulting code size and the execution time.

The remainder of the paper is organized as follows. Section 2 provides a brief insight to the mathematics and previous implementations of hyperelliptic curve cryptosystems. Section 3 introduces to the processor and different compilers. In Section 4, we present the results and will end this contribution with a discussion of our results and some conclusions in Section 5.

2 Hyperelliptic Curve Cryptosystem

2.1 Mathematical Background

Let \mathbb{F} be a finite field, and let $\overline{\mathbb{F}}$ be the algebraic closure of \mathbb{F} . A hyperelliptic curve C of genus $g \geq 1$ over \mathbb{F} is the set of solutions $(x, y) \in \mathbb{F} \times \mathbb{F}$ to the equation

$$C : y^2 + h(x)y = f(x)$$

The polynomial $h(x) \in \mathbb{F}[x]$ is of degree at most g and $f(x) \in \mathbb{F}[x]$ is a monic polynomial of degree $2g + 1$. For odd characteristic it suffices to let $h(x) = 0$ and to have $f(x)$ square free. Such a curve is

said to be non-singular if there are no pairs $(x, y) \in \overline{\mathbb{F}} \times \overline{\mathbb{F}}$ which simultaneously satisfy the equation of the curve C and the partial differential equations $2v + h(x) = 0$ and $h'(x)v - f'(x) = 0$.

If we want to define the Jacobian over \mathbb{F} , denoted by $\mathbb{J}_C(\mathbb{F})$, we say that a divisor $D = \sum m_i P_i$ is defined over \mathbb{F} if $D^\sigma = \sum m_i P_i^\sigma$ is equal to D for all automorphisms σ of $\overline{\mathbb{F}}$ over \mathbb{F} [MWZ98].

Each element of the Jacobian can be represented uniquely by a reduced divisor [Ful69, Can87]. This divisor can be represented as a pair of polynomials $u(x)$ and $v(x)$ with $\deg v(x) < \deg u(x) \leq g$, with $u(x)$ dividing $y^2 + h(x)y - f(x)$ and where the coefficients of $u(x)$ and $v(x)$ are elements of \mathbb{F} [Mum84, page 3.17]. In the remainder of this paper, a divisor D represented by polynomials will be denoted by $\text{div}(u, v)$. Cantor's algorithm describes the group addition of two divisors on $\mathbb{J}_C(\mathbb{F})$ [Can87]. In 2000, Harley proposed the first explicit formulae for a group addition and a group doubling of divisors on $\mathbb{J}_C(\mathbb{F})$ [GH00]. In this contribution we used the explicit formulae introduced in [PWP03].

2.2 Previous Implementation of Hyperelliptic Curve Cryptosystem

Koblitz's idea to use HEC for cryptographic applications has been analyzed and implemented on general purpose processors [Kri97, SS98, SSI98, Eng99, Sma99, SS00, Pel02] and on more hardware oriented implementations on FPGAs [Wol01, WP02, BCLW02, Cla02, Cla03, EMY04].

In [Ngu02] the author did the first implementation of HECC on an embedded processor namely using the FrameXE smart card coprocessor. This work was followed by [Pel02, PWP03, PWGP03, PWP04b, PWP04a]. In this contributions, the authors presented improvements for the cryptographic algorithm based on hyperelliptic curves and implementations on ARM processor.

The results presented in [WPW⁺04] appear to be the first thorough comparison of ECC and HECC on a wide range of important embedded platforms, namely ARM, ColdFire, and PowerPC. The best timings for the scalar multiplication for HEC cryptosystems could be achieved on the PowerPC running at 50MHz, resulting in 117 and 84.9 milliseconds for genus 2 and 3, respectively. The scalar multiplication for ECC could be performed fastest on the same platform in 106.3 ms. Additionally, the authors showed that the instruction cache on the PowerPC had a fundamental influence regarding the speed of one scalar multiplication. The time needed to perform one scalar multiplication can be decreased by almost a factor of 8 when using instruction as well as data cache. Further speed up by 50% of the HEC scalar multiplication could be achieved by focusing on a fixed underlying field and curve.

3 Processor and Compilers Used

As testing platform an ARM7TDMI (50MHz) was used. The 16/32-bit ARM7TDMI RISC core is a low-power, general purpose, microprocessor macro-cell that was developed for use in application-specific and custom-specific integrated circuits. It is widely used for embedded applications such as mobile phones and network elements. We choose the Samsung SNDS100 platform (KS32C50100 core) which offers a configurable 8-Kbyte unified cache/SRAM and Ethernet controller.

The code for the processor was created with five different compilers. Since their first appearance, compilers changed from simple code translators to powerful tools. Several optimization techniques have been implemented to translate C/C++ into highly efficient machine code for the huge variety of today's processors. Generating optimal code is a challenging task and highly processor dependent.

The optimization and generation of code by a compiler can be divided into five stages:

1. Global optimization: The global optimizer performs a wide range of general optimizations and a high-level program analysis. Usually, this type of optimizations are database-driven and, thus, can be matched to a particular architecture (e.g. to available registers of a particular processor).

2. Code selection: The same database as for the global optimizer is used to identify the optimal assembly language instructions for the implementation of the intermediate language operations. In this process, a certain number of registers is reserved for temporary variables.
3. Code generation: Ascertain register usage in such a way that unused registers for temporary variables are utilized for variables that would otherwise have to be fetched from memory. All dependencies between instructions become clear by fixating the register usage.
4. Further optimization: Obvious eliminations of code for clearly inefficient sequences. Earlier made failures in the optimization are corrected.
5. Instruction scheduling: The final step of code generation is performed after all other optimization steps when all interdependencies of data are known.

We provide a short snapshot of the five compilers used for the comparison.

DIAB 5.1.2: The DIAB C/C++ compiler from Windriver targets many popular embedded processors including the ARM microprocessor. It parses the source code into a common token-based representation of the source. Optimizations are applied to this intermediate language. The optimized code is then converted to machine dependent code, supported by table-lookups.

Version 5.1 is compatible with the latest ANSI C++ specs (ISO/IEC 14882:1998(E) C++ standard) and ANSI C specs (X3.159-1989).

GNU 3.3: Since the first release of the freely available Gnu C++ Compiler (GCC) in 1987, it has become a powerful tool and creates code for almost all processor types available. It runs on a huge variety of operating systems as a native or cross compiler. Several stages of the optimization process result in a highly efficient code.

MetaWare 4.5: ARC International's MetaWare C/C++ Compiler offers complete control over the compiler's language enforcement, code generation and optimization. The compiler generates code for the most popular embedded processors ARM, MIPS, PowerPC and XScale. It supports strict ANSI C standard and allows for specific optimizations such as loop unrollment, zero delay loops and jump optimization.

ARMCC 2.0: The ARM C compiler is based on Codemist Ltd's multi-target, multilanguage compiler suite (also known as the Norcroft C compiler). By default the ARM C compiler compiles ANSI C (X3 J11/90-013) for the ARM microprocessor.

CodeWarrior 2.0: Metroworks compiler CodeWarrior is available for almost all modern embedded processors including the ARM. It supports the ANSI/ISO C/C++ standards.

4 Results

In this section we present our results and compare the performance and the code size of our implementation of HECC on five different compilers.

We implemented the a hyperelliptic curve cryptosystem using the group operations presented in [PWP03] with group order $\approx 2^{212}$. Basic operations of the HECC were executed multiple times. The code was written using ANSI C language and we compiled the program using the five most used compilers targeting the ARM processor. For the timing measurements we used the Samsung SNDS100 (ARM7TDMI).

Figures 1 and 2 depict execution times and code sizes for different compilers and different optimization levels. Except for one case, no remarkable differences in the execution time as well as in the code size are noticeable.

One exception is the GNU compiler using optimization level one which seems to be a bad choice for performance and program code compared to the other optimized settings. The GNU compiler creates highly efficient code for higher optimization levels compared to the other compilers under test.

We conclude from the given figures that switching to higher optimization levels does not gain big advantages. Nevertheless, when speed or code size is very critical for a given application a high optimization level should be applied.

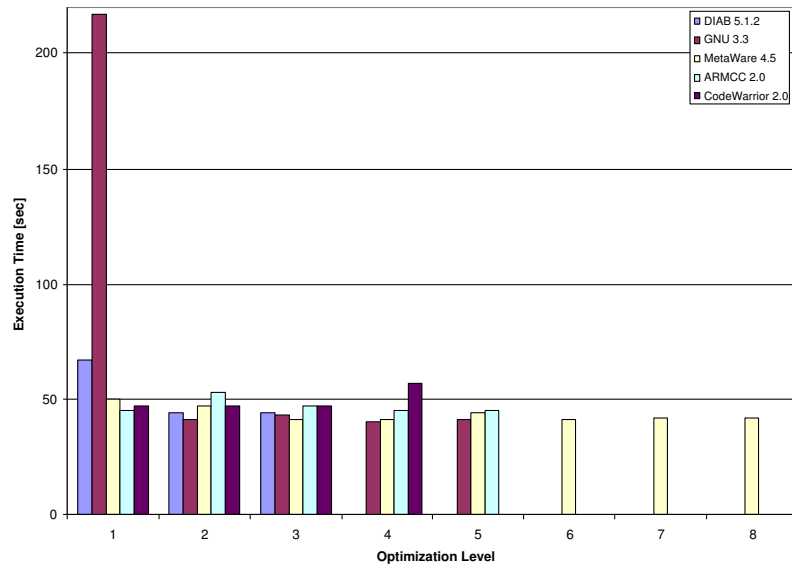


Fig. 1. Time comparison of HECC using different compilers (ARM7TDMI@50MHz)

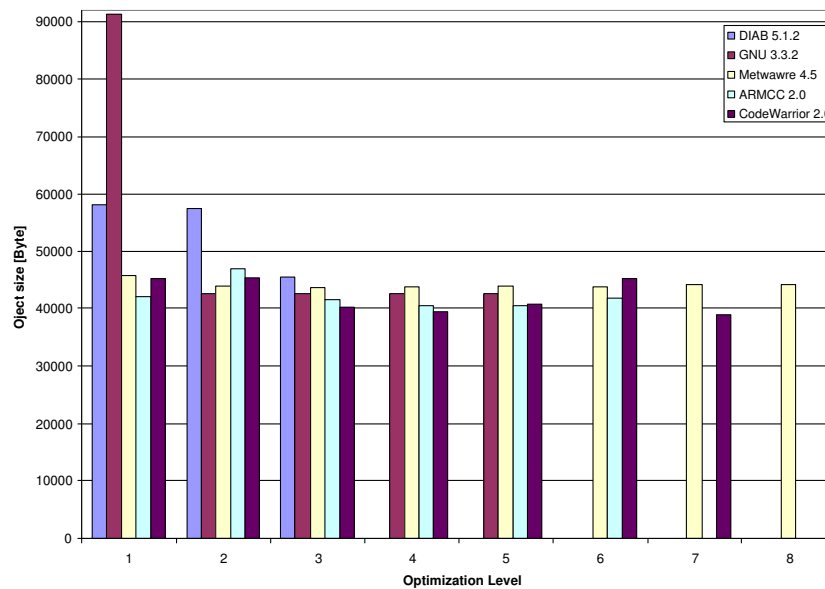


Fig. 2. Code size comparison of HECC using different compilers (ARM7TDMI@50MHz)

The use of the GNU compiler with high optimization level results in the the best possible performance (Figure 1). The CodeWarrior compiler can produce the smallest code for the given cryptographic application (Figure 2).

5 Conclusion

In this contribution, we implemented a public-key cryptosystem based on hyperelliptic curves in C. This system is very well suited for the use in embedded environments because of the short operand length compared to other public-key algorithms. We compiled the code on five different compilers using the optimization levels available.

We conclude from the given results that the code produced from the compilers perform within the same magnitude. Additionally, the code size is very similar in all cases. When targeting high efficiency, the optimization level one within the GNU compiler should be avoided.

Acknowledgement: We would like to thank “HSP Embedded Tools” for assisting our project with professional equipment and advice. Especial thanks go out to E. Barteska and V. Wittelsberger for the help on getting the performance and code size numbers.

References

- [BCLW02] N. Boston, T. Clancy, Y. Liow, and J. Webster. Genus Two Hyperelliptic Curve Coprocessor. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2002*, LNCS 2523, pages 529–539. Springer-Verlag, 2002. Updated version available at <http://www.cs.umd.edu/~clancy/docs/hec-ches2002.pdf>.
- [Can87] D.G. Cantor. Computing in Jacobian of a Hyperelliptic Curve. In *Mathematics of Computation*, volume 48(177), pages 95 – 101, January 1987.
- [Cla02] T. Clancy. Analysis of FPGA-based Hyperelliptic Curve Cryptosystems. Master’s thesis, University of Illinois Urbana-Champaign, December 2002.
- [Cla03] T. Clancy. FPGA-based Hyperelliptic Curve Cryptosystems. invited paper presented at AMS Central Section Meeting, April 2003.
- [EMY04] G. Elias, A. Miri, and T. H. Yeap. High Performance Hyperelliptic Curve Cryptosystem on an FPGA. In *22nd Biennial Symposium on Communications*, May 2004.
- [Eng99] A. Enge. Computing Discrete Logarithms in High-Genus Hyperelliptic Jacobians in Provably Subexponential Time. http://www.math.waterloo.ca/Cond0_Dept/CORR/corr99.html, 1999. Preprint.
- [Ful69] W. Fulton. *Algebraic Curves - An Introduction to Algebraic Geometry*. W. A. Benjamin, Inc., Reading, Massachusetts, 1969.
- [GH00] P. Gaudry and R. Harley. Counting Points on Hyperelliptic Curves over Finite Fields. In W. Bosma, editor, *ANTS IV*, LNCS 1838, pages 297 – 312, Berlin, 2000. Springer Verlag.
- [Kri97] U. Krieger. signature.c. Master’s thesis, Mathematik und Informatik, Universität Essen, Fachbereich 6, Essen, Germany, February 1997.
- [Mum84] D. Mumford. Tata Lectures on Theta II. In *Prog. Math.*, volume 43. Birkhäuser, 1984.
- [MWZ98] A. Menezes, Y. Wu, and R. Zuccherato. *An Elementary Introduction to Hyperelliptic Curves*. Springer-Verlag, Berlin, Germany, first edition, 1998. In: N. Koblitz, *Algebraic Aspects of Cryptography*.
- [Ngu02] K. Nguyen. Curve based cryptography - the state of the art in smart card environments. In *6rd Workshop on Elliptic Curve Cryptosystems (ECC '02)*, Essen, Germany, September 23–25 2002. Invited Contribution.
- [Pel02] J. Pelzl. Hyperelliptic Cryptosystems on Embedded Microprocessor. Master’s thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitaet Bochum, Bochum, Germany, September 2002.
- [PWGP03] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic Curve Cryptosystems: Closing the Performance Gap to Elliptic Curves. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Workshop on Cryptographic Hardware and Embedded Systems — CHES 2003*, LNCS 2779, pages 349 – 365. Springer-Verlag, September 2003.
- [PWP03] J. Pelzl, T. Wollinger, and C. Paar. Low Cost Security: Explicit Formulae for Genus-4 Hyperelliptic Curves. In M. Matsui and R. Zuccherato, editors, *Tenth Annual Workshop on Selected Areas in Cryptography — SAC 2003*, LNCS 3006, pages 1 – 16. Springer-Verlag, 2003.

- [PWP04a] J. Pelzl, T. Wollinger, and C Paar. *Embedded Cryptographic Hardware: Design and Security*, chapter Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation. Nova Science Publishers, NY, USA, 2004. editor Nadia Nedjah.
- [PWP04b] J. Pelzl, T. Wollinger, and C. Paar. High Performance Arithmetic for Special Hyperelliptic Curve Cryptosystems of Genus Two. In *International Conference on Information Technology: Coding and Computing - ITCC 2004*. IEEE Computer Society, April 2004.
- [Sma99] N. Smart. On the Performance of Hyperelliptic Cryptosystems. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99*, LNCS 1592, pages 165–175. Springer-Verlag, 1999.
- [SS98] Y. Sakai and K. Sakurai. Design of Hyperelliptic Cryptosystems in small Characteristic and a Software Implementation over \mathbb{F}_{2^n} . In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98*, LNCS 1514, pages 80 – 94, Berlin, 1998. Springer Verlag.
- [SS00] Y. Sakai and K. Sakurai. On the Practical Performance of Hyperelliptic Curve Cryptosystems in Software Implementation. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E83-A NO.4, pages 692 – 703, April 2000.
- [SSI98] Y. Sakai, K. Sakurai, and H. Ishizuka. Secure Hyperelliptic Cryptosystems and their Performance. In H. Imai and Y. Zheng, editors, *Public Key Cryptography: First International Workshop on Practice and Theory in Public Key Cryptography — PKC'98*, LNCS 1431, pages 164 – 181, Berlin, 1998. Springer-Verlag.
- [Wol01] T. Wollinger. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves. Master's thesis, ECE Department, Worcester Polytechnic Institute, Worcester, Massachusetts, USA, May 2001.
- [WP02] T. Wollinger and C. Paar. Hardware Architectures Proposed for Cryptosystems Based on Hyperelliptic Curves. In *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002*, volume III, pages 1159 – 1163, September 2002.
- [WPW⁺04] T. Wollinger, J. Pelzl, V. Wittelsberger, C Paar, G. Saldamli, and Ç. K. Koç. Elliptic & Hyperelliptic Curves on Embedded μ P. *ACM Transactions in Embedded Computing Systems (TECS)*, 2004. Special Issue on Embedded Systems and Security, to appear.