RSA CRYPTOGRAPHY, THE SOLUTION TO THE PUBLIC-KEY PROBLEM

[redacted]

ABSTRACT. This paper briefly discusses the public-key problem before delving into the RSA cryptography algorithm. We see security of the system demonstrated and discuss a few possible attacks on the system along with the corresponding defenses.

1. THE PUBLIC-KEY PROBLEM

The dawn of the computing age has brought in a myriad of mathematical problems and opportunities. Many of these modern problems can be solved using one of the oldest branches of mathematics: number theory. One of the most intriguing of these is the issue of public-key cryptography.

Obviously any network of computers, most recently and expansively the Internet, provides a massive channel for the exchange of ideas and information. But what if one wants to exchange a message in secret? In more primitive times, any coded message would have to be encrypted using a symmetric key, that is, an encryption system that would be shared by the message sender and receiver and kept secret from all others. But this presented many problems: the code had to be exchanged through a secure network, the code could be stolen and all messages would be completely compromised until an entirely new symmetric code could be formulated and exchanged, and so on. If there were a way to encrypt messages through a code visible to all, but only able to be decrypted by the intended receiver, a sort of public key cryptography system, then secure messages could easily be sent over insecure methods.

The public-key cryptography problem was shaken by the introduction of the Diffie-Hellman method [3] in 1976 in the paper "New Directions in Cryptography," which introduced the idea of using modular arithmetic calculations to provide a secure message transfer system over an insecure network. The following year, at the Massachusetts Institute of Technology, the public-key cryptography problem was solved by

three professors: computer science professor Ronald Rivest and mathematics professors Adi Shamir and Leonard Adleman. The cryptography algorithm is named after the first initials of their last names, RSA.

(Note: a system very similar to RSA was developed independently in secret by Clifford Cocks at Government Communications Headquarters in Cheltenham, England, in 1973, but it was not released until well after RSA had been published and implemented.) [1]

2. How RSA works

The RSA algorithm follows a few simple arithmetic and modular arithmetic calculations. First, select two primes, and call them p and q. Then perform the following calculations:

$$n = pq$$

$$\Phi(n) = \Phi(p) * \Phi(q) = (p-1)(q-1)$$

Next choose an integer e such that $1 < e < \Phi(n)$ and $gcd(e, \Phi(n)) = 1$. Since e and $\Phi(n)$ are relatively prime, then there exist $k, d \in \mathbb{Z}$ such that

$$de + k\Phi(n) = 1 \tag{1}$$

by Euclid's Lemma. Find d by the division algorithm.

We now have our public key e, our private key d, and our modulus n, which is also made public, by which all further modular arithmetic calculations will be made.

To encode a message, Person A looks up Person B's e and n. Then A takes his message m and raises it to B's e power, modulo B's n.

$$m^e \equiv c (\mathrm{mod}\, n) \tag{2}$$

The resulting number c is then sent to B. To decode the message, B takes the received message c and raises it to the power of his private key d modulo n.

$$c^d = (m^e)^d = m^{ed} \equiv m (\mathrm{mod}\, n)$$

The result is the original message m.

The validity of this algorithm is based on the following theorem:

Theorem 2.1. *Let n be the product of 2 primes, let e be relatively prime to $\Phi(n)$, and let d exist such that $de \equiv 1 (\mathrm{mod}\, \Phi(n))$. Then for all $b \in \mathbb{Z}$, $b^{de} \equiv b (\mathrm{mod}\, n)$.*

Proof [1]: Since $de \equiv 1(\mathrm{mod}(p-1)(q-1))$, we know that

$$de \equiv 1(\mathrm{mod}(p-1))$$

$$de \equiv 1(\mathrm{mod}(q-1))$$

By Fermat's Little Theorem, we get

$$b^{de} \equiv b^1(\mathrm{mod}\, p)$$

$$b^{de} \equiv b^1(\mathrm{mod}\, q)$$

Since $p \mid (b^{de} - b)$ and $q \mid (b^{de} - b)$, it follows that

$$lcm(p,q) = pq \mid (b^{de} - b)$$

Therefore, $b^{de} \equiv b(\mathrm{mod}\, n)$, as desired.

So, anyone can encrypt a message to send to A, but once encrypted, the message cannot be decrypted by anyone other than A, not even the sender of the message.

3. Digital signatures

The problem with such a public-key system is that anyone can create a message to send to B using B's encryption data. How can B be sure from whom the message is? Fortunately, RSA solves that problem as well, by the use of digital signatures.

A raises his signature s to the power of his private d modulo his n.

$$s^d \equiv t(\mathrm{mod}\, n)$$

Then A sends the resulting t to B along with the message encrypted using B's e and n. Then all B has to do is raise t to the power of A's public e to retrieve the original signature. Note that this is not a secure signature. Anyone who intercepts the message can verify A's signature. But the message itself is still safe.

The problem lies in the security of the signature. Not only can any outside interceptor determine that A sent the message, but the interceptor can also send messages to B claiming to be A, and vice versa. So A can equip what is called a "hash function," which is generally agreed to be a function $Hash$ with the following properties:

1: Given any h, it should be very difficult to find an m such that $Hash(m) = h$.

2: Given any m, it should be very difficult to find an $n \neq m$ such that $Hash(m) = Hash(n)$.

3: Given any $Hash(m)$, it should be very difficult to find an $n \neq m$ such that $Hash(m) = Hash(n)$.

This hash function is then applied to A's signature and to a part of the encrypted message that A sends to B. Then, when B decodes the

message, the two hash values will match if and only if the sender has A's private key.

4. How secure is it?

We have seen that the RSA algorithm is an effective way to transfer information. But what security risks exist? Just how secure is the system? If an outsider, Person X, wants to decode the message c from equation 2, without knowing B's private decryption key, he has several options. First, he can attempt to calculate $c^{-e} (\mathrm{mod}\, n)$ directly. However, no reliable method for this exists. So, X must use a more roundabout way to find the message m. He knows that $de + k\Phi(n) = 1$ by equation 1. e is public knowledge, so if X can determine what $\Phi(n)$ is, he can deduce d from the division algorithm. Determining $\Phi(n)$ requires factoring n into its two prime factors. This presents a problem. The primes p and q chosen may be over 300 digits long (1024 bits). Even if one possible combination of primes could be run every nanosecond (10^{-9} second), it would still take well over 10^{50} years to run them all. The problem of factoring large numbers continues to be of interest; in fact, in 1991 RSA Laboratories put forth a challenge to factor several large RSA numbers (that is, numbers that are the product of two primes) with cash awards for the first to factor each number. [1] Several were factored, the largest of which was 200 decimal digits long, but many remain unfactored. Most factored RSA numbers took years to break down.

So it may appear that we are out of the woods. Any interceptor cannot decipher the message without knowing d, nor can he determine what d is by mathematical force. However, unfortunately, slightly more cunning techniques of cracking the system do exist. One of the more interesting approaches is called a timing attack. [1] Person X determines the hardware capabilities of B's machine and records decryption times for several messages. From this, X can quickly narrow down a range for d and find the value for d relatively quickly.

To safeguard against a timing attack, B can take the message c from A and multiply it by a secret random number r and computes $r^e c (\mathrm{mod}\, n)$. B then decodes using the secret exponent d to obtain

$$(r^e c)^d = r^{ed} m^{ed} \equiv r * m (\mathrm{mod}\, n)$$

B then simply multiplies by the inverse of r to obtain the message m. As long as r is randomly generated every time a new message is decrypted, the timing will be thrown off enough that X cannot determine d reliably.

Another problem is the threat of a "man-in-the-middle" attack, in which Person X presents their own encryption keys, masquerading as Person B, to Person A, in an attempt to get Person A to send a message intended for Person B across the network, in prime position for X to decode it. This problem can be averted by attaching a digital certificate to a public key, assigned by a trusted third party.

5. Other problems of RSA

RSA cryptography depends on the use of modular arithmetic to ensure security. If the calculations were not taken modulo our n, it would be an easy task to take the e^{th} root of our encrypted message to determine the content of the message. But what if, by some unlucky flaw, we code the message in such a way that raising certain parts of a message to the e^{th} power gave us a c less than our n? Our interceptor X could simply take the e^{th} root of this c without regard to our modulus n. Also, no matter what e and n are, inputs of 0 and 1 give outputs of 0 and 1, respectively. What can be done to ensure absolute security of these small values?

Ingeniously, cryptographers can make use of what is known as a "padding scheme." In a padding scheme, parts of our m are run through a function that amplifies their values to a point that, while each part remains less than n, by necessity that all distinct inputs remain distinct modulo n, each output of our padding scheme is sufficiently large to end up larger than n when raised to the e^{th} power. [1]

In addition to making n very difficult to factor, working with such large primes raises other issues. How can such a large calculation as raising a 300+ digit number to a power in the hundreds or thousands? Computers are able to do these without doing hundreds of large calculations by a method called exponentiation by squaring. Here's an example: [4]

Say we want to calculate 29^{47}. Note that 47 is equal to 101111 in binary. So

$$47 = 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0$$

From this, we can break up 29^{47} as follows:

$$\begin{aligned} 29^47 &= 29^{2^5+2^3+2^2+2^1+2^0} \\ &= 29^{2^5} * 29^{2^3} * 29^{2^2} * 29^{2^1} * 29^{2^0} \\ &= 29^{2*2*2*2*2} * 29^{2*2*2} * 29^{2*2} * 29^2 * 29 \\ &= ((((29^2)^2)^2)^2)^2 * ((29^2)^2)^2 * (29^2)^2 * 29^2 * 29 \end{aligned}$$

At first glance this looks no simpler. But notice that now we must only do 15 calculations, instead of the 46 required to multiply out 29*29*29*...*29. Also, we are greatly assisted by the fact that after each calculation, we can perform a modular reduction to get smaller numbers to work with. By this method, large exponential calculations can be done extremely quickly.

6. Conclusion

So, we see that a relatively simple mathematical solution, applied through modular arithmetic, has solved one of history's most intriguing problems, at least for the moment. New forms of aggressive decrypting are being developed every day, and those who seek to protect the security of the system must work tirelessly to this end. But, amazingly, in this dynamic fast-paced world, one technique has stood the test of 30 years of attacks and remains seemingly uncrackable in the foreseeable future. RSA has transformed the way we work and secure our data. One can only imagine what else future mathematicians can find to use it for, and what other fantastic solutions will arise to cryptographical problems.

References

[1] "RSA." *Wikipedia: The Free Encyclopedia.* 28 May 2007. 28 May 2007. <http://en.wikipedia.org/wiki/RSA>

[2] "Cryptogenic hash function." *Wikipedia: The Free Encyclopedia.* 22 May 2007. 28 May 2007. <http://en.wikipedia.org/wiki/>

[3] Robinson, Sara. "Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders." *SIAM News* Volume 36, Number 5, June 2003. 28 May 2007. <http://www.msri.org/people/members/sara/articles/rsa.pdf>

[4] "Prime Number Hide-and-Seek: How the RSA Cipher Works." *Muppetlabs.* 28 May 2007. <http://www.muppetlabs.com/ breadbox/txt/rsa.html>

[5] "RSA Algorithm." *DI Management.* 2 Oct. 2006. 28 May 2007. <http://www.di-mgt.com.au/>

The Ohio State University
E-mail address: [redacted]