RSAmazing Algorithm

New Mexico

Supercomputing Challenge

Final Report

March 29, 2007

Team 16

Artesia High School

Team Members

Craig Mayberry

Ashley Durham

Teacher

Randall Gaylor

Project Mentor

Matt Conn

Table of Contents:

Executive Summary:

The purpose of The RSAmazing Algorithm Project was to create a way to make the RSA Algorithm more secure and difficult to break. The RSA Algorithm is used to encrypt a message being sent to another person who has a private key (private password) that can be used to decrypt the message. Encryption takes a message, or any important information, and puts it in a form that no one can read unless they have a special "password". The main reason the RSA Algorithm is so strong is because it uses large numbers in the encryption process. Our goal was to find a way to enhance the algorithm without relying on length of numbers alone.

Our original theory was based on problems that occur in math class. In Calculus, large problems are very easy to get wrong. If you miss one step or incorrectly solve another, your answer can be very far away from the target. Our hypothesis was that if we added a simple cipher to an algorithm, it would work in the same manner, making an encryption method stronger than it would be on its own.

The first step in this process was creating a model of the RSA Algorithm. Our model was created with Java based on its user-friendliness. After we had a working computer model, we could test ideas and implement the speed of a computer at the same time. We are able to work the algorithm on paper, but it is not as time-efficient. From here, adding simpler algorithms to the process was extremely easy.

We experimented with various ways of combining simpler algorithms with the RSA Algorithm, such as a Caesar Shift. In contradiction with our hypothesis, we found that it did not always make the algorithm stronger. The Caesar Shift is a very simple algorithm that shifts a letter of the alphabet over a certain number of places (for example, a shift of one would change "A" to become "B", "B" to "C", and so on). In some cases, combining certain algorithms together can make it easier to break the original encryption instead of harder.

Introduction:

In the process of deciding a suitable project for the Supercomputing Challenge, our team immediately agreed that we wanted a project that was mathematically based. The most obvious application of mathematics in the computer world is in encryption. The RSA Algorithm, a widely renowned algorithm used in computer encryption, was selected by our team to be the basis of our study. The first weeks of our project were spent analyzing the properties of the algorithm and understanding its strengths and weakness. The weaknesses left a door wide-open for our team to enter.

Encryption, our field of study for this project, is the science of taking comprehensible data and obscuring it so that it can be protected from unwanted surveillance or knowledge. Much like grade-schoolers have code words and private languages amongst friends, large businesses and important agencies must use encryption in much the same way. The growing world of computers presents people with many opportunities, such as banking or shopping online. However, this also creates new dangers for predators trying to take advantage of this situation. Encryption is used as a safe-guard for those who roam the vast world of cyberspace. It can mask a credit card number, disguise private files, and keep your information secret from any but those you wish to see it.

The RSA Algorithm is a key-based algorithm. This means that in order to successfully encrypt or decrypt data with it, you need a numerical key, which acts as a password and tool in the encryption process. There are two keys, known as the public key and the private key. The public key, as its name implies, is generally accepted to be known by everybody. It is the key that is used in the encryption process. However, to a hacker desiring information, being able to encrypt data repeatedly isn’t much use. The private key, which is kept only between the people allowed to see the protected information, is the number used in the decryption process. The only logical attack in breaking an encryption by the RSA Algorithm would be to find the private key by using the public key. This can be done because both keys are produced using numbers of values that are close together.

This factor means that the major security factor keeping your credit card number safe is exactly how big those numbers are. The numbers are measured in bit-length to give computer analysts a defined variable on how tough an encryption is to crack. Currently, many individuals used keys that are computed with numbers over 300 digits in length. As technology increases, the predicted amount of time information encrypted with certain bit-lengths of keys drops. The only way to increase the security of the RSA Algorithm is by using larger and larger numbers. Our team wanted to explore the possibility of modifying the algorithm so that it didn’t rely on brute force for its security.

The first and most challenging portion of our project was to create a computer program that could successfully encrypt and decrypt our input. Modeled from the mathematical backbone of the RSA Algorithm, we coded a program in Java that would serve our purpose. The reason we chose Java was for its user friendliness and ease of debugging. It was a language we were more comfortable with, being web-based, and not learning a new language would save us time in our research. After our program worked, we experimented combining the RSA Algorithm with a simple Caesar Shift. A Caesar Shift is a cipher that adds a number value to a specified letter, say A, and creates a new letter from that (if A = 1, adding one would be 2, resulting in letter B).

Our hypothesis stated that in combining these ciphers, our result would be information encrypted with two methods, making it stronger and less permeable to an attack. In examining our results, we realized that even if the RSA Algorithm and the Caesar Shift were combined, the

Caesar shift would be a walk in the park to break if someone successfully broke the RSA Algorithm before it. Since the Caesar Shift didn't yield a wide margin of difficulty when added after an RSA encryption, we decided to try using it beforehand. If the public key was modified with a Caesar Shift, or any other form of encryption, it would be much more difficult for a criminal with malevolent intent to break the code. Since the public key is in numerical form, it can be shifted many times, but will always yield the same thing: a number. When the Shift is applied to data already decrypted from the RSA Algorithm, it only has to be processed until the information is in a form that is comprehensible. If the information doesn't make sense, you only need to try again. The correct public key, however, is not distinguishable from any other key. If a key had only 50 digits, which is minimal security, there would be 10^{50} different possibilities to test. A trial and error approach to a problem such as this would greatly increase the time needed to decrypt data.

Code:

```
<html>

        <head>

                <script type="text/javascript">

                        var letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
.,:?'-/()\"";
                        var numbers = new Array("56", "55", "54", "53", "52",
                                                                        "51",
"50", "49", "48", "47",
                                                                        "46",
"45", "44", "43", "42",
                                                                        "41",
"40", "39", "38", "37",
                                                                        "36",
"35", "34", "33", "32",
                                                                        "31",
"30", "29", "28", "27",
                                                                        "26",
"25", "24", "23", "22",
                                                                        "21",
"20", "19", "18", "17",
                                                                        "16",
"15", "14", "13", "12",
                                                                        "11",
"10");

                        function getPrimes(limit) {
                                var result = new Array();
                                var primes = new Array(limit +  1);
                                var current = 2;
                                for (var i = 2; i <= limit; i++) {
                                        primes[i] = true;
                                }
                                while (current <= Math.sqrt(limit)) {
                                        if (primes[current]) {
                                                var multiple = 2 * current;
                                                while (multiple <= limit) {
                                                        primes[multiple] =
false;

                                                        multiple += current;
                                                }
                                        }
                                        current++;
                                }
                                for (var i = 2; i <= limit; i++) {
                                        if (primes[i]) {
                                                result[result.length] = i;
                                        }
                                }
                                return result;
                        }
                        function buildSelect() {
                                var primes = getPrimes(10000);
                                var select1 = document.parameters.P;
                                var select2 = document.parameters.Q;
                                for (var i = 0; i < primes.length; i++) {
                                        var option1 =
document.createElement("OPTION");
                                        var option2 =
document.createElement("OPTION");
                                        option1.text = primes[i];
                                        option2.text = primes[i];
                                        option1.value = primes[i];
                                        option2.value = primes[i];
                                        select1.options.add(option1);
                                        select2.options.add(option2);
                                }
                        }

                        function gcd(a,b){
                                while(b != 0) {
                                        temp = b;
                                        b = a % b;
                                        a = temp;
                                }
                                return a;
                        }

                        function calcN() {
                                var N = parseInt(document.parameters.P.value,
10) * parseInt(document.parameters.Q.value, 10);
                                document.parameters.Nfield.value = N;
                        }

                        function calcM() {
                                var M = (parseInt(document.parameters.P.value,
10) - 1) * (parseInt(document.parameters.Q.value, 10) - 1);
                                document.parameters.Mfield.value = M;
                        }

                        function validatePQ() {
                                if (document.parameters.P.value ==
document.parameters.Q.value) {
                                        alert("P and Q cannot be equal.");
                                        return false;
                                }
                                else {
                                        return true;
                                }
                        }

                        function calcE() {
                                if (validatePQ()) {
                                        var M =
document.parameters.Mfield.value;
                                        var E = 2;
                                        for (var E = 2; gcd(E, M) != 1; E++) {}
```

```
                                      document.parameters.Efield.value = E;
                               }
                       }

                       function powMod(base, exp, modulus) {
                          var accum = 1;
                          var i = 0;
                          var basepow2 = base;
                          while ((exp >> i) > 0) {
                                 if(((exp >> i) & 1) == 1) {
                                        accum = (accum * basepow2) % modulus;
                                 };
                                 basepow2 = (basepow2 * basepow2) % modulus;
                                 i++;
                          };
                          return accum;
                       }

                       function eeaD(a, b) {
                               var a0 = a;
                               var x = 1;
                               var v = 1;
                               var y = 0;
                               var u = 0;
                               var q;
                               var r;
                               var newu;
                               var newv;
                               while (b != 0) {
                                      q = Math.floor(a / b);
                                      r = a % b;
                                      a = b;
                                      b = r;
                                      newu = x - q*u
                                      newv = y - q*v
                                      x = u;
                                      y = v;
                                      u = newu;
                                      v = newv;
                               }
                               return ((y > 0) ? y : y + a0);
                       }

                       function encrypt() {
                               var E = document.parameters.Efield.value;
                               var Binput =
parseInt(document.numericEncryption.Binput.value, 10) ;
                               var N = parseInt(document.parameters.P.value,
10) * parseInt(document.parameters.Q.value, 10);
                               var Coutput = powMod(Binput, E, N) ;
                               document.numericEncryption.Coutput.value =
Coutput;
                       }

                       function decrypt() {
                               var D = document.parameters.Dfield.value;
```

```
                                var Cinput =
parseInt(document.numericEncryption.Cinput.value, 10) ;
                                var N = (parseInt(document.parameters.P.value,
10)) * (parseInt(document.parameters.Q.value, 10));
                                var Boutput = powMod(Cinput, D, N);
                                document.numericEncryption.Boutput.value =
Boutput;
                        }

                        function translate(inputField, outputField) {
                                var output = "";
                                var input = inputField.value.toUpperCase();
                                for(var count = 0; count < input.length;
count++) {
                                        daChar = input.charAt(count);
                                        for (var i = 0; i < letters.length;
i++) {
                                                if (daChar == letters.charAt(i))
{
                                                        output += numbers[i] + "
";
                                                        break;
                                                }
                                        }
                                }
                                outputField.value = output;
                        }

                        function translateBack(inputField, outputField){
                                var input = inputField.value;
                                var output = " " + input + " ";
                                output = output.replace(/ /gim, "~~");
                                alert(output);
                                for (var i = 0; i < letters.length; i++){
                                        if (letters.charAt(i) != " ") {
                                                output = output.replace(new
RegExp("~" + numbers[i] + "~", "gim"), "~" + letters.charAt(i) + "~");
                                        }
                                }
                                output = output.replace(/~~~~/gim, " ");
                                output = output.replace(/~/gim, "");
                                output = output.replace(/~~~~/gim, " ");
                                outputField.value = output;
                        }

                </script>
        </head>


        <body onload="buildSelect()">

                <form name="parameters">
                        <p>
                                P <select name="P"
onchange="validatePQ()"></select>
                                Q <select name="Q"
onchange="validatePQ()"></select>
```

```
                        </p>
                        <hr />
                        <p>
                                N = <input type="text" name="Nfield">
                                <input type="button" name="Calculate N"
value="Calculate" onClick="calcN()">
                        </p>
                        <p>
                                &#934 = <input type="text" name="Mfield">
                                <input type="button" name="Calculate M"
value="Calculate" onClick="calcM()">
                        </p>
                        <p>
                                E = <input type="text" name="Efield">
                                <input type="button" name="Calculate E"
value="Calculate" onClick="calcE()">
                        </p>
                        <p>
                                D = <input type="text" name="Dfield">
                                <input type="button" name="Calculate D"
value="Calculate" OnClick="this.form.Dfield.value =
eeaD(parseInt(this.form.Mfield.value), parseInt(this.form.Efield.value))">
                        </p>
                </form>

                <hr />

                <form name="numericEncryption">
                        <p>
                                Message = <input type="text" name="Binput"
size="40">
                                <input type="button" name="Encrypt"
value="Encrypt" onClick="encrypt()">
                        </p>
                        <p>
                                Encrypted message C = <input type="text"
name="Coutput" size="40">
                        </p>
                        <hr />
                        <p>
                                Message C = <input type="text" name="Cinput"
size="40">
                                <input type="button" name="Decrypt"
value="Decrypt" onClick="decrypt()">
                        </p>
                        <p>
                                Decrypted message = <input type="text"
name="Boutput" size="40">
                        </p>
                </form>

                <hr />

                <form name="conversion">
                        <p>
                                <textarea name="textInput" cols="20" rows="4"
wrap="physical"></textarea>
```

```
                              <input type="button" value="To Number"
onClick="javascript:translate(this.form.textInput, this.form.numericOutput)">
                              <textarea name="numericOutput" cols="20"
rows="4" wrap="physical"></textarea>
                       </p>
                       <p>
                              <textarea name="numericInput" cols="20"
rows="4" wrap="physical"></textarea>
                              <input type="button" value="To Letter"
onClick="javascript:translateBack(this.form.numericInput,
this.form.textOutput)">
                              <textarea name="textOutput" cols="20" rows="4"
wrap="physical"></textarea>
                       </p>
               </form>

        </body>

</html>
```

Description and Methods:

The methods we used in pursuit of our goal involved becoming familiar with the RSA Algorithm using small, single-digit prime numbers. This was made easier as our mentor has done a great deal of study related to the RSA Algorithm. We broke down the algorithm and learned how each step in the process worked. After that we developed our own Java-based program that would follow the same methods.

This program has several functions within it that enable it to run in much the same was as the RSA Algorithm. The first function in the code of the program is the function “getPrimes”. This function does just what its name implies; it generates an extensive list of prime numbers. These are the prime numbers that will be used as the bases for the rest of the algorithm. The next functions are the functions “buildSelect” and “buildSelectTwo”. The functions enable the user to select which two of their list of prime numbers will be their p and q. The function “gcd” calculates the greatest common denominator between the p and the q. This will prove very useful in future steps of the algorithm. Next are the functions “calcN” and “calcM”. The latter of the two, the M is actually representing what is represented as phi in the RSA Algorithm. The functions “calcE”, “powMod”, and “eeaD” are all used to determine the final variables of the algorithm; the d and e. Much like the first function in the program, the next two functions do just as their name implies. The function “encrypt” encrypts the given message, and the function “decrypt” takes the encrypted message and decrypts it back into the original. The final two functions; “translate” and “translate_back” convert letters to a number value and vice versa. All of these steps combine to make up our program that enables a simple running of the RSA Algorithm.

Research:

- History of Encryption
- Various Methods of Encryption (RSA, DES/3DES, Blowfish, Idea, Seal, RC4)
- Affine Cipher
- Caesar Shift
- RSA Algorithm
- Euclidean Algorithm

Results and Conclusion:

The conclusion our team arrived at was that some combinations of encryption do not necessarily increase the strength of that particular algorithm. In addition, some combinations can produce unforeseen enigmas that weaken the algorithm. In this way, combining encryption methods is extremely risky if thorough research is not involved beforehand. Unfortunately we were ultimately unable to achieve our goal of making the RSA Algorithm even stronger and more effective. We had hoped that incorporating a Caesar Shift would allow us to enhance it, but we realized through results that a Caesar Shift would be relatively simple to render ineffective. In this way, higher levels of algorithms may not be able to be combined to create a much securer and safer realm for our personal information, bank account numbers, ISP security passwords, and even pictures of our grandchildren.

Acknowledgements:

Mr. Matt Conn for being our mentor and our main guide in learning the basics of the RSA Algorithm.

Mr. Randall Gaylor for being our sponsor, organizing our events, and keeping us on track.

Mr. Nick Bennett for helping us to organize our thoughts into something we could use to make a program and donating suggestions to clean up our final product.

Works Cited

"Java Technology." Java. 1994. <http://www.java.sun.com>.

"RSA Algorithm." Di-Mgt. 2 Oct. 2006. DI Management Services. 4 Nov. 2006 <http://www.di-mgt.com.au/rsa_alg.html>.

"RSA." Wikipedia. 2 Apr. 2006. Wikimedia Foundation. <http://en.wikipedia.org/wiki/RSA>.