Performance Measurement and Workflow Impact of Securing Medical Data Using HIPAA Compliant Encryption in a .NET Environment

---

A Thesis

Presented to

The faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

Of the requirements for the Degree

Master of Science (Computer Science)

By

Andrew Morgan Snyder

August 2003

APPROVAL SHEET

The thesis is submitted in partial fulfillment of the

requirements for the degree of

Master of Science (Computer Science)

_____

Author

This thesis has been read and approved by the examining Committee:

_____

Thesis Advisor

_____

_____

_____

_____

Accepted for the School of Engineering and Applied Science:

_____

Dean, School of Engineering
and Applied Science

August 2003

ABSTRACT

The Health Information Portability and Accountability Act of 1996 called for new standards to be set in the health industry. These standards included new privacy and security laws along with more administrative reform. In the long run, these new standards are supposed to save the healthcare industry time and money. One of the security standards calls for encryption of all digital data in both transmission and storage. The impact of this requirement on the workflow in a healthcare entity is unknown. The Department of Radiology at the University of Virginia is interested in implementing a solution using .NET technologies in order to gain the benefits of portability and security from managed code. Since HIPAA does not require a specific encryption, I first analyze different prominent industrial encryption standards from both a performance and security standpoint and review the literature on encryption algorithm performance. I then recommend one that should be used in a .NET healthcare environment. I use the performance statistics in a workflow model used by the Department of Radiology at the University of Virginia. From the results, I determine the possible impact on the workflow at the University of Virginia hospital given different concurrency in their systems.

Performance Measurement and Workflow Impact of Securing Medical Data Using HIPAA Compliant Encryption in a .NET Environment

Chapter 1 – HIPAA

1.1 Introduction

In 1996, the U.S. Congress signed into law the Health Insurance Portability and Accountability Act (HIPAA) to initiate the process of healthcare reform. HIPPA is divided into two separate sections: Health Insurance Reform (Title I) protects health insurance coverage for workers and their families when they change or lose their jobs; Administrative Simplification (Title II) authorizes the U.S. Department of Health and Human Services (HSS) to establish national standards for electronic health care transactions and national identifiers for providers, health plans, and employers. During the period 1996-2002, HSS developed and promulgated a series of reforms aimed at improving the effectiveness and efficiency of the national healthcare system. The "Electronic Health Care Transactions and Code Sets" rule seeks to standardize the descriptions of medical procedures and drugs, and is a required enabler for electronic filing of insurance claims and Medicare reimbursements; these regulations go into effect October 16, 2003. The "Employer Identifier Standard," an eight-character identification code, uniquely identifies a healthcare provider in a national database; this code must be used after July 30, 2004.

HIPAA's most ambitious requirements, however, are embodied in the so-called Privacy and Security Rules. Effective April 14, 2003, for most covered entities (and April 14, 2004, for everyone), the Privacy Rule seeks to ensure patient privacy by regulating how doctors, hospitals, healthcare plans, insurance companies, and other covered entities collect, manage, store, disclose, and utilize a patient's medical information. The Security Rule standards, effective April 21, 2005, cover administrative procedures, physical safeguards, technical security services, and technical security mechanisms. The security services cover access control, audit control, authorization control, data authentication, and entity authentication. The security mechanisms guard against unauthorized access to data by requiring integrity controls and message authentication, by requiring access controls and/or encryption, and, if medical data is transmitted over a network (which is increasingly common), by requiring alarm reporting, audit trails, entity authentication, and event reporting.

As laudable as these overall goals may be, it is their implementation that could subject the healthcare community to initial problems involving performance and security. For example, the University of Virginia Medical Center is a "covered entity" under the HIPAA definitions and thus is obligated to comply with its privacy and security regulations. This in turn implies that all of the hospital's medical records, which are routinely exchanged over computer networks, are subject to the audit control and encryption requirements mandated for data security. The hospital's electronic patient record includes all diagnostic imagery acquired by the Department of Radiology, which conducts over 380,000 examinations and generates around 9 TB

of data annually. While encrypting and decrypting a few digitized x-rays will probably cause no workflow problems, no one has investigated the potential workflow disruption that might result from having to first decrypt the 500 to 1000 separate images that comprise a modern computerized tomography (CT) or magnetic resonance (MR) examination. One goal of this thesis is to conduct that investigation and to then determine (a) which of the allowable encryption methods are preferable and (b) what the performance cost of encryption will be for the computers that implement them, and (c) the resulting impact on the hospital's patient workflow.

1.2 HIPAA Law

HIPAA was passed "To amend the Internal Revenue Code of 1986 to improve portability and continuity of health insurance coverage in the group and individual markets, to combat waste, fraud, and abuse in health insurance and health care delivery, to promote the use of medical savings accounts, to improve access to long-term care services and coverage, to simplify the administration of health insurance, and for other purposes."[1] The law requires health plans, health care clearinghouses, and those health care providers who conduct certain financial and administrative transactions electronically (such as eligibility, referral authorizations and claims) to comply with each set of final standards. This law was passed because there are numerous overlapping "standards" in the healthcare industry. Many healthcare providers have processes that are specific to them and they require all patients' data to be in their proprietary formats. Problems arise when data needs to be transferred to different locations. Since there is no single primary standard, data transfer becomes a

large issue. If a patient at one hospital is treated at another, and if the two hospitals have different formats for storing data, it would be difficult for one hospital to find the data it needs to serve the patient since it does not know which format the other hospital uses. If there were a single standard that all providers adhered to, it would allow the industry to move forward as a whole. The law does not focus on standards for every possible format and transaction, so some data is still going to be stored differently at varying locations. However, when dealing with transactions for which there are rules, then every provider will follow the same rules.

These rules were not just created by the DHHS and set in stone. Proposed rules are first given to the industry with the desire for feedback. Each proposed rule set is given about a two-year period for comments from the healthcare community. After that time, the DHHS sets forth the final collection of rules for the set. Once the "final standard" is published, covered entities have two years to become compliant. Small health plans have three years to become compliant. The electronic transaction standards "final rules" were published August 17, 2000. Most entities were supposed to become compliant October 16th, 2002. Since few of the entities were prepared, Congress passed a one-year extension for those entities. This does not affect the small health plan compliance schedule of October 16th, 2003. The privacy rule standards "final rules" had a compliance date of April 14th, 2003. Most healthcare entities are simply not ready.

The Centers for Medicare & Medicaid Services (CMS) will be responsible for enforcing the transaction and code set standards. The HHS Office for Civil Rights (OCR) will enforce the HIPAA privacy standards. Enforcement activities will focus

on obtaining voluntary compliance through technical assistance. The process will be primarily complaint driven and will consist of progressive steps that will provide opportunities to demonstrate compliance or submit a corrective action plan.

Patients have the right to file a formal complaint with the U.S. Department of Health and Human Services (DHHS) if they believe a covered entity has violated HIPAA requirements. DHHS has the authority to investigate and penalize covered entities. There are civil and criminal penalties associated with HIPAA noncompliance. The civil penalties are $100 per violation, up to $25,000 per person, per year for each requirement or prohibition violated. The criminal penalties are for knowingly violating patient privacy. There is a fine of up to $50,000 and one year in prison for obtaining or disclosing protected information. There is a fine of up to $100,000 and up to five years in prison for obtaining or disclosing protected information under false pretenses. There is also a punishment of up to $250,000 and up to 10 years in prison for obtaining or disclosing protected information with the intent to sell, transfer, or use it for personal gain, malicious harm, or commercial advantage such as spam. These categories mainly aim to curb the purposeful disclosure of patient information.

1.3 HIPAA Requirements

There are four basic sections of HIPAA: Transaction and Code Set standards, Privacy standards, Security standards, and Identifier standards. Each of these sections has its own deadline for compliance and timeline for acceptance. The Transaction and Code Set standards contain rules that establish standard data content,

codes and formats for submitting electronic claims and other administrative health care transactions. By encouraging the greater use of electronic transactions and the elimination of inefficient paper forms, these standards are expected to provide a net savings to the health care industry of $29.9 billion over 10 years[2]. All health care providers will be able to use the electronic format to bill for their services, and all health plans will be required to accept these standard electronic claims, referral authorizations and other transactions.

The Identifier standards are a group of proposed rules to allow unique identification of health plans, employers, doctors, hospitals, nursing homes and other health providers. This would allow easier communication between entities when accomplishing such activities as filing claims for insurance. Today, most health providers are given multiple identity numbers by different organizations, such as hospitals, nursing homes, and insurance companies. This results in events such as slow payment and lack of synchronization.

The Privacy standards focus on providing more confidentiality to the patient. It places more restrictions on the disclosure of patient data and requires that logs be kept on when and to whom patient data is distributed. Any kind of patient data given out to a third party must either have the permission of the patient, be for the purpose of TPO (Treatment, Payment or the Organization), or be stripped of all demographic and personal data (anything that can link a person to the data); this latter category is intended to enable research analysis of collected data. Any illegal disclosure of data is viewed as a violation of HIPAA and is subject to criminal and civil sanctions. This

makes it more difficult for research to be done in areas where the demographic information itself could identify the person.

The last of the four sections is Security standards. This thesis will be focusing on this area in detail. These standards focus on the actual security of the data in a health care system. There are three different sections in the proposed security standards. The first section is administrative procedures to guard data integrity, confidentiality, and availability. This area focuses on requirements such as internal audit, security incident procedures, and training. The requirements have associated implementations. For instance, security configuration management must have documentation, hardware/software installation and maintenance, inventory, security testing and virus checking.

The second section is physical safeguards. This section deals with such requirements as media controls (data backup, data storage), physical access controls (sign-in for visitors, facility security plan), and policy/guideline on workstation use. The third section covers technical safeguards. This section contains technical security services to guard data integrity, confidentiality, and availability. This section has the following requirements: access control, audit controls, data authentication, transmission security, and entity authentication. Access control must have a procedure for emergency access and either context-based access, role-based access or user-based access. According to HIPAA, encryption is optional when dealing with a closed network. However, if information is going to be passed over an open network, then encryption must be utilized. The audit controls would be used to monitor suspect data access activities, assess its security program and respond to potential

weaknesses. This can be implemented in any manner, as long as it is present. Data authentication is required in order for each entity to be able to prove that no data has been altered in any unauthorized manner. This can be implemented using such measures as checksums or a digital signature. Principal authentication revolves around proving whether someone who is accessing the system is who he or she claims to be. There must be automatic logoff, unique user identification, and at minimum one verification utility, such as a password or biometric confirmation. In order to comply with the transmission security, one must have integrity control and encryption. If a network is used for communications then alarms, audit trails, entity authentication and event reporting must be used. Although HIPAA states that encryption is not mandatory, it does require encryption whenever data is transmitted or available over an open network (e.g., Internet). This thesis deals primarily with the last section that deals with the technology side of data security during storage and transmission. Becoming compliant with the HIPAA encryption rule requires more than simply using an appropriate algorithm. There is other paperwork that must be maintained, such as a risk analysis. For the purposes of this thesis, HIPAA compliant encryption refers solely to encryption algorithms that are appropriate for fulfilling the technical side of rules.

1.4 Impact

HIPAA is such a conglomerate of rules and standards that it impacts all health providers, ranging from clinics to hospitals to health plans. Each entity will be impacted in different ways by the new sets of rules. Very small practices will have

an easier time becoming compliant with the new rules, as they have less to change. Having to use new standard forms will be simpler for a one-person practice than a whole hospital. Adding security to a network with no link to the Internet will be easier than securing an open network. But at the same time, a smaller practice has fewer resources to allocate to becoming compliant, which could adversely affect its timeline. Hospitals will no longer be able to give out patient data to third parties, either for monetary gain or research. This increase in privacy is immense for a patient, but what exactly will it do to a hospital? Having to encrypt 500-1000 images for every MRI will take longer than just storing the MRI, and so how will that affect workflow? Will there be new bottlenecks? The hospital and the Department of Radiology do not have any studies to show them what performance degradation they might face given encryption and are therefore concerned. They are interested in using .NET technologies to maintain a safe and flexible environment. When Christopher Reeve (the actor who played Superman) was injured while horseback riding, he was transported to the U.Va. neurosurgery unit for surgery. All of his medical data was at risk of release because reporters (e.g., National Enquirer) and hackers were trying to get it. All of his records and images were on paper and film, thus making physical security the brunt of the data's security. If HIPAA had already been in place, the hospital would have been prepared and better able to thwart any attempt to access data without authorization.

1.5 Our Research Project

Recognizing that Microsoft's .NET XML web services offer a promising framework for developing next-generation distributed healthcare solutions, our Internet Commerce Group and the U.Va. Department of Radiology wrote a joint proposal to Microsoft to investigate how to design a prototype system that meets the needs of the U.Va. Medical Center while simultaneously satisfying HIPAA's objectives. That proposal is based upon the concept of "Federated Trust Systems"[3] and has now been funded. The overall research project (of which this thesis is one component) embraces these issues:

(1) Today's healthcare IT infrastructure is fragmented. Doctors typically use a Hospital Information System (HIS) to view patient data; radiologists use a Radiological Information System (RIS) to store and retrieve diagnostic images; administrative staff use PCs for scheduling and billing. Although these systems are physically distinct, they need to be logically integrated to avoid lost data, improve workflow, enhance patient satisfaction, and reduce cost. Our solution is to use a web services approach that hides the underlying implementation. In our prototype, doctors, patients, and administrators all utilize a common medical portal to gain access to data and services.

(2) Today's medical professional uses multiple devices to access and record data. EKGs can be read on a Pocket PC. Tablet PCs can be used to capture hard-written case notes, and then transcribe them into the medical record. Prescriptions can be

dictated on a cell phone, and voice recognition software can update the patient record and electronically transmit a prescription to a pharmacy. Our prototype supports wireless PDAs and Tablet PCs in addition to traditional wired desktop/laptop access. In the future we will add support for cell phones.

(3) Reliable authentication is essential. Biometric devices (e.g., fingerprint and iris scanners), smartcards, ID badges with radio transmitters, and similar techniques will be needed to achieve the level of privacy and security demanded by HIPAA. The authentication service of our federated security system, building upon Passport.NET, must be capable of working with multiple identification technologies. We currently support fingerprints and iris scans for biometric identification.

(4) Authorization rules must be dynamic and context-dependent. Staff come and go and get reassigned; their authorization privileges change as a result. A request to alter a medical record might be subjected to differing levels of trust depending upon whether it came from inside the hospital or from an external mobile device. Requests for data access come from programs as well as humans. To handle this complexity, we are developing a programmable rule engine that uses policies to determine what data accesses are permitted and under what circumstances, all following WS-Policy and other GXA security specifications.

(5) External interfaces are required. The healthcare IT infrastructure must extend beyond the medical center into the real world of external participants, e.g., pharmacies, insurance companies, and other healthcare providers. This introduces

the concept of federated trust systems, and we are designing solutions for sharing trust among disparate entities.

(6) All data must be secured via encryption, and we must understand what impact this requirement will have on the Radiology department's workflow. These two topics are the focus of this thesis. The other topics are being addressed in parallel by other members of our research group.

1.6 Approach

First, I analyzed the different recommended encryption standards. I determined which algorithms are more mathematically and computationally secure and then implemented the standards in a programming environment, so as to discern which methods are the fastest in a real-world application. I made comparisons on these algorithms based on security and speed. The results from the mathematical analysis and speed tests are quantitative. I used the gathered data as input to a Department of Radiology model to determine what kind of delay the encryption methods would cause in the overall scheme of a hospital's workflow. This allowed me to make recommendations on encryption algorithms. The encryption application was created in Microsoft Visual Studio .NET. By using this framework, all services can be programmed in different languages or used on different operating platforms without any additional programming required. By making the solution modular, it increases the independent development of services. Encryption algorithms can be easily switched at any time due to the known service interfaces.

1.7 Goals

One of the many goals of this thesis is to broaden public knowledge of HIPAA and how it will affect healthcare. The short synopsis of HIPAA should provide one with enough background information to appreciate it. This thesis will result in recommendations for compliance with the HIPAA security standards with its basis in reasoning, experimental and technical knowledge. It will evaluate alternative methods for security compliance and provide recommendations for those methods. Using the quantitative results obtained experimentally, I will comment on the predicted workflow impact at the University of Virginia hospital. The workflow impact will help determine whether there is a need for a trade-off between security and performance when choosing an encryption method in order to maintain an uninterrupted workflow.

Chapter 2 – Encryption Algorithms

2.1 Introduction

There are hundreds of possible encryption methods that could be used to secure medical data. The four I will consider offer differed advantages. DES (Data Encryption Standard) was chosen because of its 20 years of security and speed. It was long the standard by which all commercial encryptions were done. The algorithm has implemented in hardware for even faster encryption and decryption. 3DES (Triple-DES) was chosen because it is the successor to DES and it maintains backward compatibility with DES. However, it is up to three times slower than DES. AES (Advanced Encryption Standard) is the newest NIST approved encryption standard. It runs using different sized keys, and in some implementations runs faster than DES. RSA, named for its authors, has security bounded only by the chosen key size. The large key size, and the fact that the Department of Radiology was interested in this algorithm is why it was chosen. Its security lies in the difficulty of factoring large numbers. Each of these algorithms offers something different, but our emphasis will be weighing security versus speed and its impact on workflow in a healthcare environment.

2.2 Encryption - DES

In 1971, IBM created an encryption algorithm called Lucifer.[4] It was initially an encryption algorithm for Lloyds of London for use in a cash-dispensing system. Shortly after its creation, the National Bureau of Standards (NBS) issued a request for proposals for encryption algorithms that could be a national cipher standard. IBM

submitted Lucifer, and after analysis (and some modifications) by the National Security Agency (NSA), it was accepted as the national standard in 1977 and named DES, the Data Encryption Standard.[3]

Initially there were two technical issues regarding the algorithm. First, the key was only 56 bits, whereas the key in Lucifer was originally 128 bits. This huge difference (a modification suggested by NSA) suggested that DES could be broken by brute force given NSA's resources. With a 56 bit key, there are 2^{56} ($\sim 7x10^{16}$) possible keys. This means that a computer, trying one key every second, would take over 2 billion years to try every key. In 1977 this seemed strong enough to convince the NBS that DES was adequately secure for commercial transactions. However, given the vastly improved technology of 1999, a distributed attack on DES using 100,000 computers was able to break the code in just over 22 hours[5]. So just how secure is DES in 2003? Some say the NSA/FBI has chips that can break DES in a matter of hours or even faster[6].

The second issue with DES was with a section of the algorithm that uses substitution boxes (S-Boxes). The internals of these S-Boxes were not released, and it was believed that the NSA might have chosen values to be used internally to allow for quick back-door decryption. It is now more widely accepted that the NSA chose S-Box modifications that made DES more resistant to differential cryptanalysis, which is a technique that was unknown to the public at that time.

The algorithm overview can be seen in figure 1. DES takes a 64-bit input block and performs an initial permutation on it to rearrange the bits; this permutation is

identical for every input block. After the initial permutation, sixteen rounds of key-dependent functions are performed. At the conclusion of the sixteenth round, the

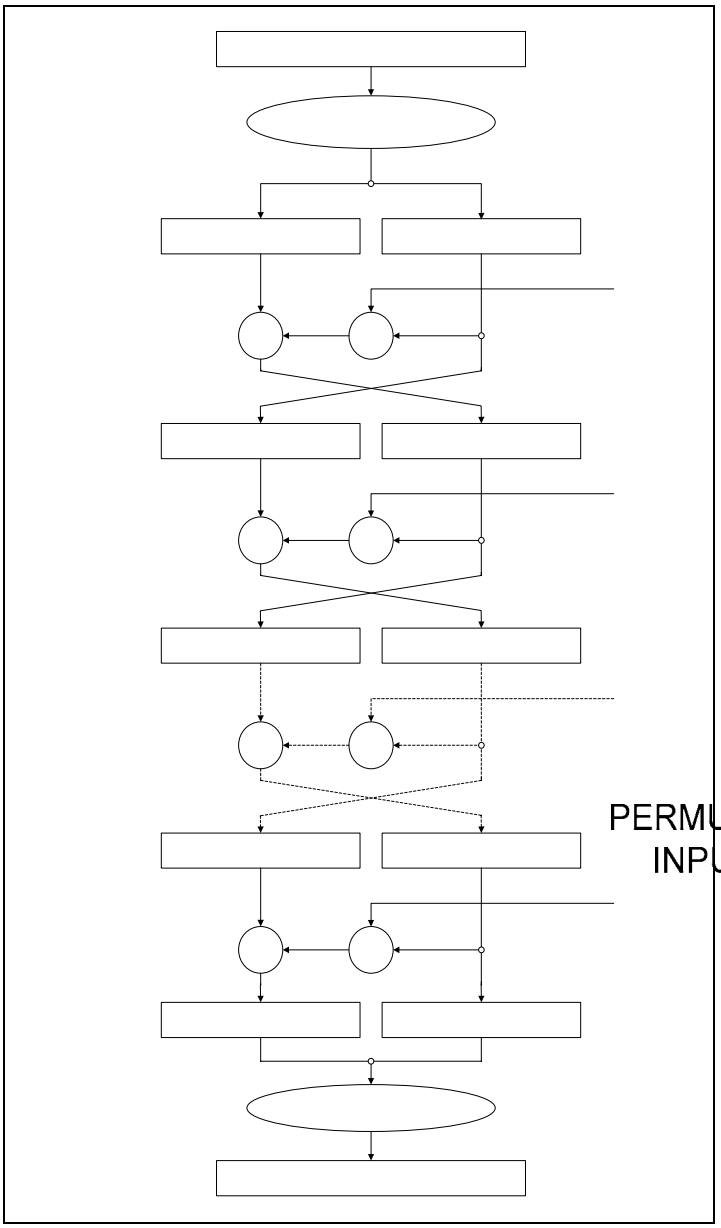

Figure 1 – Overview of DES Algorithm[7]

output is permuted again, using the inverse of the initial permutation. The result of that permutation is the ciphertext output of DES for the original input block.

Each of the sixteen rounds is identical with regard to the operations performed. This makes implementation in hardware much easier. After the initial permutation, the input block is divided in half. The right half of the DES algorithm in figure 1 is where most of the computation is done. The left half is quite simplistic in computation. L_1 comes directly from R_0, and L_2 comes directly from R_1, etc. without any work. The right half requires two computations to become the next round's left half. First, R_0 and the round-specific key (K_1 in this case) are inputs to a function, *f*, which is shown in figure 2.

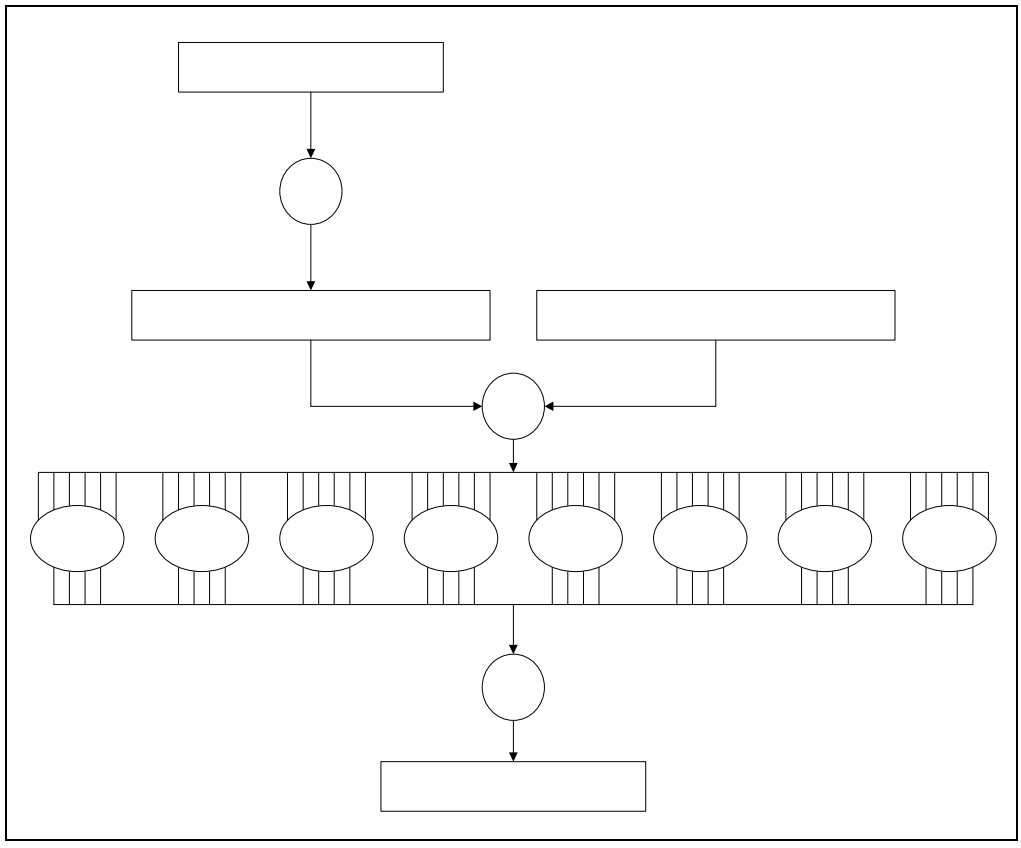

Figure 2 – Detailed view of the *f* function[6]

The function *f* takes a 32-bit input block and a 48-bit key and creates a 32-bit output. The input block is subjected to an expanding function, which takes the 32-bit input block and creates a 48-bit block by selective duplication of 16 bits. The

R (32 BITS)

E

expansion function is identical in every round. The resulting expanded block is added (bitwise modulo 2) to the key. This result is then put through eight substitution boxes (S-Boxes). Each of these S-Boxes takes six bits as input, and creates a 4-bit output. This is how a 48-bit input becomes a 32-bit output. Each of the S-Boxes takes a different six bits of the pre-S-Box block and uses them as row and column indexes into an S-Box-specific table. The resulting value found in the table is then output as a 4-bit value. After the S-Box substitution, we are left with a 32-bit output, which is the result of the function *f*. This result is then added (bitwise modulo 2) to the L_n of that round. For example, $R_1=L_0+f(R_0,K_1)$. This can be generalized to $R_n=L_{n-1}+f(R_{n-1},K_n)$.

In each of the sixteen rounds, a round-specific key is used (Figure 3). While the initial key is 64 bits, eight of those bits are used as parity checks, and so the effective key length is 56 bits. The algorithm does not simply use the same key every round, but rather performs a varying number of shifts and two permutations. The 64-bit key first goes through a permutation which discards the eight parity bits. The resulting 56-bit output is then split in half. Each half then either undergoes a single or double left circular shift. The 1st, 2nd, 9th, and 16th rounds use a single shift, and all others use a double shift. After the shift is applied, the resulting two blocks are saved and re-combined and subjected to another permutation, which selects 48 of the 56 bits. This becomes the round-specific key. For each subsequent round, the previous round's saved blocks are shifted appropriately, saved again, and permuted. For example,

K1=P(C1,D1), where P is the permutation function. To generalize, $K_n = P(C_n,D_n)$, where $C_n=Shift(C_{n-1})$ and $D_n=Shift(D_{n-1})$.

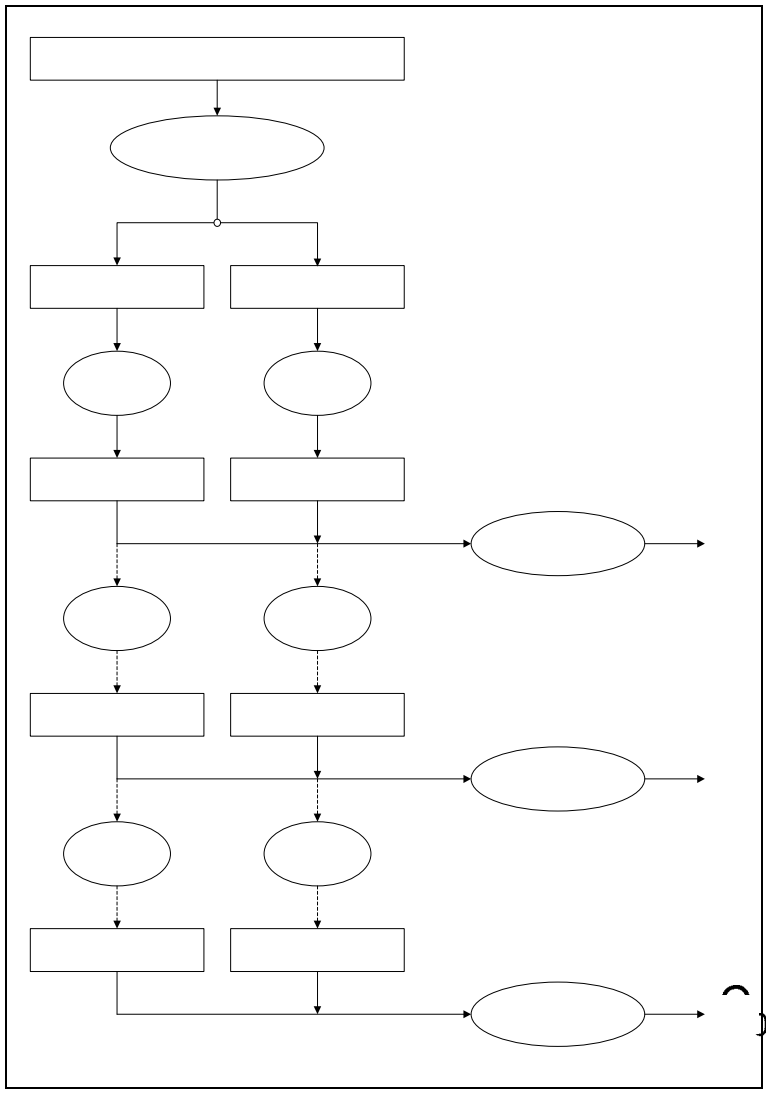

Figure 3 – Key generation algorithm[6]

The decryption is accomplished by using the same algorithm, but generating the keys in the reverse order. This makes the implementation extremely simple, and the hardware implementation fast.

DES does have a few known weaknesses. The first weakness is the small keyspace which leads to successful brute force attacks on this algorithm. There is also a set of "weak keys" which, when used, result in the output exhibiting a strange characteristic[8]. These are keys such as all zeros and all ones, for which each round key will be identical due to no bit stirring from the circular shifts. If a weak-key encrypts a block twice, the output will appear unencrypted. There are also several semi-weak keys. These are keys that come in pairs where $E_{k1}(E_{k2}(m)) = m$. That is, encrypting a message with one of the semi-weak keys, and then encrypting it again with the partner semi-weak key, will effectively nullify the encryption.

2.3 Encryption – 3DES

Triple DES (3DES) was brought about because of the desire for a standard encryption with a larger key size. At a high level, 3DES simply runs DES three times in succession. The idea of 3DES is to have three keys: k_1, k_2, and k_3. First, the message is encrypted using k_1. Then, the ciphertext is "decrypted" using k_2. Since in most cases k_1 is different than k_2, the result will still be ciphertext. This result is then encrypted with k_3. This yields a generalized equation of $E_{k3}(D_{k2}(E_{k1}(m)))$. Since each key is 56 bits, this procedure yields an effective key size of 168 bits. Because 3DES is just DES run three times, it is three times slower than DES. There are three modes of choosing the keys: (1) all keys may be unique, (2) k_1 and k_2 are independent, but $k_3=k_1$, and (3) $k_1=k_2=k_3$. Choice 3 yields $E_{k1}(D_{k1}(E_{k1}(m)))$, which reduces to $E_{k1}(m)$, which is just DES! This mode of choosing keys allows 3DES to be backward compatible with DES hardware.

The decryption of 3DES is done by $D_{k3}(E_{k2}(D_{k1}(C)))$. It can be seen that this is similar to the encryption function with the only difference being in the reverse computation of the round keys. Figure 4 is an example of how the Triple DES algorithm functions.

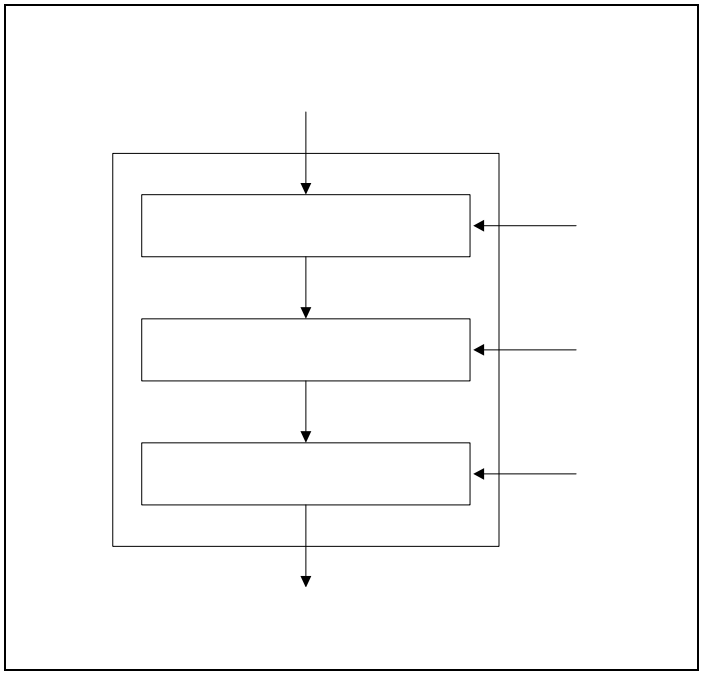

Figure 4 – Overview of 3DES

2.4 Encryption – AES

In 1997, the National Institute of Standards and Technology (NIST) announced a competition for an algorithm that would replace DES. There were fifteen approved submissions, with the final winner being declared in 2001 as the Rijndael (pronounced Rhine-doll) algorithm. The Rijndael algorithm is now known as AES (Advanced Encryption Standard) and became the new encryption standard effective May 26, 2002.

The first benefit of AES is the available keyspace. The AES standard has three different approved key sizes: 128, 192, and 256 bits. While other key sizes are possible, these three are the only ones approved in the standard. Given DES's key size of 56 bits, one can already see that a computer that could solve DES in 1 second would still require anywhere between 150 trillion and $5x10^{52}$ years to solve AES. This difference in key size is large enough to once again make brute force effectively impossible.

AES operates on 128-bit input blocks and has either 10, 12, or 14 rounds (depending on the key size). Figure 5 shows the overview of the AES algorithm.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
Begin

   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])       // Nb = 4 words (128 bits)

   for round = 1 step 1 to Nr-1         // Nr = 10, 12 or 14
       SubBytes(state)
       ShiftRows(state)
       MixColumns(state)
       AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state

end
```

Figure 5 – Code view of the AES algorithm[9]

The first step is to arrange the input block as 16 separate byte segments. The following figure describes the input, state, and output diagrams.

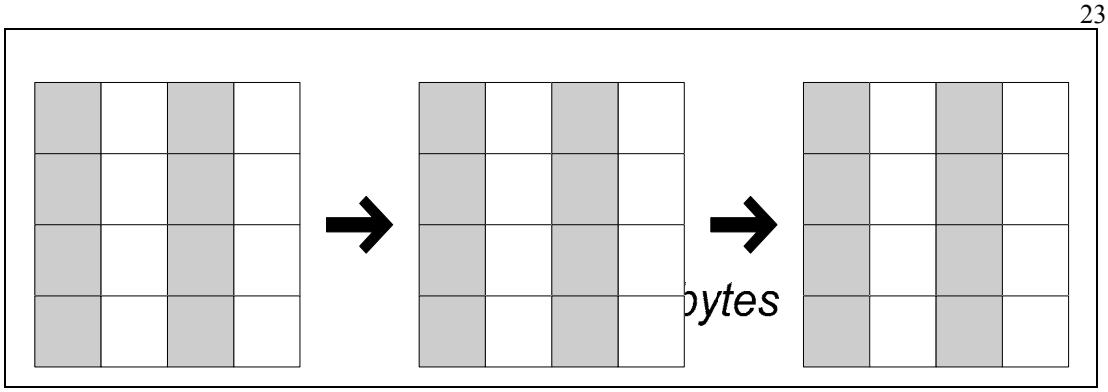

Figure 6 – Required AES data structures[8]

In the beginning of the encryption, a Round Key is XOR'ed with the input. Much like DES, AES uses round-specific keys. Once the Round Key has been XOR'ed, the real "rounds" begin.

The first step is a substitution cipher, in which each box in the State array is used as indexes into a table. The first 4 bits are used as the row index, and the second 4 bits are used as the column index. The resulting value is used to replace the value in the current box. This is the non-linear part of the algorithm that provides most of its security.

The second step in the encryption process is the ShiftRows function. This step left shifts the last three rows of the State array by 1, 2, and 3 places, respectively, as seen by the dark boxes in figure 7.

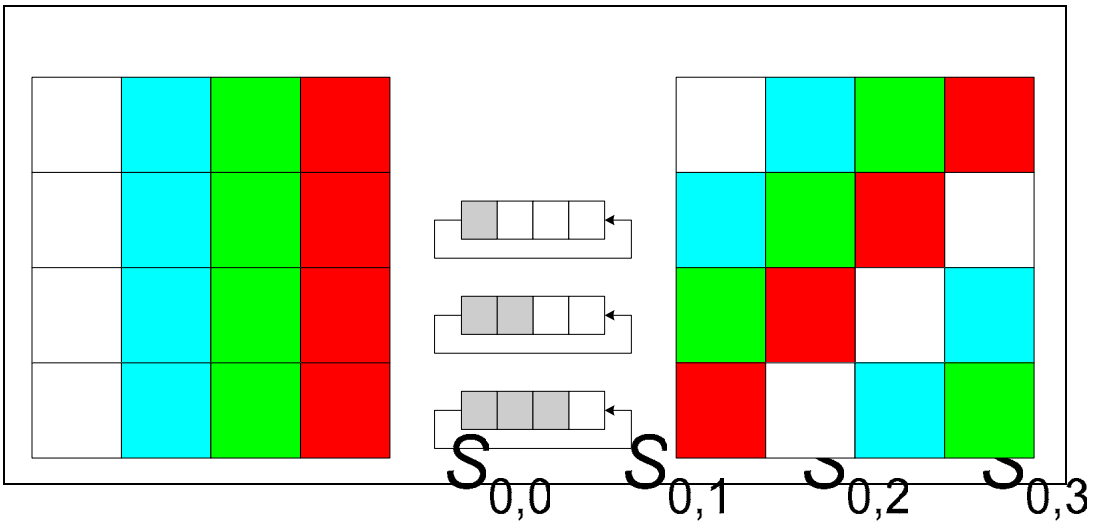

Figure 7 – ShiftRows function details[8]

This has the effect of a left circular shift, thus helping obfuscate the true format of the input block.

The third step in the round is the MixColumns function. This function operates on the State array on a column-by-column basis. It takes each of the four columns in turn and operates on it, resulting in another column that replaces the original. The function treats each column as a four term polynomial, and applies a function to it that involves polynomial reductions. Figures 8 and 9 show how MixColumns takes each column one at a time and what operations are performed on them.

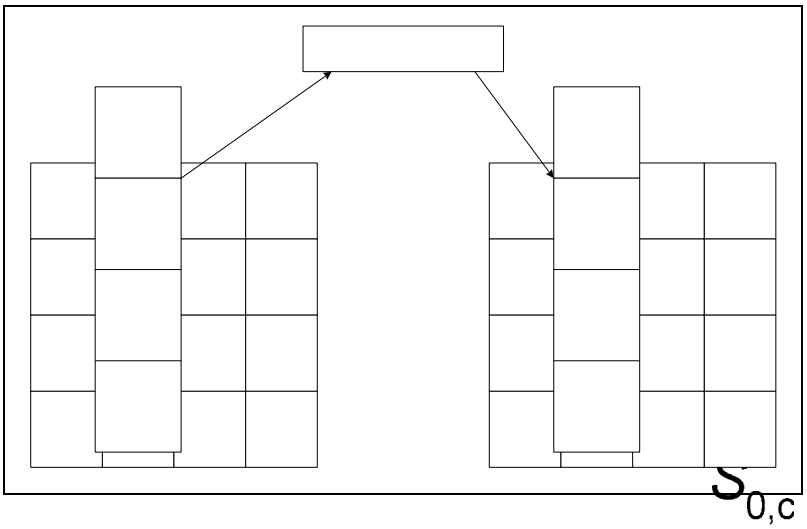

Figure 8 – MixColumns details[8]

$$S'_{0,c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S'_{1,c} = S_{0,c} \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus S_{3,c}$$

$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c})$$

$$S'_{3,c} = (\{03\} \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \bullet S_{3,c})$$

Figure 9 – Functions used in MixColumns[8]

The final step of each round is to XOR a round-specific key with the current state array. The round key is computed using its own set of functions that is applied to the original key. These functions are composed of a RotWord function (which left-shifts the most significant byte of a word), a SubWord function (which performs the same S-Box substitution as the main encryption loop), and an XOR with a round-specific constant. By using these functions in a loop that is executed (Nb*(Nr+1)) times, where Nb equals the number of words in the state array and Nr is the Round for which we want the key, we arrive at Kn (the key for round n).

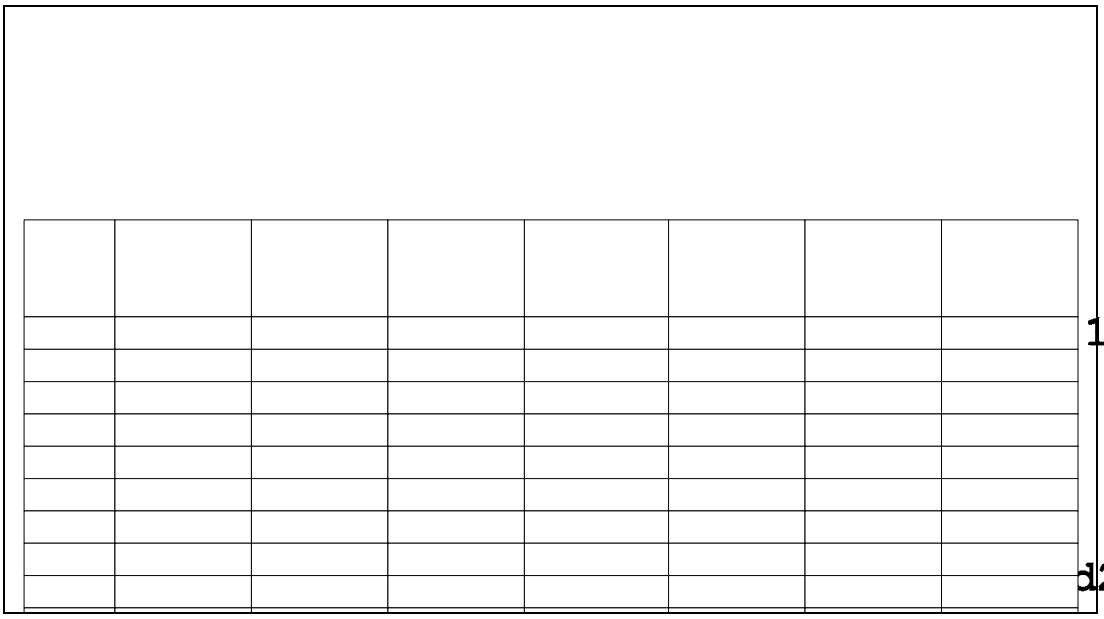

Figure 10 – AES Encryption example[8]

i (dec)	temp	After RotWord()	After SubWor
4	09cf4f3c	cf4f3c09	8a84eb
5	a0fafe17		
6	88542cb1		
7	23a33939		
8	2a6c7605	6c76052a	50386b
9	f2c295f2		
10	7a96b943		
11	5935807a		
12	7359f67f	59f67f73	cb42d2

Figure 10 shows an example of the Round Key computation. Nk is the number of words in the key. Initially, it can be seen that w_0, w_1, w_2, and w_3, which together form the first Round Key, are unmodified quarters of the Cipher Key. The temp value is always the last value from the previous computation. Thus the key used in the current round is always dependant on the key used in the previous round. The RotWord is computed against temp every Nk rounds. Then the SubWord is computed against the result from the RotWord function. That result is then XOR'ed against $[\{02\}^{i/Nk}, \{00\}, \{00\}, \{00\}]$, which is seen as Rcon[i/Nk]. That result is then XOR'ed with the value of w[i-Nk], which can be seen as part of a previous round's key. All of this makes any particular key round specific (but dependant upon all previous keys). The decryption is achieved by a very similar pseudo-code, but each step is the inverse of the original step. The ShiftRows function from the initial

algorithm which left shifted each row by 0-3 places become InverseShiftRows and right shifts each row by 0-3 places, thus undoing what the ShiftRows had done. Each step has its associated inverse step. Once all the rounds are complete, the original bits are recovered.

2.5 Encryption – RSA

RSA, named for its authors Rivest, Shamir, and Adleman, is based upon the believed to be mathematically hard problem of factoring large numbers. The encryption algorithm is simple, $C = M^e \bmod n$. M is the "message" or what is to be encrypted. e is the power to which M will be raised and is computed given a known function. n is directly related to e, but in a computationally difficult way to reverse-engineer. The result of the equation is C, which are the encrypted bits. The decryption is similar: $M = C^d \bmod n$. d in this equation is the partner of e from the encryption algorithm.

The generation of keys starts by choosing two prime numbers, p and q. The larger these prime numbers are, the more difficult it will be to break the encryption. Then the number n is computed using $n=p*q$. Then the values e and d are computed such that $e*d=1 \bmod ((p-1)(q-1))$. If e is chosen to be smaller, then encryption will be quicker, as there is less multiplication involved, and the decryption will be more difficult. In more common terminology, e and n are released as someone's public key. This is the key that anyone can use to encrypt a message that only a specific person can decrypt. The only person who will be able to do this is the person with d (the private key), or a combination of p, q, e, and lots of computation time.

Immediately it can be seen that since n is a product of primes, if n could be factored someone could immediately know p and q, thus making the encryption useless. This is why it is crucial that factoring large numbers is known to be difficult. While the security of this algorithm is related to the chosen key sizes, this algorithm quickly becomes slow, as exponential computation is time-consuming.

2.6 HIPAA

With so many encryption algorithms to choose from: Blowfish, Twofish, Serpent, Crypton, etc; it is important to choose from those that have stood the test of time and extreme scrutiny. Any of these algorithms could be used, but what would its impact be on the workflow of healthcare entities? Since HIPAA is relatively new, and healthcare providers are only just realizing its significance, there has only recently been significant research in this area. There are many companies that are providing consulting for adherence to the standards, but there are not many studies done on the workflow impact or the proper choice of appropriate standards.

There are multiple products coming into the market that boast encryption for HIPAA compliance, such as Rhapsody[10] or NetSilica[11]. These specific products use a combination of encryption techniques, such as AES, DES and Blowfish. While all of these products are HIPAA compliant, none predict the impact they might have on the workflow. Encryption that takes on the order of minutes will not be accepted for it will be wasting physicians' time. The encryption method chosen is going to have a profound impact on the workflow in the hospital. But at the same time, the method

must be secure enough that it cannot be broken in any reasonable amount of time using any available amount of resources (distributed or compact).

Chapter 3 – System Performance Issues

3.1 Managed vs. Unmanaged Code

In most cases there is a large difference in performance between managed and unmanaged code. There are many differences between both types of code and many reasons why someone would choose to use one over the other. In its simplest form, the choice comes down to security – not of the algorithm, but rather of the code itself.

Unmanaged code is all native code. Any code that is compiled into native computer code requires the programmer's management of memory, security and other facilities. When unmanaged code is run on a computer, there is no translation or high level manager that is controlling any aspect of it. Managed code is code that is executed inside a container. This container controls many aspects of the program such as memory management, security and garbage collection. In .NET, the container is known as the Common Language Runtime (CLR). Managed code is compiled into an intermediate language; in .NET, this language is the Common Intermediate Runtime (CIR). In order for this code to be run on the system, it must be translated to native code, which is normally the job of the Just-in-Time compiler. Since the program is executed inside of a container, with translation into native code performed at runtime, this allows for portability. Managed code also has restrictions on it such as the inability to handle multiple inheritance or inheritance from an unmanaged type. Further, it can only perform functions that the verifier can prove are safe. Because the CLR is always managing the runtime of a program, there is often a performance hit involved due to the extra services the CLR provides.

There are many situations where unmanaged code would be the proper choice. CPU intensive processes such as games would require the fastest execution time possible to maintain high frame rates. Other real-time applications would require fast execution as well in order for the system to operate at an optimal level. In some of these cases, security is not a large concern. Most people assume software vendors will create games that are not going to destroy the hard drive. Also, there are millions of lines of unmanaged code already written and in some cases code reuse is the best policy.

Then there are the times that managed code is much preferred. Running managed code avoids memory leaks which could lead to system instability or unresponsiveness. Managed code will not contain vulnerabilities due to programming errors such as accessing an out-of-bounds member of an array. Systems that have other factors limiting the speed, such at network latency, are often better suited for managed code. Sometimes the desire for a safe environment is more important than the system responsiveness. For example, some transaction servers would be better suited for managed code to avoid system downtime due to memory errors or malicious attempts to exploit a buffer overrun, at the cost of fewer transactions per second.

There are also times when a combination of managed and unmanaged code is the best course of action. Performance critical sections can be written in native code to reap the unmanaged benefits, while other sections requiring more security and stability, such as communications, can be written using managed code. Also, the

ability to use both managed and unmanaged code in the same file allows one to port in increments rather than all at once. It is the hybrid model which benefits from the best of both worlds as long as the native code is only used where required.

3.2 Performance Studies from the Literature

Encryption can be implemented in hardware or software; hardware generally has the advantage of speed while software has the advantage of flexibility. Numerous studies have been conducted to measure the performance of various encryption algorithms and implementations and we survey that literature here. Although these previous performance measurements were not made in the .NET environment that we are using, they nevertheless provide some insight regarding the relative performance of different techniques.

In the 2001 RSA Security Conference, Gaj and Chodowiec[12] compared the performance of several AES candidates implemented using a Field Programmable Gate Array (FPGA). A FPGA is an integrated circuit that can be reconfigured quickly by a designer to perform different objectives. For benchmark purposes they included 3-DES in both feedback and non-feedback modes. Their results showed a sustained throughput of 51.7 MB/s for the feedback mode of Rijndael versus only 7.4 MB/s for 3-DES. When they used non-feedback mode, they were able to achieve rates over 1 GB/s for Rijndael. In the IEEE Transactions on VLSI in 2001, another FPGA implementation of feedback mode AES was able to achieve 37.5 MB/s[13]. Feedback mode entails that every block to be encrypted use the output from the previous block for the purpose of creating dependence. Since each block depends on the previous block, an intermediary would not be able to snoop random blocks and

decrypt them since it would be missing vital information. While it would be possible to stagger the dependence such that block X depends on block (X-Y), this would require a staggered feedback mode standardization that is not yet in place. In the 5th Annual Workshop on Selected Areas in Cryptography (SAC '98), FPGA implementations of DES achieved rates up to 50 MB/s[14].

To test software efficiency, Dr. Brian Gladman[15] implemented the AES candidates and DES in C and C++. He reported a sustained throughput of 128-bit Rijndael at 8.6 MB/s while the 256-bit version operated at 6.3 MB/s. In comparison, DES was markedly slower at 3.8 MB/s. In the 3rd AES Candidate Conference in 2000, Aoki and Lipmaa[16] were able to write assembly level code to achieve the fastest known software throughput of 30.4 MB/s for 128-bit Rijndael. All of the previously mentioned throughputs were attained with a 32-bit architecture. Using a 64-bit architecture, faster throughputs are possible. Corella[17] was able to attain 22.8 MB/s for the DES algorithm and 9.4 MB/s for 3-DES in a 64-bit architecture.

In order to test the algorithms in a code-safe environment, Sterbenz and Lipp[18] implemented all the AES candidate algorithms in Java to study their relative performance in the AES Candidate Conference 2000. The 128-bit version of Rijndael was the fastest with a sustained throughput of 2.4 MB/s; the 256-bit version operated at 1.9 MB/s. For comparison, DES was measured at 1.3 MB/s and 3-DES operated at 0.5 MB/s. NIST also performed their own Java performance evaluation in 2000[19]. They were able to attain a 128-bit Rijndael throughput of 0.6 MB/s and a 256-bit throughput of 0.5 MB/s. These results are just a handful from a larger pool

that shows the performance of Java to be slower than native code. Since these measurements are not isolated, one can propose they were not bad implementations. This means the performance hit for Java comes more from the Virtual Machine and the Just-In-Time compiler. The overhead for code safety is apparent in the 10:1 ratio comparing native code to Java. From these numbers, it can be inferred that operating in a Java environment is not conducive to achieving high throughputs.

The RSA public key cryptography algorithm has the advantage of variable key size, but is substantially slower than the fixed-key-length symmetric key algorithms. Shand and Vuillemin[20] reported using a hardware implementation of RSA with 1024-bit keys to achieve a sustained decryption throughput of 0.02 MB/s; Johann Groszschaedl[21] was able to increase RSA's decryption performance in hardware to 0.25 MB/s by using the Chinese Remainder Theorem (CRT). RSA's software performance is even slower. Neil Wagner[22] reported implementing RSA and CRT with 1024-bit keys in Java with a resulting throughput of 4 KB/s (about 60 times slower than the hardware implementation of the same algorithm). RSA Laboratories reports[23] that, in general, RSA is about 100 times slower than DES in software and up to 10,000 times slower than DES in hardware.

3.3 Rationale for New Measurements

The UVA hospital is interested in using Microsoft .NET technologies in order to become compliant with HIPAA and there is only one performance measurement analysis[24] for the .NET Cryptographic classes; it solely deals with server responsiveness and user load. Hardware solutions are not currently being considered due to the inflexibility that accompanies them. The UVA hospital desires a flexible

architecture that allows for easy use of different algorithms should the need arise. The hospital also wishes to maintain an uninterrupted workflow while maintaining security. This leads to a tradeoff between having the security of managed code or the performance of unmanaged code. Ideally, the algorithm that is chosen will satisfy the performance requirements and be implemented in a managed environment.

The .NET Framework has a suite of Cryptographic classes that include DES, 3-DES, AES and RSA. DES, 3-DES and RSA are implemented using wrappers on the unmanaged CryptoAPI, while AES is implemented in a completely managed environment. The CryptoAPI is a suite of classes initially created in 1993 by Microsoft and has been continually updated with faster and more robust versions of the algorithms. The fact that some of the algorithms are implemented with wrappers on unmanaged code and others are implemented with purely managed code will lead to slightly skewed measurements when comparing performance. Unmanaged code implementations of DES, 3-DES and RSA will perform better than their corresponding managed code implementations. However, the unmanaged components would not be portable due to the underlying unmanaged code.

After researching and evaluating the performance characteristics of the chosen algorithms in a .NET environment, I chose an algorithm that I believe best fits the current and future needs of the hospital. By comparing the workflow impact of this algorithm against the fastest known implementation of the algorithm, one will be able to determine the price of managed code versus unmanaged code using current compilers. A rough estimate of the performance gain noted when using optimized

assembly vs. Java for 128-bit Rijndael leads to a speedup of over 12. By testing in the .NET architecture, we will be able to ascertain if the difference between managed and unmanaged code is still so large.

Chapter 4 – Performance Measurements of Software Encryption

4.1 Known Attacks

All of the algorithms that are used in this thesis have their own respective weaknesses. These are weaknesses that could be exploited by an attacker to gain access to the decrypted versions of files. It is important to understand the security involved in each of the algorithms and just how much that security can be trusted given the known attacks. The following is a quick overview of the most successful attacks on each of the encryption algorithms that are examined in this thesis.

DES has long been the standard by which all other cryptographic algorithms were measured. It has stood the test of time and decades of research and analysis. DES depends on a single 56 bit key. Many attacks against DES have been publicized throughout the years, with the most obvious being a brute-force attack (trying all 2^{56} keys). Initially infeasible in the 1970s and 1980s, this attack is now practical. Computers became fast enough that a distributed attack using a specially designed supercomputer and distributed computers could break DES in under a day in 1999[4]. The brute-force attack is the most practical way of determining the 56-bit key since it requires only a small amount of ciphertext. Other attacks can be used against DES, such as differential cryptanalysis and linear cryptanalysis. Both of these attacks are characterized as *known plaintext* attacks. The attacker chooses known blocks of plaintext and generates the corresponding ciphertext. Then the data analysis phase begins where computers analyze the differences in the ciphertext blocks and then generate parts of the key used for the encryption. In some cases the entire key can be

discovered, while in others the reduction of the effective key size then enables a brute-force search.

3DES has at least twice the key size of DES, and is backwards compatible with DES hardware. For these reasons, 3DES has been used in the time period between DES and AES. With a key length of 128 or 192 bits (112 or 168 effective bits), brute force becomes once again infeasible for 3DES. Even if there existed a way to break DES in one second, the extra 56 or 112 bits would provide an additional 2.2 billion or $1.6x10^{26}$ billion years, respectively, of computational protection. There are other attacks on 3DES, such as "Meet in the Middle"[25] (MITM) attacks. They involve taking a known plaintext, and for each key k_1, storing the encrypted text. Then, given a known ciphertext, one can begin decrypting it by trying all possible k_2, and match the resulting interim ciphertext against the database created from the multitude of k_1s. For the 112-bit version, this reduces the brute force attack from 2^{112} encryptions to 2^{56} encryptions and 2^{56} table lookups[26]. This will yield the two keys that were used for the encryption. In the case of three keys this attack requires 524,288 terabytes of storage, 2^{112} encryptions and 2^{112} table lookups. This is currently infeasible due to memory limitations and computational limitations. An optimized version of the MITM attack requires 2^{108} operations. These characteristics make breaking 3DES infeasible in any reasonable amount of time. But since 3DES is just DES run three times, in an un-optimized environment it operates on the order of three times slower than DES, which is why new algorithms were sought and the AES adopted.

With AES, keys can be 128, 192 or 256 bits. The increased key size makes brute force virtually impossible in a reasonable amount of time (at least today). With a

computer that could crack DES in one second, it would still take 150 trillion years to crack the 128 bit key of AES. While ultimately brute-force would break AES, it is not viewed as a practical attack given the time complexity. Only recently have there been any advancements in attacks on AES. These attacks are categorized as algebraic attacks. However, these attacks are purely theoretical and cannot yet be implemented. Even if they were to be implemented, they would take on the order of 2^{100} operations, which is still outside the realm of feasibility. They involve re-writing AES as multivariate quadratic polynomials and solving the resulting equations[27]. While this attack has not yet been verified by outside sources, it is the only known theoretical attack that would perform better than brute force. Other attacks on AES require infeasible amounts of chosen plaintext.

The security of RSA is based upon one important mathematical principal: the difficulty in factoring large numbers. If it was effortless to factor large numbers, some malicious person could factor *n* into *p* and *q* and thus break the cipher. The fastest method known for factoring large numbers is the General Number Field Sieve, created by John Pollard in 1988. This method takes on the order of $\exp[(1.932+O(1)) * (\log m)^{1/3} * (\log(\log(m)))^{2/3}]$ for an m-bit number. Brute force would involve trying every multiplicative pair up to $\sqrt{n}$. Because of this, the complexity would be $(\sqrt{n})^2 = n$. This means that one would have to try about 2^n operations before finding the correct factors. Since *n* can easily be on the order of 1024 bits, the brute force approach could take over 2^{1024} operations, which is outside the realm of feasibility given today's state of technology. Many other attacks against RSA are based upon poor implementation. A company might use the same *n* for

every employee's key, but choose different values of *e* and *d* for each employee. While no employee has access to any other employee's private key, he or she does have access to the public key of other employees. Using his own *e* and *d*, an employee can factor *n* and then use another employee's public key to compute the corresponding private key. Such attacks are not against RSA itself but upon poor implementations of the algorithm and so they will be noted but disregarded. In 1999, the General Number Field Sieve was used over a period of 7.4 months to break an RSA-155 key[28]. This was a key with 155 digits which corresponded to a 512-bit number. Since this attempt used approximately 292 machines, it can be seen that a much larger distributed attack would dramatically reduce this time. However, this does not mean that RSA can be easily broken, just that shorter keys can be broken. If 512-bit RSA could be broken in one second, it would still take 4.2×10^{146} years to break 1024-bit RSA.

4.2 Performance Testbed

To test the performance of each of the encryption algorithms, I created an encryption testbed. Each of the tests was run on the same computer under nearly identical load. I used a 3 GHz Pentium 4 computer with 1 GB of RAM running Windows XP. I developed the encryption application using Microsoft Visual Studio .NET 2003. Each of the encryption methods had a corresponding class in the .NET Software Development Kit (SDK). The classes were created with the fastest known implementations of the algorithms. This leads to measurements that roughly estimate the best software encryption speeds possible given early 2003 hardware and software.

My application has many options that help in testing both the encryption and decryption speeds. It is possible to select any number of files and encrypt them any number of times, with all the measurements being added to encryption-specific logs. For each different file size, using each algorithm and all approved key sizes for DES, 3DES and AES, I performed 100 encryptions and decryptions. With DES, this is just encrypting and decrypting using 64-bit keys. 3DES is approved at 128 and 192 bits while AES is approved at 128, 192 and 256 bit keys. For RSA I chose key sizes of 512 and 1024 bits, as 512 was recently broken and 1024 is viewed as adequately secure[29].

I used four different tests to measure standard loads that a hospital might see on an hourly basis. First, I chose a one-byte file, because this is the smallest file that the algorithms can encrypt; thus these results will be a lower bound for any algorithm's encryption and decryption time. The second test is a one-megabyte file, which represents a single compressed medical image file at a 2000x1500x16 screen resolution. The third test is a three-megabyte uncompressed image file (4000x3000x16 screen resolution) because that is common for a high-resolution image. The fourth test is a 500 image MRI set, each image reflecting a 256x256x16 screen resolution. This results in a 68 MB set of files. Each of the encryptions and decryptions generate an entry in a log file specific to the encryption and key size. The entry consists of the total file size and the encryption/decryption time. These data logs were used to calculate the throughput of each of the algorithms.

4.3 Performance Measurements

My first observation was the disparity between RSA public key encryption and the other three symmetric key schemes. Figure 11 shows how RSA compares against DES, 3DES and AES. It can be seen that 1024-bit RSA decrypts at speeds more than 100 times slower than the other encryptions. 512-bit RSA decrypts at speeds more than 40 times slower than the other methods discussed in this thesis. Observed delays of 40-120 seconds for a four-megabyte file are unacceptable, and thus RSA will be categorized as a low performance algorithm.

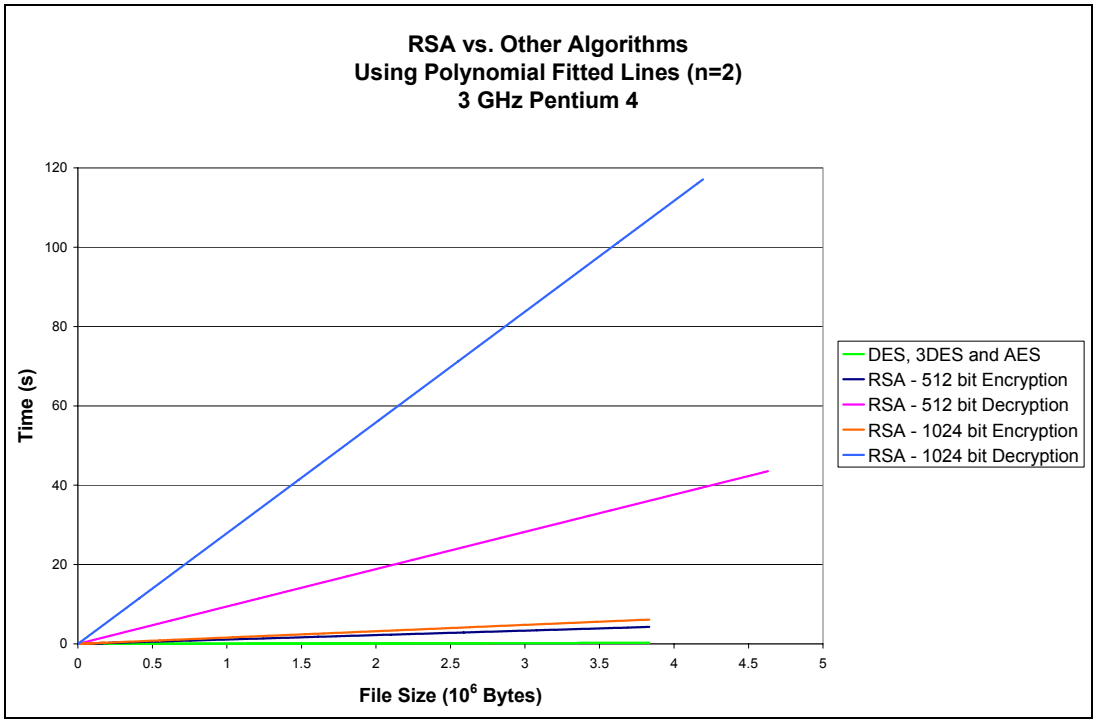

Figure 11

When looking at DES, 3DES and AES (now referred to as the high performance algorithms), the encoding and decoding characteristics are quite similar. Figure 12 shows how all these algorithms compare to each other for encryption and decryption. It can be seen that for all files DES performs fastest, and AES-256 performs slowest.

The algorithm that attains the highest overall throughput is the DES, which averages 8.10 MB/s.

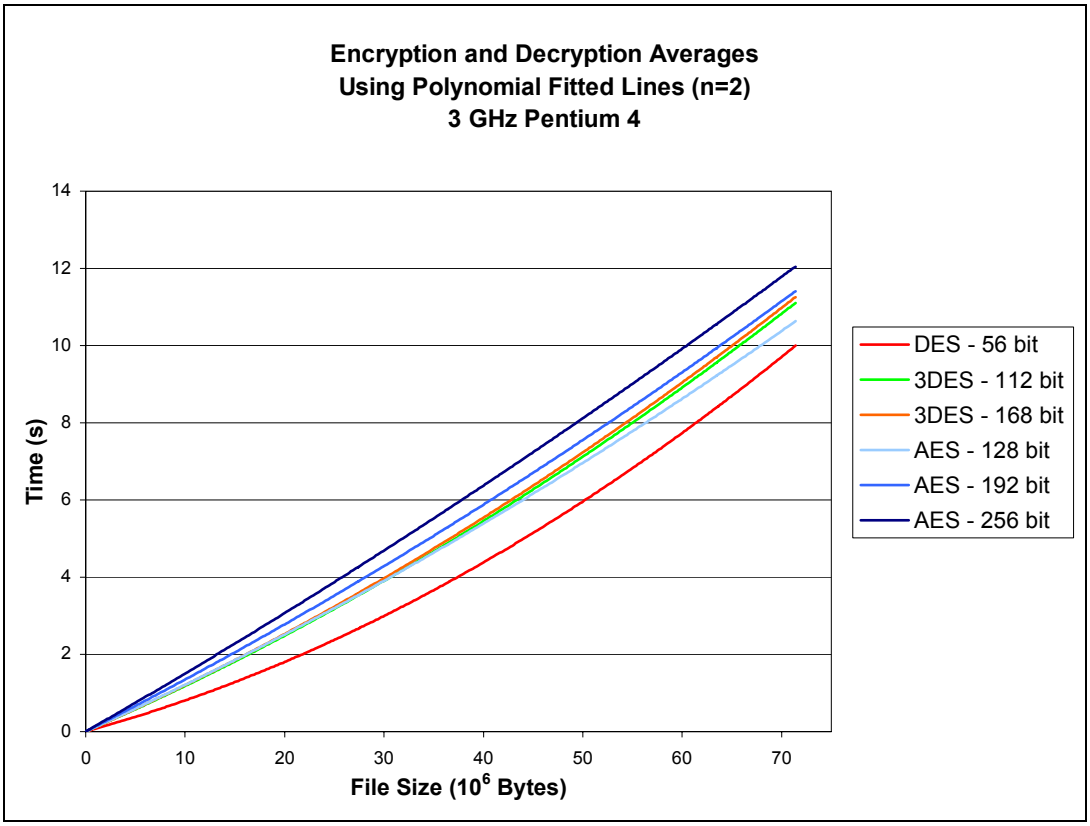

Figure 12 – Graph of Algorithm Performances

Table 1 shows the throughput of all the algorithms, sorted from highest to lowest. Each throughput measurement was averaged over the 400 tests that were preformed on each algorithm. It can be seen that DES out-performed the other algorithms in both encryption and decryption. The throughput drops dramatically when considering RSA, which is why I have categorized it under low performance.

Encryption	MB/s	Percent of Fastest Algorithm
DES 56-bit	8.51	100.00%
3DES 112-bit	7.23	84.90%
AES 128-bit	7.19	84.50%
3DES 168-bit	7.16	84.12%
AES 192-bit	6.63	77.93%
AES 256-bit	6.24	73.36%
RSA 512-bit	0.90	10.53%
RSA 1024-bit	0.62	7.34%

Decryption	MB/s	Percent of Fastest Algorithm
DES 56-bit	7.68	100.00%
AES 128-bit	6.96	90.61%
3DES 112-bit	6.56	85.42%
3DES 168-bit	6.45	83.88%
AES 192-bit	6.41	83.42%
AES 256-bit	5.95	77.40%
RSA 512-bit	0.11	1.38%
RSA 1024-bit	0.04	0.47%

Table 1 - Throughput of Encryption and Decryption on 3 GHz Pentium 4

4.4 Analysis of Results

All of the cryptographic algorithms that were used have their strengths and weaknesses. After performing all of the experiments, it became clear that performance was going to play a large role in which algorithm I recommended for different scenarios. There were also a few surprises in the results that show the limits imposed by current hardware.

RSA showed once again that it is not a suitable encryption for large amounts of data. While the security was adequate for large enough key sizes, the performance is undeniably poor. Good uses of RSA are for very small files or small amounts of data, such as the encryption of hashes or other (symmetric) encryption keys. The discrepancies in encryption and decryption times for similar key sizes come from the different choices of *e* and *d*. Most people choose a smaller *e* so that encryption takes less time and the decryption is more difficult, which is what I tried to mirror.

The 128-bit version of 3DES performed almost identically to the 192-bit version, which is to be expected, given they are the identical algorithm, and effectively the

same key size. Although the 128-bit version only technically has two keys, one of the keys acts as a third for the final step of the algorithm. So while in some of the tests the 128-bit version did slightly better, in others the 192-bit version performed better, but neither was ever dominant by more than 1%.

As expected, DES performed better than 3DES but not by as much as I expected. I had expected 3DES to be roughly half as fast as DES, given the knowledge of the algorithm. Upon closer examination, it became apparent that the throughput on the 68 MB set of files was approximately 70% as fast as the throughput on the other files. Other than size, the only difference was in the number of files being encrypted and decrypted. I re-ran each of the tests on a single 68 MB file (the conglomerate of the MRI set). The throughput of these tests was along the lines of the other single file encryptions and decryptions. This shows there is a great deal of overhead when dealing with multiple files.

Since there was a large overhead for the encryption and decryption of multiple files, I wondered how much of an impact the overhead actually had on the results. This led me to perform the tests on a slower computer, to examine the computational difficulty of the algorithms, and not just the overhead. I used a 600 MHz Pentium 3 machine with 512 MB of RAM running Windows XP. After performing the identical tests, the throughputs fell into almost the same ordering, as can be seen in the following table.

Encryption	MB/s	Percent of Fastest Algorithm
DES 56-bit	2.76	100.00%
3DES 168-bit	1.81	65.38%
3DES 112-bit	1.80	65.30%
AES 128-bit	1.79	64.85%
AES 192-bit	1.58	57.06%
AES 256-bit	1.41	51.06%
RSA 512-bit	0.28	10.27%
RSA 1024-bit	0.21	7.76%

Decryption	MB/s	Percent of Fastest Algorithm
DES 56-bit	2.14	100.00%
AES 128-bit	1.64	76.40%
3DES 168-bit	1.54	71.86%
3DES 112-bit	1.53	71.57%
AES 192-bit	1.47	68.50%
AES 256-bit	1.37	64.06%
RSA 512-bit	0.03	1.39%
RSA 1024-bit	0.01	0.54%

Table 2 - Throughput of Encryption and Decryption on 600 MHz Pentium 3

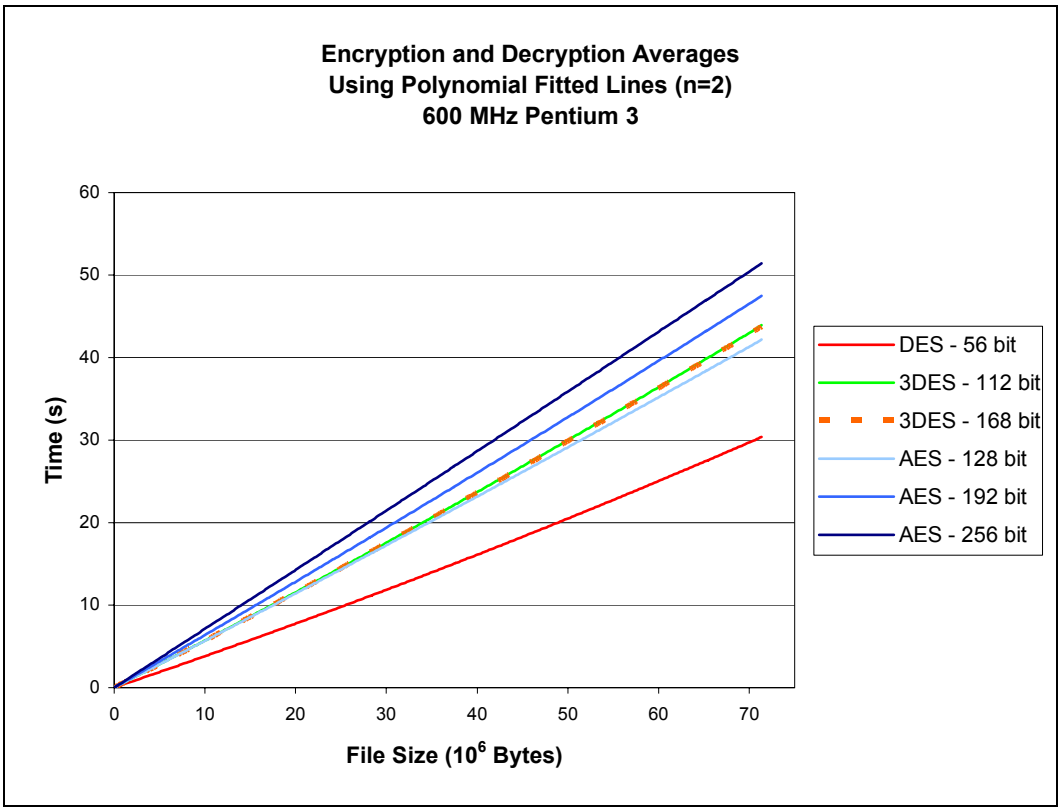

Figure 13 – Graph of Algorithm Performances

The important thing to note is that there is a larger difference in performance on the 600 MHz machine than on the 3 GHz machine. Table 2 includes all the tests done

on the 500 image MRI where all 500 images are separate files. Once those tests were replaced by tests with a single 68 MB file, the throughputs mirrored what I had initially expected. On a computer where the computation is the intensive driving force, DES was almost twice as fast as 3DES.

AES performed extremely well in all scenarios. The 128-bit version was even faster than 3DES overall. This makes sense given that AES was chosen by NIST because of its performance. Also, AES uses 128-bit blocks, and so it has to hit memory fewer times to get the required information (3DES uses 64-bit blocks). This means that on any computer where memory access is the limiting factor, AES will perform better than 3DES. The 128-bit AES performed better than the 192-bit and 256-bit versions in every test, which is to be expected as it has the fewest rounds and should therefore be fastest.

4.5 Recommendations

After reviewing the results, it appears that DES, 3DES and AES will perform similarly given the current top-of-the-line consumer hardware and software. RSA is best suited for smaller amounts of data, such as the possible encryption of AES keys on a per-employee basis, or encryption of data hashes to ensure integrity. My recommendation is to use the 256-bit version of AES. While the 128-bit or 192-bit versions are suitable and faster, the performance gap is shrinking. On the 600 MHz machine, 256-bit AES performed at 81% the speed of 128-bit AES. Using the 3 GHz machine, 256-bit AES performed at 86% the speed of the 128-bit version. And on both machines, when encrypting a large number of files, 256-bit AES performed at between 91%-93% the speed of 128-bit AES. Given the continuous improvement in

computational speed, this performance gap will continue to narrow over time. I thus recommend using 256-bit AES encryption for two reasons: (1) It is implemented using purely managed code. This leads to portability and security that would otherwise be difficult to achieve. (2) The data will be more secure. AES 256-bit keys are 2^{128} times as secure as 128-bit keys with regard to brute force search. Assuming that there is not a backdoor found to AES, eventually the 128-bit version will be susceptible to brute force. Also, quantum computers have now reached the size of 7 qubits[30]. This quantum computer created by IBM had the ability to factor the number 15. The rate of computational growth for quantum computers is unknown as thus far the number of bits has doubled every two years. It could potentially follow Moore's Law but it could also follow some kind of linear or exponential law. Both of these potential problems warrant the use of a slightly slower, but vastly more secure key size.

Chapter 5 – Workflow Modeling

5.1 Radiology Department Workflow Model

Having recommended 256-bit AES as the encryption algorithm that should be used in the healthcare industry, it is important to understand what impact, if any, this decision will have in a healthcare environment. The area where this will have the most profound effect is the storage and transmission of large amounts of data, especially medical images. The Department of Radiology at the University of Virginia conducts over 380,000 examinations and generates approximately 9 TB of data a year. This is why the Radiology department is an excellent candidate upon which to base the workflow model. Since encryption has to be completed at both the server and client, it is important to note that since the server will undoubtedly be under a high user load, it is safe to assume that the amount of encryption done at the server will outweigh any performance problem due to a clients' computer. The Department of Radiology at the University of Virginia has a workflow model that they use to predict changes in the workflow due to any number of changes to the system.

In order to model the workflow in the Radiology department, it is important to understand how the department operates and how data flows. Figure 14 is a dataflow diagram of the University of Virginia's Department of Radiology's PACS (Picture Archive and Communication System) model that integrates with the other vital components of the department, such as the Hospital Information System (HIS), Radiology Information System (RIS), Digital Imaging and Communications in Medicine (DICOM) gateways, various image modalities, reporting systems, Health

Level 7 (HL7) communications and SQL databases. This model[31] is used by the University of Virginia to conduct throughput assessments and locate bottlenecks in their clinical procedures.

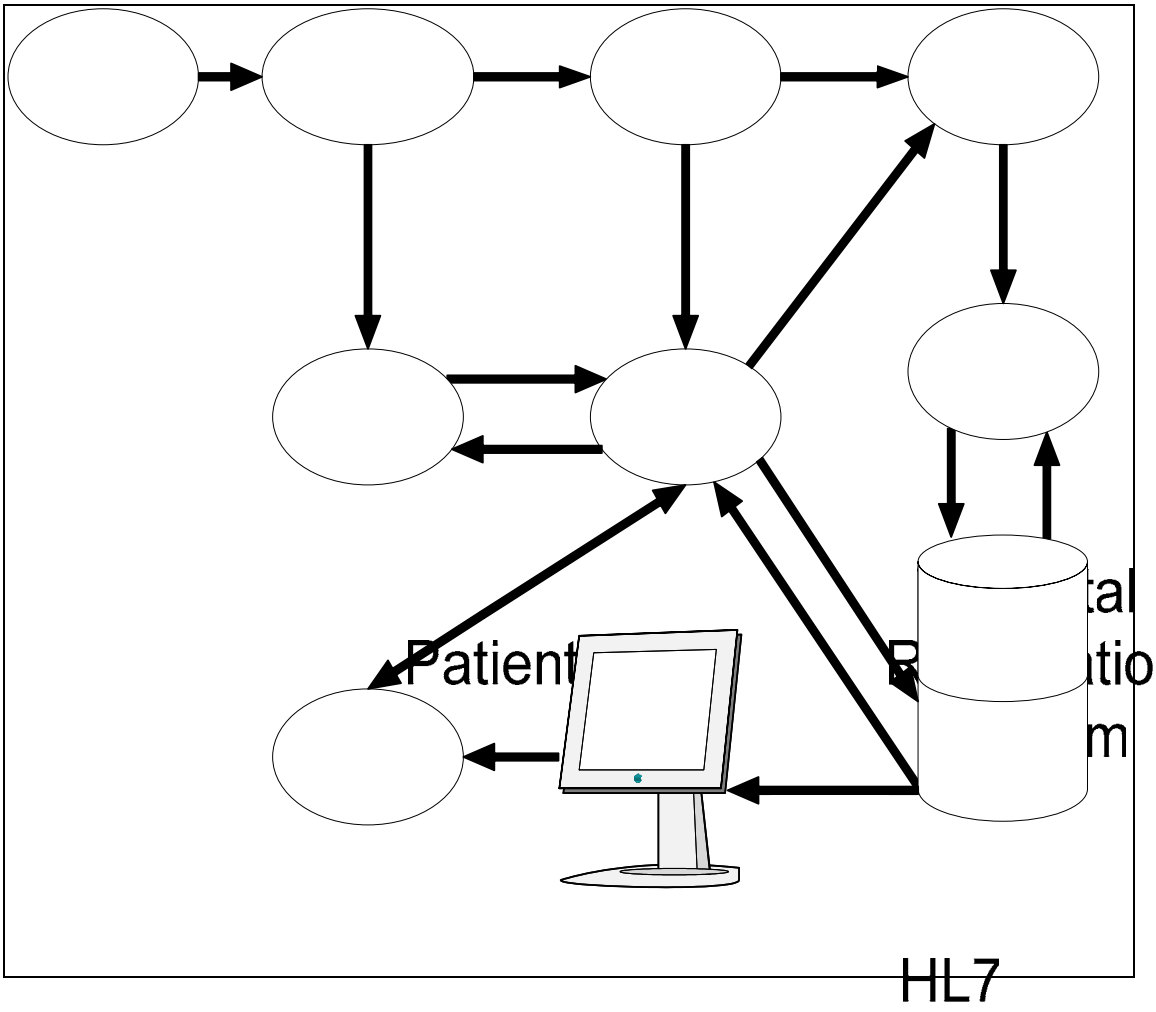

Figure 14 – Dataflow in the Radiology Department

5.2 Workflow Architecture

In the proposed system, there are many steps that need to be accomplished from the moment a patient walks in the door until the department sends its report to the

HIS. Each of these steps has associated resources that are used. When multiple steps use the same resource, it degrades the possible throughput, and creates potential bottlenecks. This makes it important to create a set of steps that mimics the actual system and uses the available resources in order to find bottlenecks and the impact of the encryption. This method of workflow analysis stems from the work of BW Stuck and E. Arthurs[32]. Table 3 contains 13 steps that are based upon typical groups of steps in the radiology department[33].

Steps	Description
A	Patient Registration by hospital registration system
B	Notify HIS of patient and data using HL7
C	Schedule exam and notify RIS
D	Patient data to RIS and to PACS archive
E	DICOM worklist to image modality
F	Conduct patient exam
G	Patient image data to gateway using DICOM
H	Relational data to gateway (required prior images)
I	DICOM image data from gateway to PACS archive
J	DICOM image data to workstation from PACS archive
K	Patient report generated in reporting system
L	Patient report send to RIS from reporting system
M	Patient report sent from RIS to HIS

Table 3 – Typical Steps in the Radiology Department

In order to find the bottlenecks in the system, it is important to note the resources associated with the system. Each of the resources is a potential bottleneck depending on which steps use them. If every step has to use a database, and there is only one connection allowed, then each step has to run separately, and there can be no pipelining. The following table contains the list of resources that would be used in this system.

Resource	Description
R_1	Hospital Registration System
R_2	HIS (Hospital Information System)
R_3	RIS (Radiology Information System)
R_4	Examination Schedule System
R_5	HL7 Communications for Text Data
R_6	DICOM Communications for Image Data
R_7	Image Modality Unit
R_8	DICOM Gateway
R_9	Relational Database
R_{10}	PACS Archive
R_{11}	Workstation
R_{12}	Reporting System
R_{13}	Encryption/Decryption Application

Table 4 – Resources in the System

The list of resources is used in combination with the steps in the system to form a matrix of usage. The matrix shows the resources that each step uses and is used to calculate the throughput of each of the resources. The resource allocation table is listed in table 5.

STEP	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	TIME
A	1	0	0	0	0	0	0	0	0	0	0	0	0	T_1
B	1	1	0	0	1	0	0	0	0	0	0	0	0	T_2
C	0	0	1	1	1	0	0	0	0	0	0	0	0	T_3
D	0	1	1	0	1	0	0	0	0	1	0	0	1	T_4
E	0	0	1	0	0	1	1	0	0	0	0	0	0	T_5
F	0	0	0	0	0	0	1	0	0	0	0	0	0	T_6
G	0	0	0	0	0	1	1	1	0	0	0	0	1	T_7
H	0	0	0	0	0	1	0	1	1	0	0	0	1	T_8
I	0	0	0	0	0	1	0	1	0	1	0	0	1	T_9
J	0	0	0	0	0	1	0	0	0	1	1	0	1	T_{10}
K	0	0	0	0	0	0	0	0	0	0	0	1	0	T_{11}
L	0	0	1	0	1	0	0	0	0	0	0	1	0	T_{12}
M	0	1	1	0	1	0	0	0	0	0	0	0	0	T_{13}
	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	

Table 5 – Resource Allocation Table

In the resource allocation table, each step either uses resource R_x, noted as a one, or does not use resource R_x, noted as a zero. At the bottom of the table is the name of the bottleneck (B_x) associated with each resource. At the right side of the table is the time required to complete a given step. T_{10} is the time required to complete step J, which uses R_6, R_{10}, R_{11} and R_{13}. The total of all the times ($\sum T_i$) yields the average time to complete one job. In order to determine the bottleneck(s) in the system, one computes the throughput of each of the possible bottlenecks. The system can only operate as fast as its slowest bottleneck. In order to compute the bottlenecks, one solves $B_x = 1 / (T_i + T_j + \ldots + T_z)$ where resources i, j, …, z are used. The times that are used are the times associated with the steps that use resource R_x. For example, $B_{13} = 1 / (T_4 + T_8 + T_9 + T_{10})$. The unit of the result B_x is jobs per second. The resource bottleneck equations are listed below in table 6.

Bottleneck	Equation
B_1	$1 / (T_1 + T_2)$
B_2	$1 / (T_2 + T_4 + T_{13})$
B_3	$1 / (T_3 + T_4 + T_5 + T_{12} + T_{13})$
B_4	$1 / (T_3)$
B_5	$1 / (T_2 + T_3 + T_4 + T_{12} + T_{13})$
B_6	$1 / (T_5 + T_7 + T_8 + T_9 + T_{10})$
B_7	$1 / (T_5 + T_6 + T_7)$
B_8	$1 / (T_7 + T_8 + T_9)$
B_9	$1 / (T_8)$
B_{10}	$1 / (T_4 + T_9 + T_{10})$
B_{11}	$1 / (T_{10})$
B_{12}	$1 / (T_{11} + T_{12})$
B_{13}	$1 / (T_4 + T_7 + T_8 + T_9 + T_{10})$

Table 6 – System Resource Bottlenecks

In order to solve each of the above equations, the average time to complete each step is required. These times were collected using a mini-PACS system at the

University of Virginia Hospital without using encryption, and are listed below in Table 7. The average times with encryption are estimated based upon the MRI examination and the encryption/decryption rate of 6.81 MB/s measured in chapter three for 256-bit AES.

Time	Average Time without Encryption	Average Time with Encryption	Short Description
T_1	900 seconds	900 seconds	Patient registration
T_2	5 seconds	5 seconds	Notify HIS of patient
T_3	30 seconds	30 seconds	Schedule exam
T_4	10 seconds	11 seconds	Patient data to RIS and PACS
T_5	10 seconds	10 seconds	Worklist to image modality
T_6	1200 seconds	1200 seconds	Conduct patient exam
T_7	180 seconds	240 seconds	Patient image data to gateway
T_8	180 seconds	240 seconds	Relational DB images to gateway
T_9	180 seconds	240 seconds	Image data from gateway to PACS
T_{10}	120 seconds	180 seconds	Image data to workstation
T_{11}	120 seconds	120 seconds	Patient report generation
T_{12}	30 seconds	30 seconds	Patient report to RIS
T_{13}	30 seconds	30 seconds	Patient report from RIS to HIS

Table 7 – Average Times for Each Step in the System

Using the times from Table 7 and the equations from Table 6, the following values were calculated as possible bottlenecks.

Bottleneck	Equation	Without Encryption	With Encryption
B_1	$1 / (T_1 + T_2)$	3.98	3.98
B_2	$1 / (T_2 + T_4 + T_{13})$	79.92	78.26
B_3	$1 / (T_3 + T_4 + T_5 + T_{12} + T_{13})$	32.73	32.43
B_4	$1 / (T_3)$	120.00	120.00
B_5	$1 / (T_2 + T_3 + T_4 + T_{12} + T_{13})$	34.29	33.96
B_6	$1 / (T_5 + T_7 + T_8 + T_9 + T_{10})$	5.37	3.96
B_7	$1 / (T_5 + T_6 + T_7)$	2.59	2.48
B_8	$1 / (T_7 + T_8 + T_9)$	6.67	5.00
B_9	$1 / (T_8)$	20.00	15.00
B_{10}	$1 / (T_4 + T_9 + T_{10})$	11.61	8.35
B_{11}	$1 / (T_{10})$	30.00	20.00
B_{12}	$1 / (T_{11} + T_{12})$	24.00	24.00
B_{13}	$1 / (T_4 + T_7 + T_8 + T_9 + T_{10})$	N/A	3.95

Table 8 – Bottleneck Results in Patients/Hour

5.3 Workflow Results

From these values, it can be seen that in the Department of Radiology that encryption is not the bottleneck when considering all the system resources, nor does it change the actual bottleneck resource. However, the encryption does lower the throughput of the bottleneck. If we were dealing with a single doctor, and he was not using encryption, the resource bottleneck B_7 (from Table 8) would allow up to 62 patients per 24 hours. If the same doctor was using encryption, they would be able to see up to 59 patients per 24 hours. So the encryption has an effect of degrading the performance 5% in a system that is optimally concurrent. In the worst case scenario in which each patient must be seen and all pertinent documents completed before the next patient can be registered, the system performance will be lowered by 7%.

A 7% decrease in throughput could pose a problem if the operating margin is not very high. If a throughput decrease of no more than 2% was desired, it would require an algorithm with a throughput of at least 17 MB/s. While this throughput was not achieved using 256-bit AES in .NET, previous published results have achieved throughputs above 17 MB/s using native code. Unfortunately, this would mean giving up the security and portability of managed code.

In order to understand the effect of encryption on patient throughput, especially when considering concurrent patients, it is important to establish bounds. In order to graph these bounds, it is important to note that the T_e is the time required to do the encryption when considering the life-cycle of the patient. T_s is the time spent completing the rest of the required steps for a patient. If we can assume that our

system contains one pipeline and each step is independent of all other steps then we can establish the following bounds on the mean throughput rate.

(1) In the best possible scenario, each patient is independent of each other patient, and does not have to wait for any step, which leads to an absolute upper bound of $N/(T_e + T_s)$. As N increases, this bound becomes unattainable, as there are not enough resources.

(2) The maximum attainable throughput is achieved by considering solely the bottleneck and is not affected by the number of patients waiting in the queue. This bound is equal to $1/ T_b$, and in our case T_b is equal to T_6, as the patient exam clearly takes the longest to complete.

(3) The actual upper bound throughput is dependent on the number of patients in the queue, and the availability of the bottleneck step. This throughput is equal to $N/(T_e + T_s + (N-1)*T_6)$. This models the effect that each patient has to wait some amount of time for all the previous jobs to complete. This bound is always less than or equal to the minimum between the absolute and attainable bound. As N approaches infinity, this bound will approach $1/T_b$.

(4) The lower bound assumes that there is no pipeline but rather a sequential execution of the steps and each patient must wait for the previous patient to complete all steps before entering into the system. This lower bound is simply $1/(T_e +T_s)$, and is unaffected by the number of patients waiting in the queue.

If one uses the preceding bounds to graph the mean throughput, it can be seen that the encryption does not affects the attainable throughput ($1/ T_b$) since the bottleneck

is unaffected by the encryption. However, it does affect the absolute upper bound, the actual upper bound convergence speed, and the lower bound.

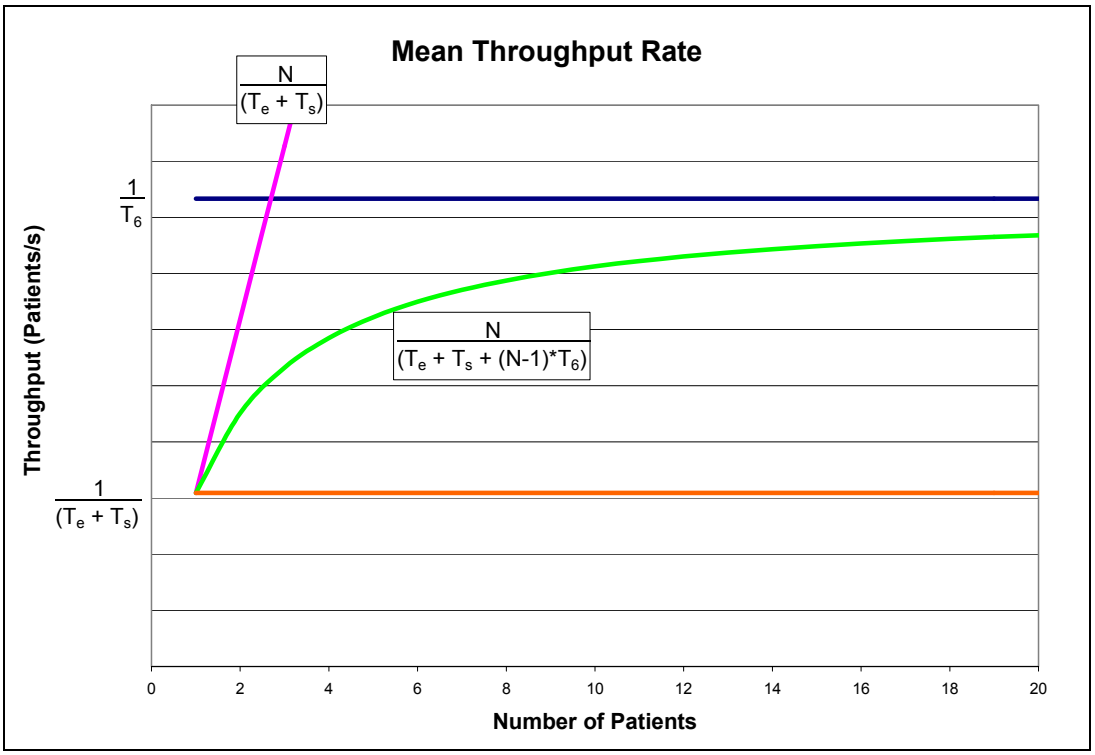

Figure 15 – Throughput of System with Encryption

Since T_6 is identical for both the system with and without encryption, one can plot the actual upper bounds and lower bounds for both systems in order to see the performance difference given N jobs in the system. Note that the absolute upper bound will be different as well.

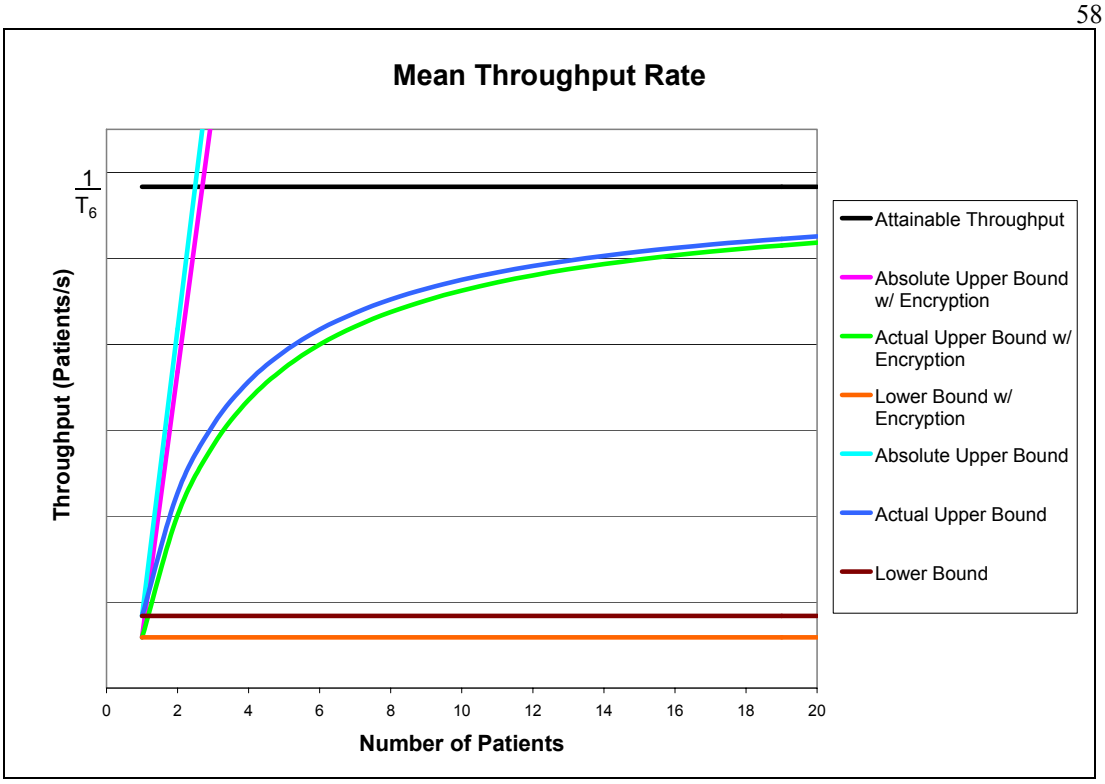

Figure 16 – Comparison between Throughput with and without Encryption

The actual operating mean throughput rate is somewhere between the upper and lower bounds. This is why it is important to have a solid system implementation. The graphs show that increasing the concurrency of patient encounters reduces the performance impact of using or not using encryption. However, if the system is fairly sequential, the performance hit for encryption will be steady. These results suggest that encryption will not adversely affect the healthcare industry if reliable pipelined implementations are in place.

This is a welcome result for the healthcare industry. Prior to this study, the impact of HIPAA was feared but unknown. The Department of Radiology was very concerned over the possible impact of encryption in their network. Based upon our modeling of the University of Virginia Radiology Department, we can now predict a

5-7% decrease in patient throughput. While no decrease is desirable, these results suggest that no panic is warranted. The projected decrease in patient throughput attributable to encryption is small, and can be overcome by increased efficiency in other steps.

Chapter 6 – Conclusions

6.1 Recommendations

HIPAA brings about a new age of security and privacy for healthcare information. With these new standards, healthcare entities will have to adhere to practices for which many are unprepared. Some of the published standards are already in effect for major healthcare entities, with the smaller entities having under a year to become compliant. In fact, the Privacy rules have already gone into effect as of April 14, 2003. The security standards which go into effect in April 21, 2005 have points in them which require implementations for encryption, authentication, auditing, authorization and more.

Healthcare entities are not yet prepared to enact these new standards. In fact, healthcare entities are not even sure how these new standards will affect their processes and workflow, which is a disturbing item for them. This thesis set out to determine what effect encryption would have on the workflow in a healthcare environment. With so many encryption technologies to choose from, such as DES, 3-DES, AES, RSA, and more, it was important to see the advantages and disadvantages of each of these algorithms and what they offered in terms of performance and security.

In order to test the performance of each of the encryption methods, I created a testbed that included software implementations of DES, 3-DES, AES and RSA created in a Microsoft Visual Studio .NET environment. With the new HIPAA standards that are coming into effect, one of the possible compliance solutions will be

to use a conglomerate of Microsoft products plus some home-grown code. The .NET platform works well in this scenario because of all the distributed systems and the ease of communication between them utilizing .NET. By implementing the encryption in a software application or service, a healthcare entity could easily change encryptions by just switching the encryption module. There would not be a need for any hardware changes.

In the testbed, I encrypted files of differing sizes, each one corresponding to a typical medical image. With each of the encryption methods and sizes, I encrypted and decrypted each file 100 times. This lead to more than 16,000 data points for the 3 GHz and 600 MHz machines. Using these data points, I graphed the encryption and decryption speeds of each of the algorithms and quickly discovered how rapidly RSA's performance deteriorates as file size increases. DES proved to be the fastest algorithm, with 128-bit AES right behind. 3-DES proved to be almost half the speed of DES when disregarding overhead. In the encryption and decryption process, there is a large overhead associated with multiple files. When the MRI test was performed with a single 68 MB file as opposed to 500 separate images, the throughputs of each of the algorithms grew by as much as 50%.

After analyzing these encryption methods and their performance, I recommended 256-bit AES as the encryption algorithm to use. While it is not as fast as the 128-bit version, it is exponentially more secure, and the performance difference between the two versions diminishes as computers get faster. Also, by encrypting the medical data with 256-bit AES, one would not have to "upgrade" the encryption on the data

when, at some inevitable future time, 128-bit AES is deemed insecure. 128-bit AES has been deemed as secure for at least a decade. Since the Department of Radiology generates over 9 TB of digital data a year, this would lead to over 90 TB of data to be upgraded in a decade. This number does not even include data from the rest of the hospital. By using 256-bit AES now, one avoids the "upgrade" problem for the foreseeable future. Also, the AES implementation is in managed code. This means that the hospital does not have to worry about a hacker trying to exploit a buffer overrun or other programmer exploit in the encryption code to gain access to or delete sensitive data. The code is also portable enough to run on different platforms and can be explicitly denied access to certain parts of the system by the .NET trust scheme.

Using a workflow model that mimicked our radiology department, I was able to find the bottleneck and throughput of the system before any encryption was introduced. After adding encryption to the model, it became clear that while it did not affect the bottleneck of the actual patient exam, it did affect the overall performance of the system. Using a fully concurrent system, our model predicts that the patient throughput could be degraded by up to 7%. While this does not sound like an overwhelming number, for a hospital like the University of Virginia that operates on ~5% margin, this patient reduction could reduce or eliminate that modest margin. By using an algorithm with a throughput of at least 17 MB/s, the hospital would be able to still operate with at least a 3% margin. Previous published results have shown

assembly-level code for AES to run at over 30 MB/s. But with this choice the hospital would give up the benefits of managed code.

This thesis sought to contribute an answer to the question of what would happen to the workflow in a healthcare entity if the encryption standard set forth by HIPAA was implemented and enforced. It also sought to show some difficult decisions when dealing with performance vs. security. It was also intended to generate the exposure of HIPAA to people not working directly with healthcare entities, as there are many studies that can be done to predict other areas that might be affected by the new standards set forth in HIPAA. It also served as a performance analysis of the most popular encryption methods to date when implemented in a software environment based on the .NET framework architecture. The performance measurements show similar trends to previously published results. The recommendation of the thesis is to implement and use 256-bit AES algorithm in the healthcare workplace to both secure medical data and avoid future encryption upgrades that will be required. While there is a small performance degradation because of the encryption, it is not unmanageably large and could be trimmed given better system implementations. For these reasons, it will be important to plan the architecture to be implemented carefully or the impact could be serious.

6.2 Future Work

With the most ambitious security standards yet to be enforced, no one really knows what will happen to the workflow in the healthcare industry. This thesis dealt with one of many new standards that are required by HIPAA. The model used in this

thesis can be easily extended to deal with other areas of HIPAA as well, such as auditing and authorization. These two standards will also have an impact on system performance and throughput.

With new encryption implementations and/or faster computers, it may be possible to achieve the desired throughput to maintain an uninterrupted workflow in the Department of Radiology and the rest of the hospital. It would be useful to re-evaluate the choice of managed vs. unmanaged code once the performance gap narrows more than its' current variance.

It would also be useful to perform a cost analysis on implementing HIPAA changes, as each healthcare entity has to become compliant using its own resources. Also, is there any kind of monetary return in the long run by implementing these standards, as alleged by the HIPAA documents? Supposedly these standards will allow entities to communicate with less friction, as information will be in a universal format that will be easily transferred from place to place.

The model used in this thesis is the model used by the Radiology Department at the University of Virginia Hospital. It would be worthwhile to investigate different models to test the workflow impact in different environments. HIPAA affects more than healthcare entities; indirectly, it affects other groups such as lawyers and consultants. There are new processes in place that lawyers have to follow to subpoena medical records. A study on the impact of HIPAA on non-healthcare entities would be an intriguing study.

[1] Public Law 104-191, "Health Insurance Portability and Accountability Act of 1996," [Online] Available http://aspe.hhs.gov/admnsimp/pl104191.htm

[2] "Standards for Electronic Transactions", Federal Register, Volume 65, Number 160, August 17, 2000. [Online] Available http://aspe.hhs.gov/admnsimp/final/txfin00.htm

[3] Alfred C. Weaver and Samuel J. Dwyer III, "Federated, Secure Trust Networks for Distributed Healthcare IT Services," Microsoft Corporation, 1/1/03--12/31/04, $250,000.

[4] Stallings, William. "Cryptography and Network Security." Prentice Hall, 1999.

[5] RSA Security, Inc. "RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF)", [Online] Available http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=462

[6] Electronic Frontier Foundation. "Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design." July 1998.

[7] "Data Encryption Standard," FIPS Publication 46-3, October 25, 1999, [Online] Available http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[8] RSA Security, Inc. "What are the most important attacks on symmetric block ciphers? " [Online] Available http://www.rsasecurity.com/rsalabs/faq/2-4-5.html

[9] "Advanced Encryption Standard," FIPS Publication 197, November 26, 2001, [Online] Available http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[10] Orion Systems International. "HIPAA Compliant, Secure Encryption." [Online] Available http://www.orion.co.nz/rhapsody_security.htm

[11] NetSilica. "Healthcare and HIPAA." [Online] Available http://www.netsilica.com/nshome/healthcare.htm

[12] K. Gaj and P. Chodowiec, "Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays," Proc. RSA Security Conf. - Cryptographer's Track, San Francisco, CA, April 8-12, 2001.

[13] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," IEEE Transactions on VLSI, August 2001, vol. 9, no. 4, pp. 545-557.

[14] J. Kaps and C. Paar, "Fast DES Implementations for FPGAs and its Application to a Universal Key-Search Machine," in 5th Annual Workshop on Selected Areas in Cryptography (SAC '98) (S. Tavares and H. Meijer, eds.), vol. LNCS 1556, (Queen's University, Kingston, Ontario, Canada), Springer-Verlag, August 1998.

[15] Gladman, Brian. "AES Second Round Implementation Experience." [Online] Available http://fp.gladman.plus.com/cryptography_technology/aes2/

[16] K. Aoki and H. Lipmaa. "Fast implementations of aes candidates." In Proc. Third AES Candidate Conference, April 13-14, 2000.

[17] Corella, Francisco. "A fast implementation of DES and Triple-DES on PA-RISC 2.0," [Online] Available http://www.usenix.org/events/osdi2000/wiess2000/full_papers/corella/corella.PDF

[18] A. Sterbenz and P. Lipp. "Performance of the AES Candidate Algorithms in JavaTM." In The Third Advanced Encryption Standard Candidate Conference, pages 161-168, New York, New York, USA, April 13-14 2000. National Institute of Standards and Technology. [Online] Available http://csrc.nist.gov/encryption/aes/round2/conf3/papers/03-asterbenz.pdf

[19] Dray, J. "NIST Performance Analysis of the Final Round Java™ AES Candidates," in The Third AES Candidate Conference, printed by the National Institute of Standards and Technology, Gaithersburg, MD, April 13-14, 2000, pp. 149-160.

[20] M. Shand and J. E. Vuillemin. "Fast implementations of RSA cryptography." In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pages 252-259. IEEE Computer Society Press, 1993. http://www.cse.cuhk.edu.hk/~phwl/ceg5010/design/SV93.pdf

[21] Groszschaedl, Johann. "The Chinese Remainder Theorem and its Application in a High-Speed RSA Crypto Chip." 16th Annual Computer Security Applications Conference. December 11-15, 2000. New Orleans, Louisiana. [Online] Available http://www.acsac.org/2000/papers/48.pdf

[22] Wagner, N. "The Laws of Cryptography: The RSA Cryptosystem." [Online] Available http://www.cs.utsa.edu/~wagner/laws/RSA.html

[23] RSA Security, Inc. "How fast is the RSA algorithm?" [Online] Available http://www.rsasecurity.com/rsalabs/faq/3-1-2.html

[24] Dhawan, P. "Performance Comparison: Security Design Choices." Microsoft Developers Network. [Online] Available http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdadotnetarch15.asp

[25] Simpson, Sam. "PGP DH vs. RSA FAQ." [Online] Available http://www.scramdisk.clara.net/pgpfaq.html#Sub3DES

[26] S.Lucks, "Attacking Triple Encryption", Fast Software Encryption, 1998.

[27] Courtois N, Pieprzyk Josef. "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations," AsiaCrypt 2002, November 9, 2002, [Online] Available http://eprint.iacr.org/2002/044.pdf

[28] RSA Security, Inc. "Factorization of RSA-155." [Online] Available http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html

[29] RSA Security, Inc. "What key size should be used?" [Online] Available http://www.rsasecurity.com/rsalabs/faq/4-1-2-1.html

[30] IBM Research. "IBM's Test-Tube Quantum Computer Makes History." [Online] Available http://www.research.ibm.com/resources/news/20011219_quantum.shtml

[31] Andriole KP, Arvin DE, Yin L, Gould RG, Avenson RL. PACS database and enrichment of the folder manager concept. J Digital Imaging 2000: 13:3-12.

[32] Stuck BW, Arthurs E. A computer and communications network performance analysis primer. Prentice-Hall Inc., Englewood Clifts, NJ, 1985.

[33] Gay, Spencer B. Modeling for Workflow in Diagnostic Radiology Department. DHHS Grant Application, 2001.