**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR. NO.** | **TOPIC** | **PAGE NO.** | |
| 1. | [Abstract](#ABSTRACT) | |  |
| 2. | Chapter 1: [Introduction](#INTRODUCTION) ......................................................................... | | 3 |
| 3. | Chapter 2: [Theoretical Background](#BACKGROUND) ......................................................  2.1 Homomorphic Encryption ..............................................  2.2 Wireless Sensor Networks ..............................................  2.3 Elliptic Curve Cryptography .......................................... | | 2  2  6  13 |
| 4. | Chapter 3: [Problem Statement & Related Work](#PSRW) | | 18 |
| 5. | Chapter 4: [Implementation](#IMPLEMENTATION) ...................................................................  4.1 Methodology of Evaluation ............................................  4.2 Tools investigated ...........................................................  4.3 Test Application ..............................................................  4.4 Profiling Tools Investigated.............................................  4.5 Profiling ......................................................................... | | 21  21  25  28  33  35 |
| 6. | Chapter 5: [Profiling Results](#PROFILING_RESULTS)  5.1 Runtime  5.2 Memory Consumption  5.3 Power Consumption | | 37  37  44  51 |
| 7. | Chapter 6: [Conclusion & Future Work](#CONCLUSION) ................................................. | | 56 |
| 8. | [References](#REFERENCES) ......................................................................................... | | III |
|  |  | |  |

**ABSTRACT**

*A combination of symmetric and asymmetric encryption techniques are used to securely transmit the sensitive data over the insecure Internet. To perform any processing of such sensitive data without giving access to the data itself requires the use of fully homomorphic encryption scheme, which requires heavy computation. This becomes a significant factor if we are using resource constrained small portable device, and gives rise to the need of Elliptic Curve Cryptography (ECC), which offers equivalent security with smaller key sizes, which can be computed with low resources.*

*This paper presents the implementation of three ECC (Elliptic Curve Cryptography) algorithms: ECIES (Integrated Encryption Scheme), ECDSA (Digital Signature Algorithm), and ECDH (Diffie Hellman), over the prime field, and the profiling of their execution time and resource consumption. The primary objective of the experiment is to perform a comparative study between the algorithms, as each algorithm can be extended to provide the full confidentiality , integrity and authentication functionality. This analysis can further achieve the enablement of these algorithms in the Wireless Sensor Networks, where security with low overheads is of prime importance.*

**CHAPTER 1**

**INTRODUCTION**

A large portion of Internet applications require the exchange of sensitive data between servers and clients over the Internet. A combination of symmetric and asymmetric encryption techniques are used to securely transmit the sensitive data over the insecure Internet. But any processing of such sensitive data requires giving access to the party which needs to process it. This again gives rises to privacy issues. A solution to this problem could be to enable processing of data without decrypting it. This is possible only if we use a fully homomorphic encryption scheme, which is a form of encryption where a specific algebraic operation is performed on the plaintext and another (possibly different) algebraic operation is performed on the ciphertext. This requires heavy computation, which becomes a significant factor in speed performance as key length grows. Meanwhile, more and smaller portable devices and embedded systems with limited computing and power resources are being used to increase the mobility of the users. Thus, we need a public-key cryptosystem, like Elliptic Curve Cryptography (ECC), that offers equivalent security with smaller key sizes, which can be computed with less processing power, memory, and lower power consumption.

This paper discusses the implementation of three ECC (Elliptic Curve Cryptography) algorithms: viz ECIES (Integrated Encryption Scheme), ECDSA (Digital Signature Algorithm), ECDH (Diffie Hellman), and their profiling to perform a comparative study of their execution time and resource consumptions. The experimental results provide useful guidance for users, enabling the selection of algorithms for use as per the security need and the resource constraints.

The remainder of this paper is organized as follows. Section 2 presents the theoretical background, consisting of homomorphic encryption, wireless sensor networks and ECC. Section 3 defines the problem statement and discusses the related work. Section 4 describes the implementation methodology and profiling details. The next section, based on profiling results, analyses usage of algorithms as per user requirements and system constraints. The last section draws conclusions and the scope of future research areas.

**CHAPTER 2**

**THEORETICAL BACKGROUND**

**2.1 HOMOMORPHIC ENCRYPTION [4]**

Homomorphic encryption is a form of encryption where a specific algebraic operation is performed on the plaintext and another (possibly different) algebraic operation is performed on the ciphertext such that the decryption of result of algebraically processed ciphertext is same as that of the result of algebraically processed plaintext. Thus, homomorphic encryption helps to process data without compromising on the privacy of the data. Homomorphic encryption scheme ε has three algorithms: KeyGenε, Encryptε, and Decryptε. The different classes of the scheme are fully homomorphic encryption, somewhat homomorphic encryption and bootstrappable encryption.

**2.2 WIRELESS SENSOR NETWORKS [5]**

A typical sensor network consists of a large number of small, low-power, low-cost nodes that form a self-organized network using wireless peer-to-peer communication [2]. In the above context, there are various limitations to wireless sensor networks : limited energy, memory, transmission range, fault tolerance, self-organization, scalability. Hence, protocols designed for sensor networks should be highly scalable. The security requirements include authentication, access-control, privacy, integrity, authorization, anonymity, non-repudiation, resilience, forward and backward secrecy, and survivability. Hence homomorphic encryption finds considerable use here by providing security in constrainted resources.

* 1. **ELLIPTIC CURVE CRYPTOGRAPHY**

Elliptic Curve Cryptography (ECC) is a public key cryptography. Some public key algorithms may require a set of predefined constants to be known by all the devices taking part in the communication. ‘Domain parameters’ in ECC is an example of such constants. The mathematical operations of ECC is defined over the elliptic curve y2 = x3 + ax + b, where 4a3 + 27b2 ≠ 0. Each value of the ‘a’ and ‘b’ gives a different elliptic curve. The public key is a point in the curve and the private key is a random number. One main advantage of ECC is its small key size. A 160-bit key in ECC is considered to be as secured as 1024-bit key in RSA.Some principles in the design and development of ECC are :

* Security: ECC should provide PKC schemes that have proved to be secure. ECC only includes support for the well-studied ECC schemes such as ECDSA, ECDH, and ECIES, which are defined in the Standards for Efficient Cryptography [10]
* Portability: ECC should run on as many platforms as possible.
* Resource Awareness and Configurability: ECC is implemented carefully to avoid unnecessary resource usage. ECC uses a set of optimization switches, which can be turned on or off to achieve different combinations of performance and resource consumptions.
* Efficiency: ECC should be computationally efficient to reduce the battery consumption as well as the delay introduced by PKC operations.
* Functionality: ECC should support the typical demands for PKC. Hence the current version of ECC includes a digital signature scheme (ECDSA), a key exchange protocol (ECDH), and a public key encryption scheme (ECIES).
  + 1. **Cryptographic Schemes**
* ECDSA - Elliptic Curve Digital Signature Algorithm [6]

ECDSA is a variant of the Digital Signature Algorithm (DSA) that operates on elliptic curve groups. For sending a signed message from A to B, both have to agree up on Elliptic Curve domain parameters. The ECDSA process is of two steps : Signature Generation and Signature Verification.

* ECIES – Elliptic Curve Integrated Encryption Scheme [16]

Elliptic Curve Integrated Encryption Scheme (ECIES) ia an incarnation of the Integrated Encryption Scheme (IES), a public-key encryption scheme which provides semantic security against chosen-plaintext and chosen-ciphertext attacks. The scheme is based on Diffie–Hellman problem. To send an encrypted message to B using ECIES , A needs the following information :

* cryptographic suite to be used:
* EC domain parameters (p,a,b,G,n,h) for a curve over prime field or(m,f(x),a,b,G,n,h) for a curve over binary field;
* B's public key
* optional shared information: S1 and S2.
* ECDH – Elliptic Curve Diffie Hellman [6]

ECDH is a key agreement protocol that allows two parties to establish a shared secret key that can be used for private key algorithms. Both parties exchange some public information to each other. Using this public data and their own private data these parties calculate calculates the shared secret. Any third party, who doesn’t have access to the private details of each device, will not be able to calculate the shared secret from the available public information.

**CHAPTER 3**

**PROBLEM STATEMENT AND RELATED WORK**

The primary objective of this paper is to implement the three ECC (Elliptic Curve Cryptography) algorithms: ECIES (Integrated Encryption Scheme), ECDSA (Digital Signature Algorithm), ECDH (Diffie Hellamn); profile them in order to find their execution time and resource consumptions, and draw conclusions and analysis for their usage suitability. This requires investigation of ECC libraries to find a suitable one for implementing the three algorithms on our chosen platform. After the implementation of the three algorithms, we have to choose appropriate performance profiling tool(s). The results of profiling will help us to perform a comparative study of execution time and resource consumption of ECIES, ECDSA and ECDH.

A few similar past attempts include [1], which presents the design, implementation, and evaluation of TinyECC, a configurable library for ECC operations in wireless sensor networks. In [1] TinyECC library requires the support of TinyOS and can only be used for simulation on motes for WSN based applications. On the other hand, we are using Cryptopp library which supports generic real world applications. Also [1] draws conclusions based on mote simulation, whereas we would be deriving the conclusions from profiling results.

[2] discusses the state of the art in terms of sensor network security and we examine the practicality of using efficient elliptic curve algorithms and identity based encryption to deploy a secure sensor network infrastructure. In [2] the library used is MIRACL and the profiler tool is Enprofiler (ARM7DTMI processor based profiler), whereas we are using CryptoPP library and a linux based profiler. The algorithms profiled in [2] are different from the algorithms we are studying. Also, [2] profiles various operations of ECC.

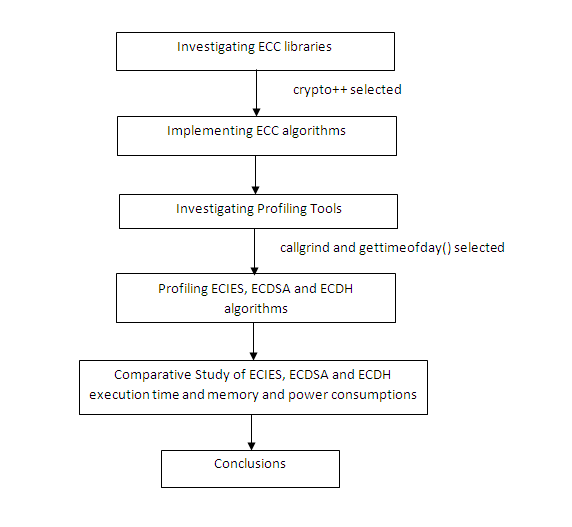
[3] discusses the software implementations of ECC on processors with different word sizes. In [3], EEA algorithm has been profiled on processors of different word sizes whereas we would be profiling three ECC algorithms namely ECIES, ECDSA and ECDH.

**CHAPTER 4**

**IMPLEMENTATION**

* 1. **METHODOLOGY OF EVALUATION**

The basic flow for the methodology of evaluation is given in fig. 4.1

****

**Figure 4.1: Block Diagram for Methodology of Evaluation**

* 1. **TOOLS INVESTIGATED**

**Libraries Tested**

* + TinyECC [8]

TinyECC is a software package providing ECC-based PKC operations that can be flexibly configured and integrated into sensor network applications. It provides a digital signature scheme (ECDSA), a key exchange protocol (ECDH), and a public key encryption scheme (ECIES). It provides a number of optimization switches, which can turn specific optimizations on or off based on developer's needs.

Problems faced:

1. TinyOS 1.1 supports the platforms mica and pc. Our objective was to simulate on pc platform. But tinyECC runs on platforms : micaZ, TelosB/ TmoteSky and imote2. It doesn’t have support for pc platform
2. TinyOS 2.0 supports the platforms required for TinyECC but it not possible to see the simulation on pc.
   * Libecc [9]

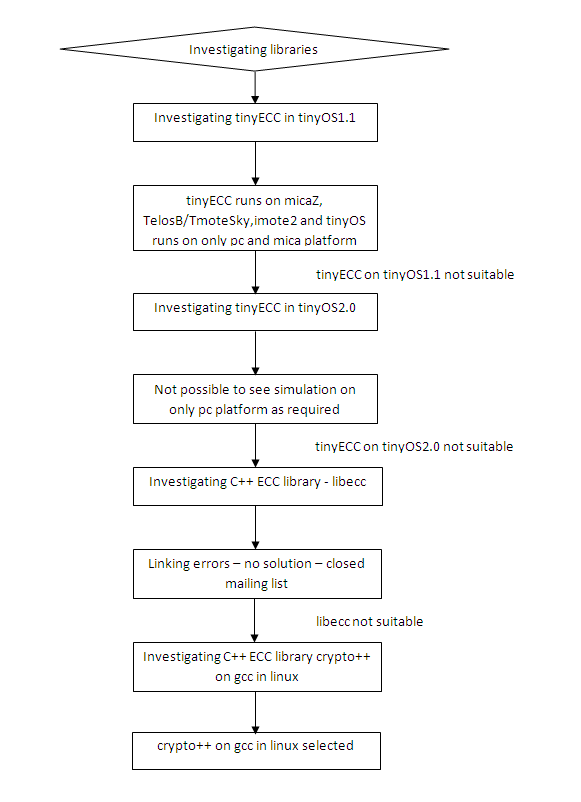
'libecc' is a C++ ECC library that supports fixed-size keys for maximum speed. Its goal is to become the first free library to let users generate safe elliptic curves, and to provide an important source of information for anyone with general interest in ECC. Platform supported are Turbo C++, Microsoft Visual C++ and GCC. Libecc version 0.12.1 was used on platform Microsoft Visual C++ and gcc.

Problems faced : Linking error, which we could not be resolved due to closed mailing list.

* + CryptoPP [11]

Crypto++ (also known as CryptoPP, libcrypto++, and libcryptopp) is a free and [open source](http://en.wikipedia.org/wiki/Open_source) C++ class library of cryptographic algorithms and schemes written by Wei Dai. Crypto++ has been widely used in academia, student projects, open source and non-commercial projects, as well as businesses. The project also supports compilation under a variety of [compilers](http://en.wikipedia.org/wiki/Compilers) and [IDEs](http://en.wikipedia.org/wiki/Integrated_development_environment), including [Borland Turbo C++](http://en.wikipedia.org/wiki/Borland_C%2B%2B), [Borland C++ Builder](http://en.wikipedia.org/wiki/Borland_C%2B%2B), [CodeWarrior Pro](http://en.wikipedia.org/wiki/CodeWarrior), [GCC](http://en.wikipedia.org/wiki/GNU_Compiler_Collection) (including Apple's GCC), [Intel C++ Compiler (ICC)](http://en.wikipedia.org/wiki/Intel_C%2B%2B_Compiler), [Microsoft Visual C/C++](http://en.wikipedia.org/wiki/Visual_C%2B%2B) and Sun Studio. ECC algorithms supported are ECDSA, ECNR, ECIES, ECDH and ECMQV. Version investigated was crypto++5.5.2 on GCC in Linux.

The flow of investigation of libraries is given in fig. 4.2.

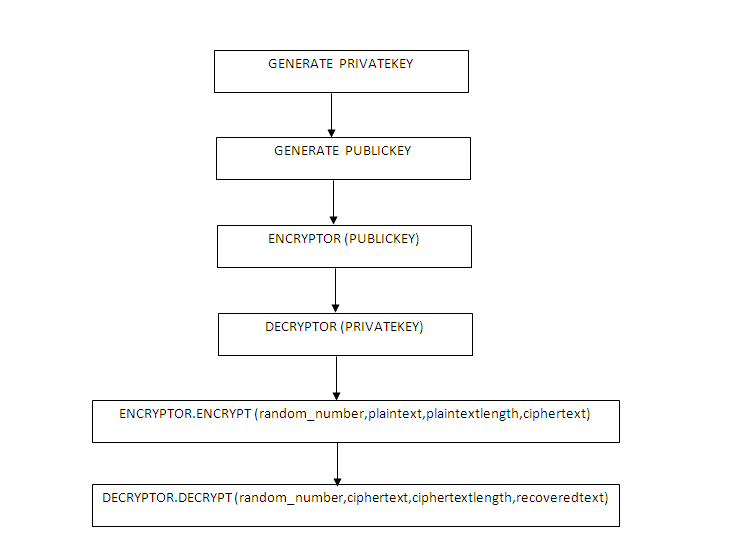
****

**Figure 4.2: Block Diagram for Investigation of ECC libraries**

* 1. **TEST APPLICATION**

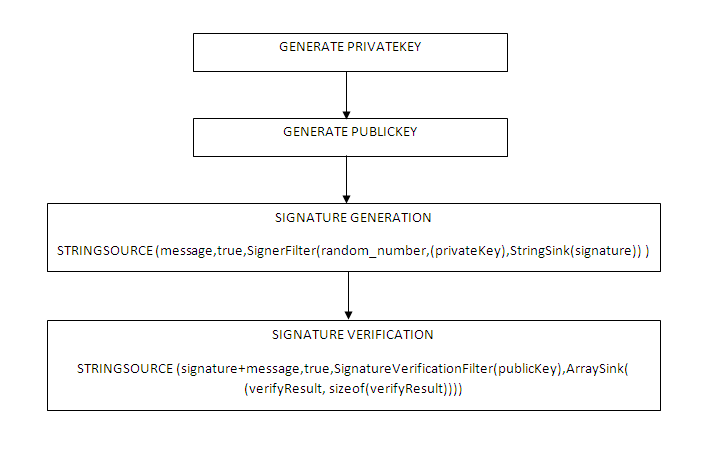
Our test application consists of 3 algorithms : ECIES, ESDSA and ECDH. The test data we used is on “Blood Transfusion Service Centre Data Set” taken from UCLA repository. The fields/attributes of this data set are Recency (months), Frequency (times), Monetary (c.c. Blood), Time (months),"whether he/she donated blood in March 2007".

* + 1. **ECIES (Elliptic Curve Integrated Encryption Scheme)**



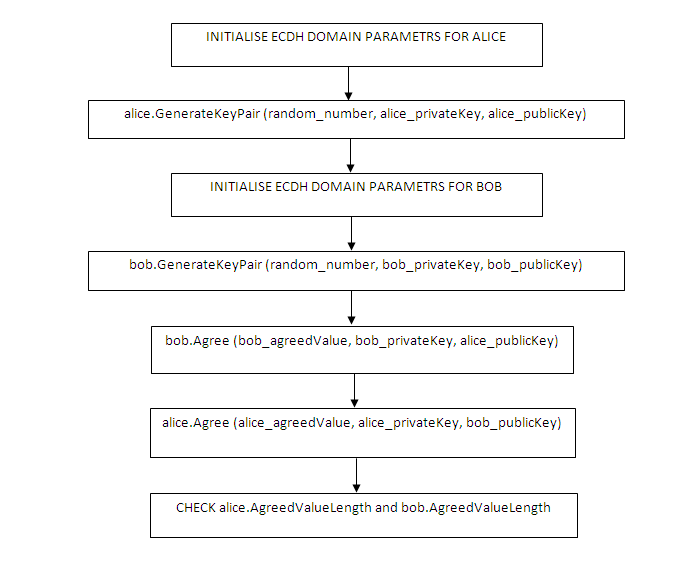
**Figure 4.3: Block diagram for ECIES**

**4.3.2 ECDSA (Elliptic Curve Digital Signature Algorithm)**

****

**Figure 4.4: Block diagram for ECDSA**

* + 1. **ECDH (Elliptic Curve Diffie Helman)**

****

**Figure 4.5: Block diagram for ECDH**

**4.4 PROFILING TOOLS INVESTIGATED**

**4.4.1 gprof**

The GNU profiler gprof is a useful tool for measuring the performance of a program--it records the number of calls to each function and the amount of time spent there, on a per-function basis. We investigated gprof for obtaining the runtime of the three algorithms, viz ECIES, ECDSA and ECDH. The reasons for the unsuitability of gprof profiling tool for our application were:

* the maximum resolution of gprof is milliseconds
* our application requires a microsecond resolution timer, for more precision.
  + 1. **gettimeofday( ) – UNIX system call [17]**

The gettimeofday() function shall obtain the current time, expressed as seconds and microseconds since the Epoch, and store it in the timeval structure pointed to by *tp*. The resolution of the system clock is unspecified.

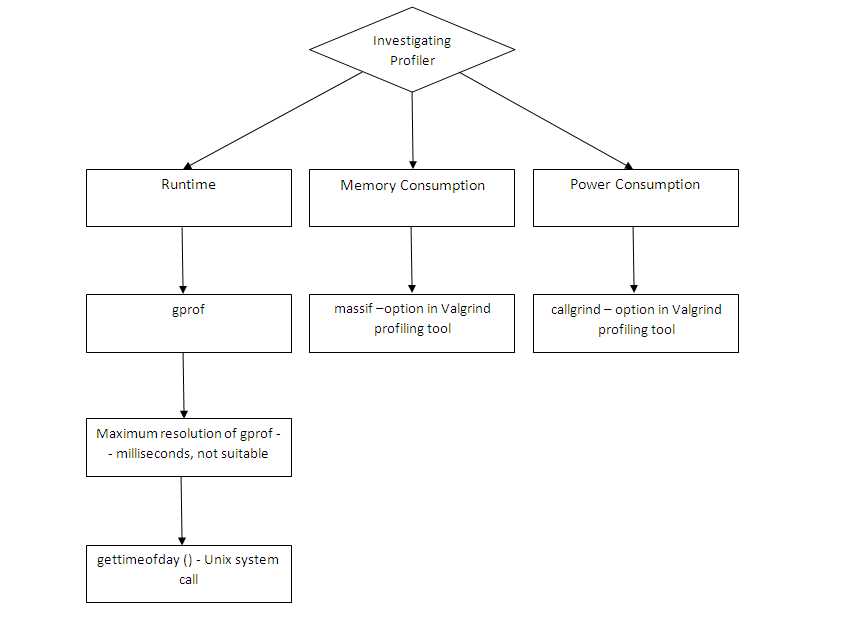
* + 1. **Valgrind [18]**

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. We can also use Valgrind to build new tools. The Valgrind distribution currently includes six production-quality tools : Memcheck, Cachegrind, Callgrind, Massif, Helgrind, Lackey and Nulgrind.

It also includes three experimental tools: a heap/stack/global array overrun detector, a second heap profiler that examines how heap blocks are used, and a SimPoint basic block vector generator. It runs on the following platforms: X86/Linux, AMD64/Linux, ARM/Linux, PPC32/Linux, PPC64/Linux, X86/Darwin and AMD64/Darwin

The tools massif and callgrind were considered for profiling the heap memory consumption and power consumption, respectively, of the three algorithms.

* Callgrind, by Josef Weidendorfer, is an extension to Cachegrind. It provides all the information that Cachegrind does, plus extra information about callgraphs. It was folded into the main Valgrind distribution in version 3.2.0. Available separately is an amazing visualisation tool, KCachegrind, which gives a much better overview of the data that Callgrind collects; as is required by our application [19]. KCachegrind is a GUI based tool for the same.
* Massif is a heap profiler. It performs detailed heap profiling by taking regular snapshots of a program's heap. It produces a graph showing heap usage over time, including information about which parts of the program are responsible for the most memory allocations. Massif runs programs about 20x slower than normal [19].

****

**Figure 4.6: Block Diagram for Investigation of Profiling Tool**

**CHAPTER 5**

**PROFILING RESULTS**

**5.1 RUNTIME**

The following readings were obtained for the runtime (in microseconds) of the three algorithms:

**Table 5.1: Runtime (microseconds) for ECIES, ECDSA and ECDH**

|  |  |
| --- | --- |
| Algorithm | Runtime (microseconds) |
| ECIES | 352810 |
| ECDSA | 287332 |
| ECDH | 28027 |

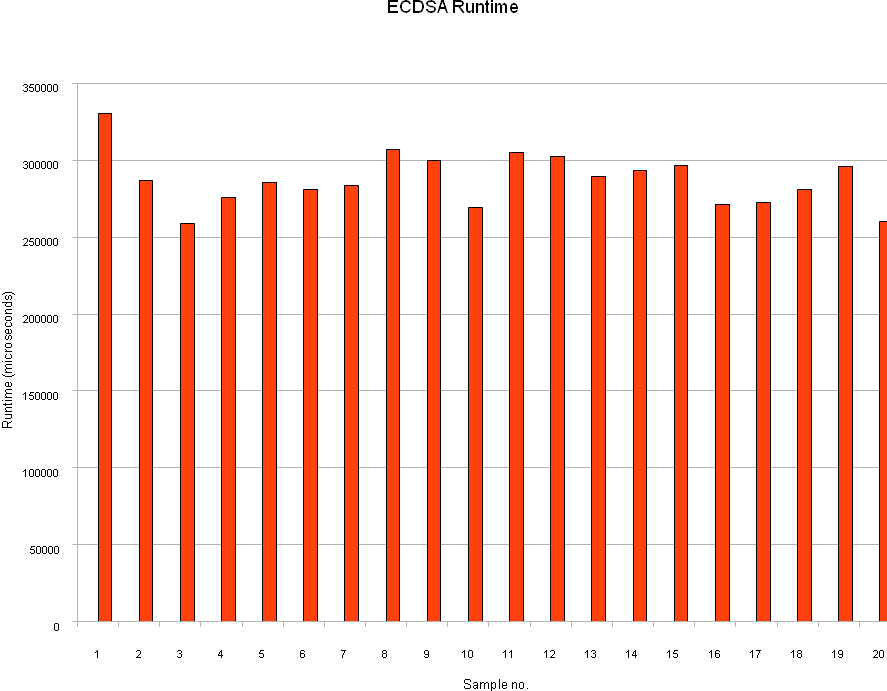
**Figure 5.1 : Graph of Runtime (microseconds) of ECIES and ECDSA**

Reason for the above results obtained is as follows:

|  |  |  |
| --- | --- | --- |
| PROGRAM | FUNCTION | RUNTIME (microseconds) |
| ECIES | Encryptor.Encrypt() | 6614 |
| Decryptor.Decrypt() | 4267 |
| ECDSA | Signer.SignandRestart() | 3398 |
| Verifer.VerifyandRestart() | 5859 |

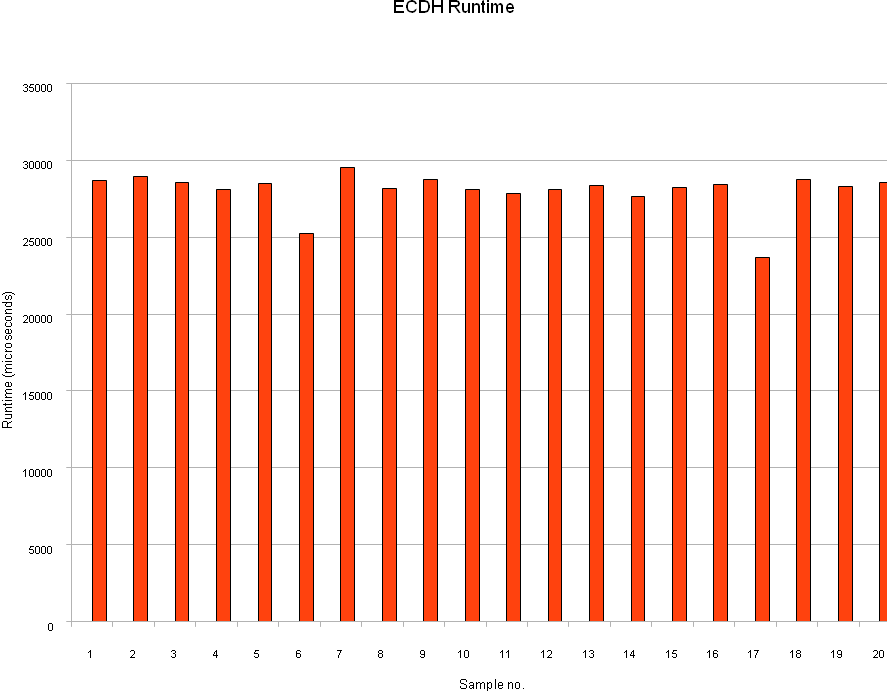
**Figure 5.2: Plot of sample readings of runtime of ECIES**

Average Runtime for ECIES = 352810 microseconds

****

**Figure 5.3: Plot of sample readings of runtime of ECDSA**

Average Runtime for ECDSA = 287332 microseconds

****

**Figure 5.4: Plot of sample readings of runtime of ECDH**

Average Runtime for ECIES = 28027 microseconds

**5.2 MEMORY CONSUMPTION**

Heap memory (bytes) consumed by each algorithm was found out by the massif tool of Valgrind. Obtained below are the snapshots for each of the three algorithms.

**Table 5.5: Heap memory consumption (bytes) for ECIES, ECDSA and ECDH**

|  |  |
| --- | --- |
| Algorithm | Heap Memory Consumption (bytes) |
| ECIES | 51424 |
| ECDSA | 43880 |
| ECDH | 15680 |

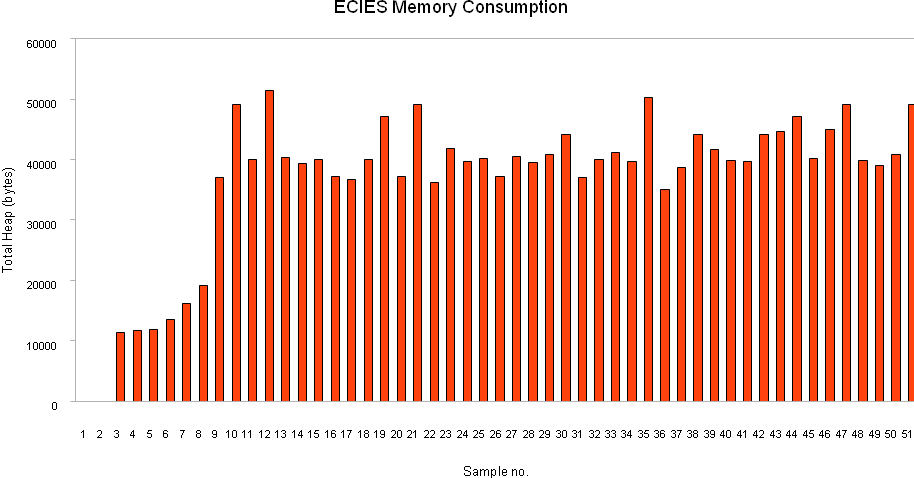
**Figure 5.5 : Graph of Heap memory Consumption (bytes) of ECIES and ECDSA**

Reason for the above results obtained is as follows:

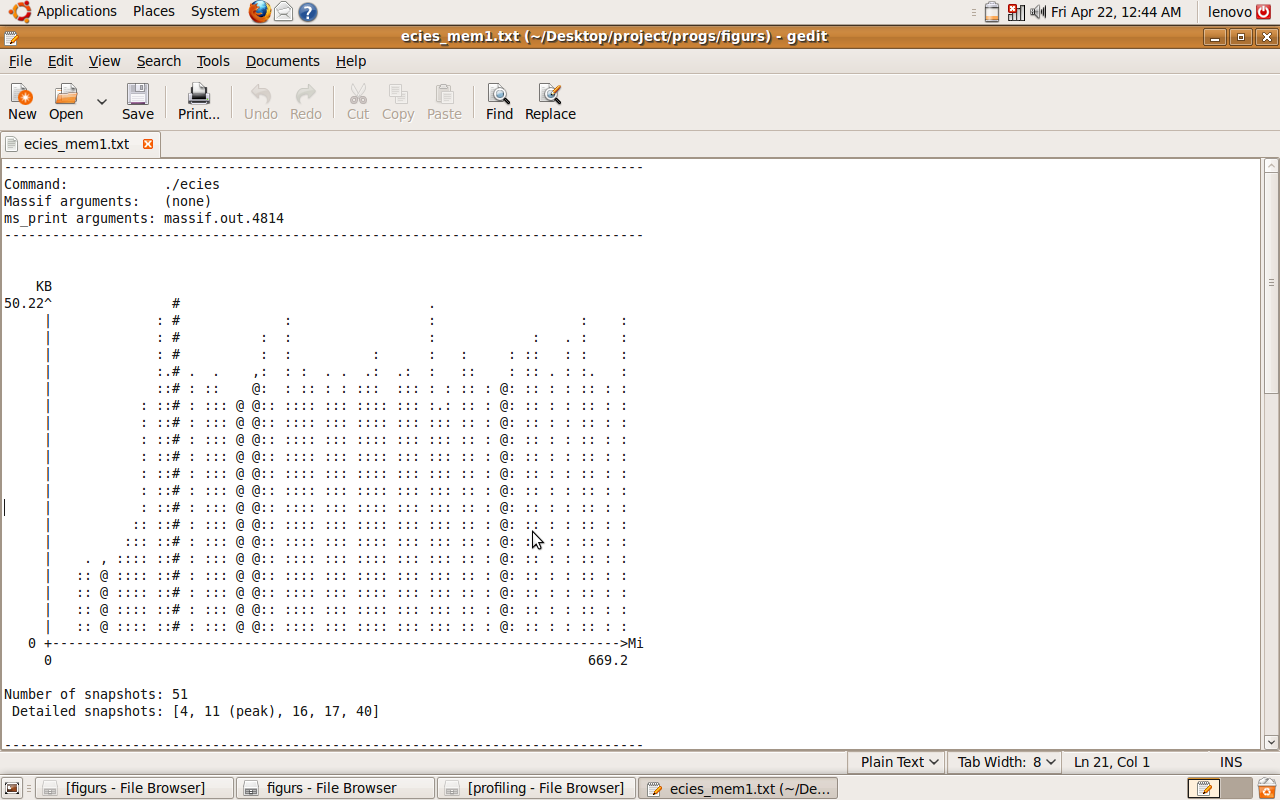
|  |  |  |
| --- | --- | --- |
| PROGRAM | FUNCTION | MEMORY (bytes) |
| ECIES | Encryptor.Encrypt() | 2880 |
| Decryptor.Decrypt() | 1536 |
| ECDSA | Signer.SignandRestart() | 512 |
| Verifer.VerifyandRestart() | 704 |

**Table 5.6: Sample Reading for peak values of heap memory consumed (bytes) by ECIES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample no. | time(i) | Total Heap | Useful Heap | Extra Heap | Stack |
| **11 (peak)** | **152997914** | **51424** | **44546** | **6878** | **0** |

****

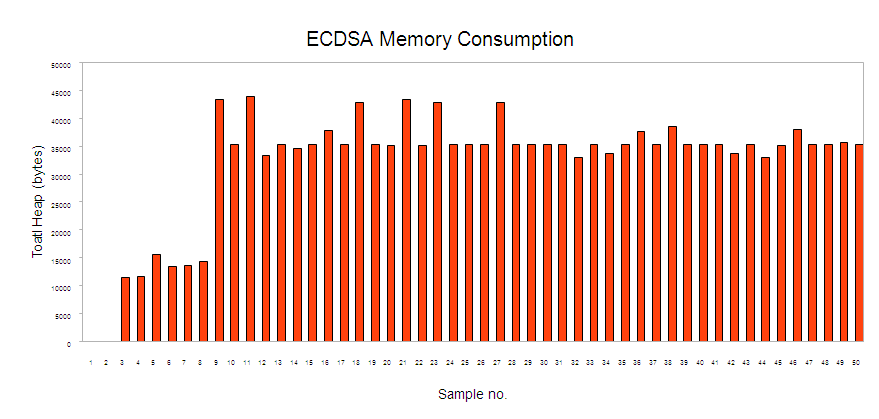
**Figure 5s.6: Plot of Sample Readings for heap memory consumed (bytes) by ECIES**

****

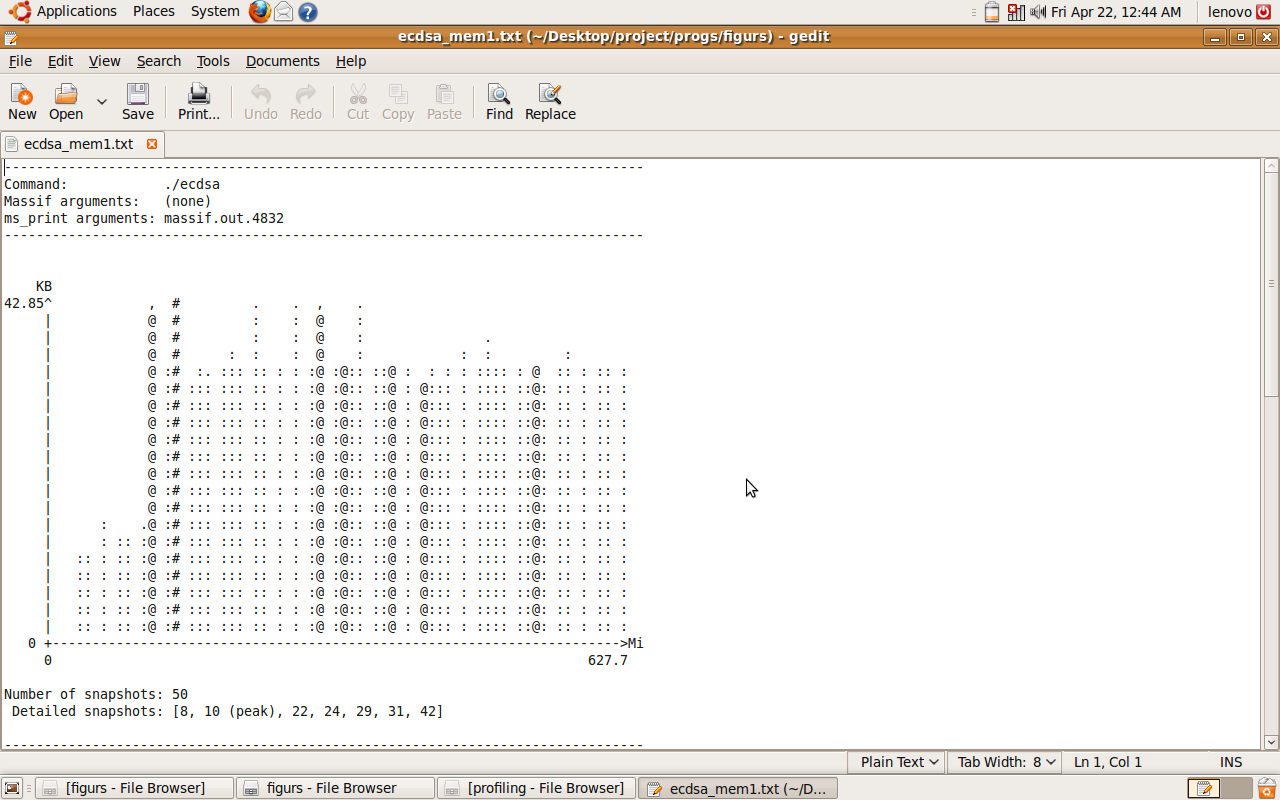
**Figure 5.7: Screenshot of result obtained by msprint command in massif tool for ECIES**

**Table 5.7: Sample Readings for peak values of heap memory consumed (bytes) by ECDSA**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample no. (n) | time(i) | total heap | useful heap | extra heap | stack |
| **10 (peak)** | **139675996** | **43880** | **38710** | **5170** | **0** |

****

**Figure 5.8: Plot of Sample Readings for heap memory consumed (bytes) by ECDSA**

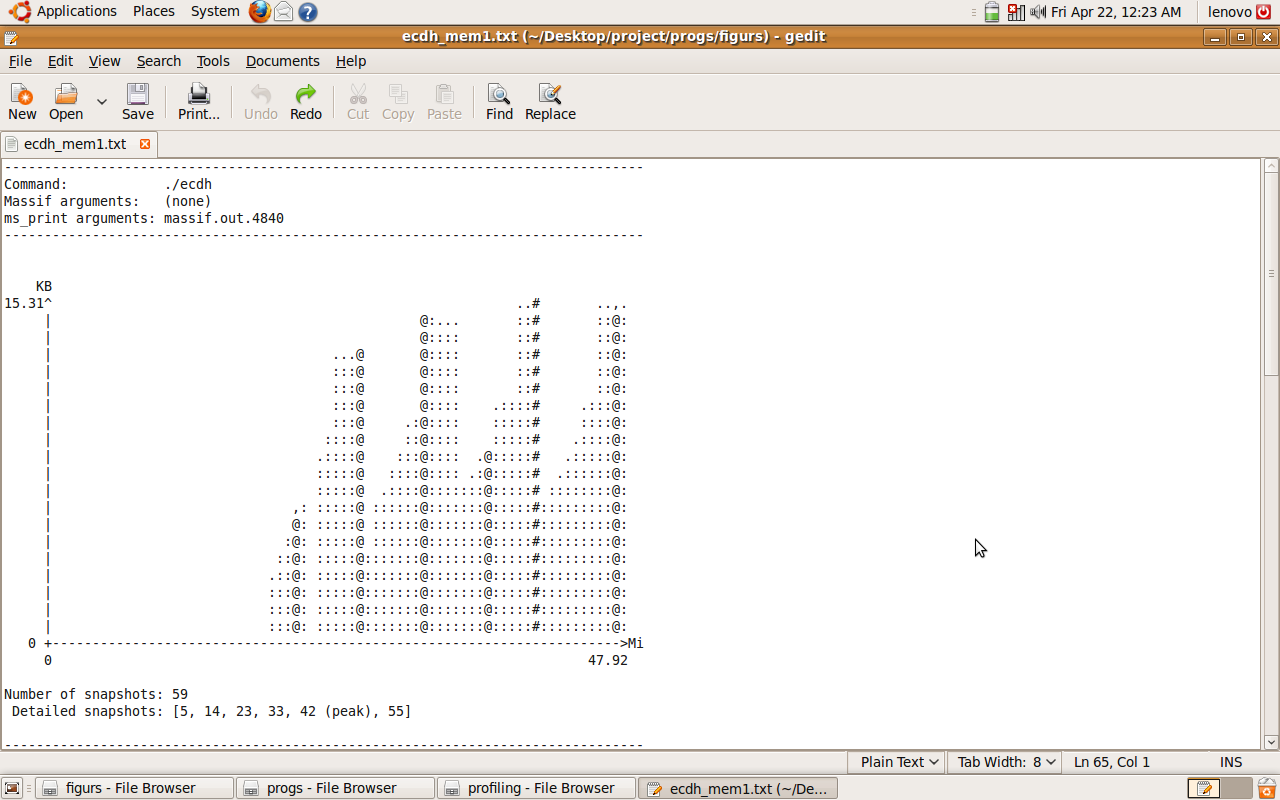
****

**Figure 5.9: Screenshot of result obtained by msprint command in massif tool for ECDSA**

**Table 5.8: Sample Readings for peak values of heap memory consumed (bytes) by ECDH**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample no. | time(i) | total heap | useful heap | extra heap | stack |
| **42 (peak)** | **42492883** | **15680** | **11776** | **3904** | **0** |

**Figure 5.10: Plot of Sample Readings for heap memory consumed (bytes) by ECDH**

****

**Figure 5.11: Screenshot of result obtained by msprint command in massif tool for ECDH**

**5.3 POWER CONSUMPTION**

As discussed earlier, the power consumption would be obtained through the total Instruction Fetch (IF) cost parameter of the algorithms, for which the Valgrind tool Callgrind was employed. The following are the snapshots available from Kcachegrind.

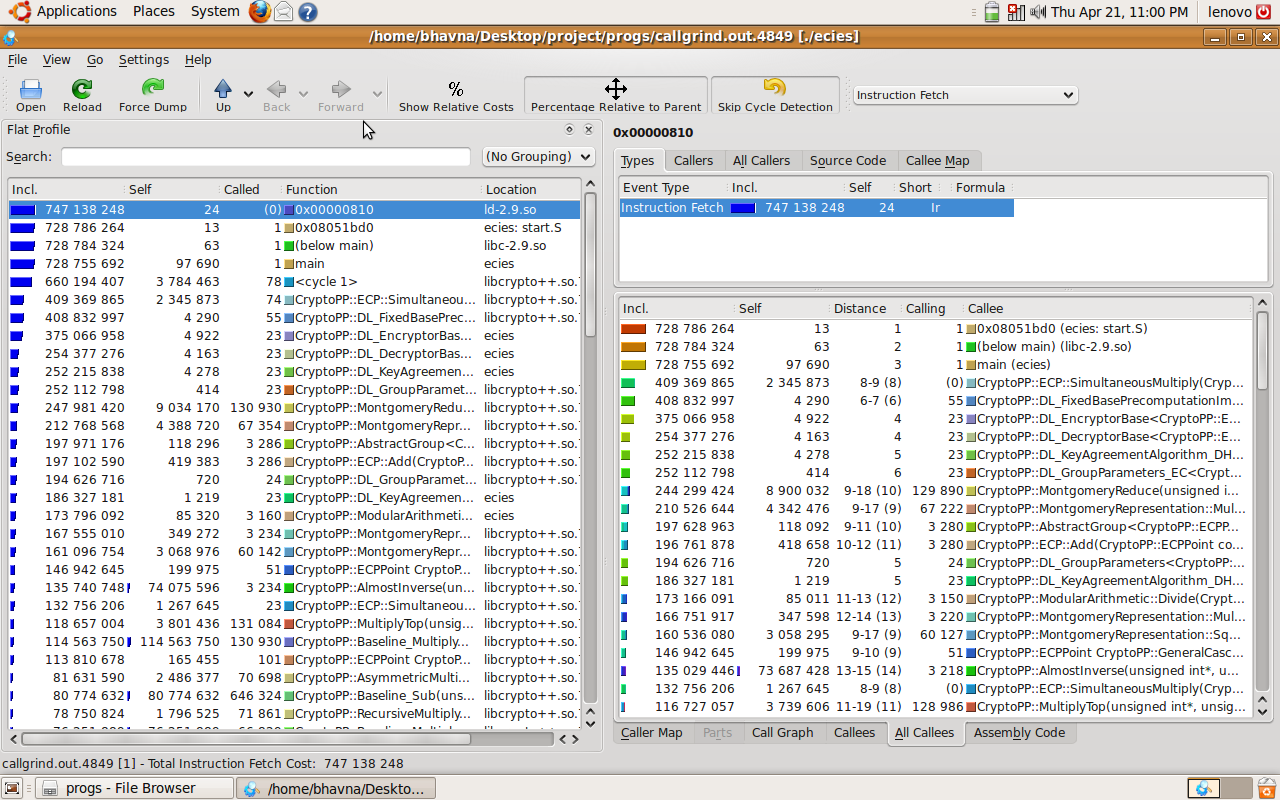
**Table 5.9: Total Instruction Fetch Cost for ECIES, ECDSA and ECDH**

|  |  |
| --- | --- |
| Algorithm | Total Instruction Fetch Cost |
| ECIES | 747 138 248 |
| ECDSA | 688 745 567 |
| ECDH | 52 746 600 |

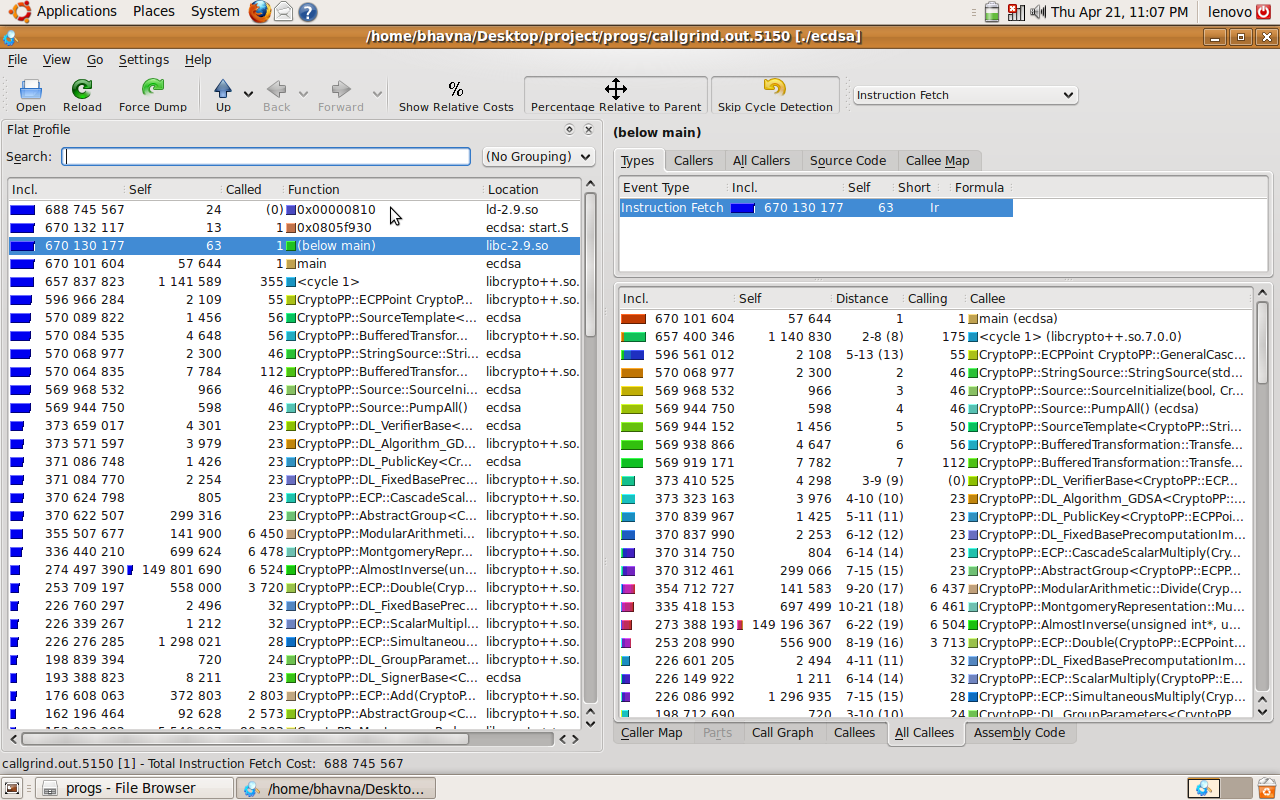
**Figure 5.12: Graph of Total Instruction Fetch Cost of ECIES and ECDSA**

Reason for the above results obtained is as follows:

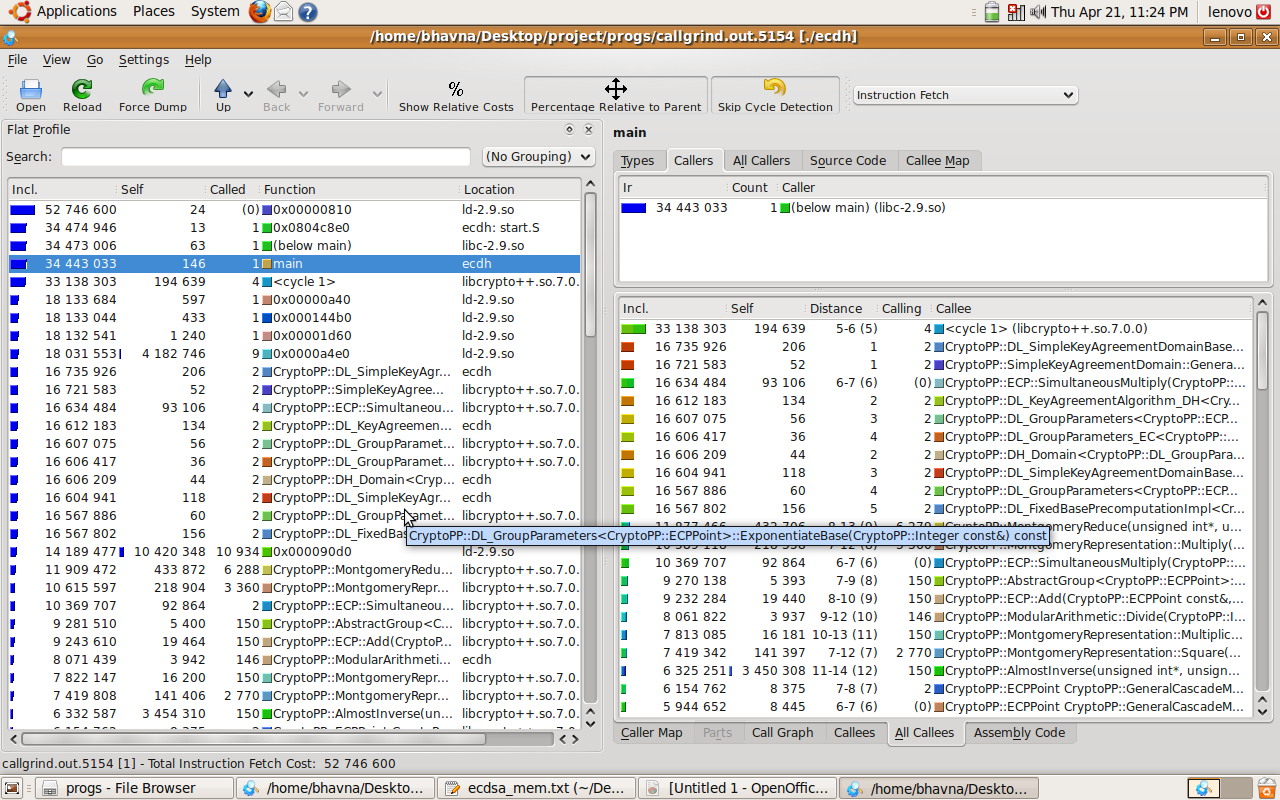
|  |  |  |
| --- | --- | --- |
| PROGRAM | FUNCTION | INSTRUCTION FETCH COST |
| ECIES | Encryptor.Encrypt() | 375 066 958 |
| Decryptor.Decrypt() | 254 377 276 |
| EDSSA | Signer.SignandRestart() | 193 260 215 |
| Verifer.VerifyandRestart() | 373 410 525 |

****

**Figure 5.13: Screenshot of results obtained in Kcachegrind for ECIES**

****

**Figure 5.14: Screenshot of results obtained in Kcachegrind for ECDSA**

****

**Figure 5.15: Screenshot of results obtained in Kcachegrind for ECDH**

In time, memory and power constrained applications, and where authentication is of prime importance for e.g. cellular telephony, access control applications like web-based result, ATMs, physical entry to restricted areas or border crossings, ECDSA should be preferred over ECIES.

In applications where secrecy of communication is of prime importance, in comparison to time, memory and powerconsumed, for e.g. PIN no., credit card no. , defence communications, ECIES should be preferred.

**CHAPTER 6**

**CONCLUSION AND FUTURE WORK**

**Conclusion**

After studying and testing various ECC libraries; we found Crypto++(a C++ library) suitable for our work on the PC platform. Henceforth, we implemented the three ECC algorithms, namely ECDSA, ECIES and ECDH using Crypto++ 5.5.2. Our work progressed further by searching for appropriate profiling tools for runtime, memory and power consumption. After selecting the tools, we analyzed the results and the algorithms for their suitability to different applications, and as per the different constraint environments.

**Future Work**

The work done in this paper could be extended further. Firstly, extending each of the ECC algorithms viz. ECDSA, ECIES and ECDH to include the complete set of security features i.e. Confidentiality, Integrity and Authentication. For e.g. ECDSA (which provides authentication) could be extended to provide secrecy. Secondly, improvement could be made on the individual algorithms in terms of performance and memory. Also improvised algorithms and their results could be used in Wireless Sensor Networks, where ECC algorithms hold prime importance in providing security in a memory and power constrained environment. Another improvement could be cross language implementation of ECC algorithms utilizing the capabilities and features of other languages like Java. Also, the performance, integration cost and overhead of inter-processor communication for ECC algorithms could be improved.

**REFERENCES**

1. An Liu and Peng Ning, *TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks* in Proceedings of the 7th International conference on Information processing in sensor networks, 2008
2. Barry Doyle et al., *Security Considerations and Key Negotiation Techniques for Power Constrained Sensor Networks* in The Computer Journal Advance Access, May 2006
3. Hai Yan and Zhijie Jerry Shi, *Studying Software Implementations of Elliptic Curve Cryptography* in Proceedings of ITNG 2006, April 2006.
4. Craig Gentry, *Computing Arbitary Functions of Encrypted Data* in Communications of the ACM, Volume 53, No. 3, March 2010.
5. T. Kavitha and D. Sridharan, *Security Vulnerabilities In Wireless Sensor Networks: A Survey* in Journal of Information Assurance and Security, May 2010
6. Anoop MS, *Elliptic Curve Cryptography, An Implementation Guide*, January 2007,

Available: http://hosteddocs.ittoolbox.com/AN1.5.07.pdf

1. “<http://en.wikipedia.org/wiki/Elliptic_curve_cryptography>”. Last Accessed: 29th November, 2010
2. An Liu, *Readme for TinyECC Version 1.0*, November 2007. Available: <http://discovery.csc.ncsu.edu/software/TinyECC/README.txt>
3. directory.fsf.org/project/libecc/
4. Daniel R. L. Brown, *Certicom Research Standards for Efficient Cryptography – SEC 1: Elliptic curve cryptography, Version 1.0*, September 2000. Available: “http://www.secg.org/download/aid-385/sec1\_final.pdf”.
5. “<http://en.wikipedia.org/wiki/Crypto%2B%2B>”. Last Accessed: 29th November, 2010
6. Brian J. Gough and Richard M. Stallman, [*An Introduction to GCC - for the GNU compilers gcc and g++*](http://www.network-theory.co.uk/gcc/intro/), Network Theory Limited, March 2004, Available:“http://www.network-theory.co.uk/docs/gccintro/gccintro\_80.html”.
7. “http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html”. Last Accessed: 29th November, 2010
8. “http://en.wikipedia.org/wiki/DTrace”. Last Accessed: 29th November, 2010
9. “<http://oprofile.sourceforge.net/about/>”. Last Accessed: 29th November, 2010
10. “<http://en.wikipedia.org/wiki/ECIES>”. Last Accessed: 29th November, 2010
11. “<http://en.wikipedia.org/wiki/Profiling_%28computer_programming%29>”. Last Accessed: 29th November, 2010
12. “http://linux.die.net/man/3/gettimeofday”. Last Accessed: 8th May 2011
13. “http://valgrind.org/”. Last Accessed: 8th May 2011
14. “http://valgrind.org/info/tools.html”. Last Accessed: 8th May 2011

**LIST OF FIGURES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIGURE NO.** | | **TOPIC** | **PAGE NO.** | |
| 4.1 | Block Diagram for methodology of evaluation .................................................. | | | 21 |
| 4.2 | Block Diagram for investigation of ECC libraries ……..................................... | | | 22 |
| 4.3 | Block Diagram for implementation of ECC algorithms .................................... | | | 23 |
| 4.4 | Block diagram for investigation of profiling tool …………………………….. | | | 24 |
| 4.5 | Block diagram for profiling of ECC algorithm………………………………... | | | 25 |
| 4.6 | Block diagram for ECIES ……………………………………………………. | | | 30 |
| 4.7 | Block diagram for ECDSA ……………………………………………………. | | | 31 |
| 4.8 | Block diagram for ECDH ……………………………………………………. | | | 32 |
| 6.1 | Graph of Runtime (microseconds) of ECIES and ECDSA | | | 37 |
| 5.2 | Plot of sample readings of runtime of ECIES ………………………………… | | | 39 |
| 5.3 | Plot of sample readings of runtime of ECDSA ………………………………. | | | 41 |
| 5.4 | Plot of sample readings of runtime of ECDH ………………………………… | | | 43 |
| 5.5 | Graph of Heap memory Consumption (bytes) of ECIES and ECDSA ……….. | | | 44 |
| 5.6 | Plot of Sample Readings for heap memory consumed (bytes) by ECIES ……. | | | 45 |
| 5.7 | Screenshot of result obtained by msprint command in massif tool for ECIES .. | | | 46 |
| 5.8 | Plot of Sample Readings for heap memory consumed (bytes) by ECDSA …... | | | 47 |
| 5.9 | Screenshot of result obtained by msprint command in massif tool for ECDSA | | | 48 |
| 5.10 | Plot of Sample Readings for heap memory consumed (bytes) by ECDH ……. | | | 49 |
| 5.11 | Screenshot of result obtained by msprint command in massif tool for ECDH .. | | | 50 |
| 5.12 | Graph of Total Instruction Fetch Cost of ECIES and ECDSA ………………... | | | 51 |
| 5.13 | Screenshot of results obtained in Kcachegrind for ECIES …………………… | | | 52 |
| 5.14 | Screenshot of results obtained in Kcachegrind for ECIES …………………… | | | 53 |
| 5.15 | Screenshot of results obtained in Kcachegrind for ECIES …………………… | | | 54 |

**LIST OF TABLES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE NO.** | | **TOPIC** | **PAGE NO.** | |
| 5.1 | Runtime (microseconds) for ECIES, ECDSA and ECDH ………………………... | | | 37 |
| 5.2 | Sample readings for Runtime of ECIES ………………………………………….. | | | 38 |
| 5.3 | Sample readings for Runtime of ECDSA ……………………………………........ | | | 40 |
| 5.4 | Sample readings for Runtime of ECDH …………………………………………... | | | 42 |
| 5.5 | Heap memory consumption (bytes) for ECIES, ECDSA and ECDH …………….. | | | 44 |
| 5.6 | Sample Readings for peak values of heap memory consumed (bytes) by ECIES .. | | | 45 |
| 5.7 | Sample Readings for peak values of heap memory consumed (bytes) by ECDSA . | | | 47 |
| 5.8 | Sample Readings for peak values of heap memory consumed (bytes) by ECDH ... | | | 49 |
| 5.9 | Total Instruction Fetch Cost for ECIES, ECDSA and ECDH …………………….. | | | 51 |

**NOMENCLATURE**

|  |  |
| --- | --- |
| **ABBREVIATION** | **EXPANSION** |
| ECC | Elliptic Curve Cryptography |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| ECIES | Elliptic Curve Integrated Encryption Scheme |
| ECDH | Elliptic Curve Diffie Hellman |
| WSN | Wireless Sensor Networks |
| PKC | Public Key Cryptography |
| ANSI | American National Standard Institute |
| KDF | Key Derivation Function |
| EEA | Elliptic Euclidean Algorithm |
| DOS | Denial Of Service |
| DLIES | Discrete Logarithm Integrated Encryption Scheme |
| MAC | Message Authentication Code |
| SHA | Secure Hash Algorithm |
| ECNR | Elliptic Curve Nyberg Rueppel |
| ECMQV | Elliptic Curve Menezes–Qu–Vanstone |
| RTS | Request To Send |
| CTS | Clear To Send |
| GCC | GNU Compiler Collection |
| IDE | Integrated Development Environment |
| GNU GPL | GNU General Public License |