MATH 168: FINAL PROJECT
Troels Eriksen

1 Introduction

In the later years cryptosystems using elliptic curves have shown up and are claimed to be just as secure as a system like RSA with much smaller key sizes. This makes it faster and less resource demanding, and hence allows implementation in a wide array of applications. It is generally accepted [1] that a 160-bit elliptic curve key provides the same level of security as a 1024-bit RSA key.

This project will be a discussion about the security of cryptosystems based primarily on a comparison between the classical and widely used RSA and the elliptic curve variant of the El-Gamal system.

Both systems are public key systems, meaning that there is a secret key used for decryption and a publicly available key used for encrypting. In theory everyone can send you an encrypted message by using the public key (if they know where to obtain it), but you're the only one who can decrypt the message. Of course these systems rely on the difficulties in obtaining the private key, when knowing the public key and some mathematical properties of the used cryptosystem.

Over the last years breaking cryptosystems, also known as the art of cryptoanalysis has received more and more attention. This is easy to understand as cryptography plays a bigger and bigger role in our life. It is used to store sensitive informations like pin-codes, bank informations and sending protected informations e.g. over the Internet. For the last many years the people behind RSA have been posting competitions to factor large numbers on the RSA-website with rather large money prizes. But this doesn't mean that average John Doe can take out his PET 2001 from the closet and hope to gain some easy money by writing a quick basic program to factor numbers. It has turned out to be very difficult and extremely time consuming to factor numbers as they get large enough. There is no known easy way around, but there are better ways than plain brute force. We will look at these methods later.

2 Overview of the cryptosystems

2.1 RSA

The RSA cryptosystem was invented by R. L. Rivest, S. Shamir, and L. M. Adleman in 1978. The simplicity of the RSA system is almost shocking. Everyone with just a tiny knowledge about mathematics can understand the basics behind the system. The basic idea about the RSA system is that it is far easier to find two large primes and multiply them together than it is to find the two primes given their product. The system is based on an old theorem by Euler, which is a generalization of Fermat's Little Theorem:

Theorem 2.1 (Euler's Theorem)**.** *If a, m are integers satisfying* $\gcd(a, m) = 1$*, then* $a^{\phi(n)} \equiv 1 \pmod{m}$

A proof for this theorem can be found in most elementary number theory books.

The RSA systems consists of a public and a private key, where the public key is used for encryption and the private for decryption. The two keys are naturally tied together in a clever way to make this work.

Encrypting and decrypting using RSA:

Public key Consists of two integers n and e: n is called the modulus and is a product of two primes p and q, and e is called the encryption exponent and is an integer satisfying $1 < e < \phi(n)$ and $\gcd e, \phi(n) = 1$. As n is a product of two primes, $\phi(n)) = (p-1)(q-1)$

Private key Consists of two integers n and d: d is the decryption exponent, and is an integer satisfying $1 < d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$.

Encrypting is done by splitting the plain text m in parts and computing $c = m^e \bmod n$

Decrypting is done by computing $m = c^d \bmod n$. Decryption works as $c^d \equiv (m^e)^d \equiv m \pmod{n}$.

It could seem fairly easy to break a system like RSA given all the informations one knows. The only problem is that $\phi(n)$ isn't known (it is most likely deleted right after the keys are generated), so in order to obtain the private key d, only knowing n and e, one would most likely need to know $\phi(n)$, and the whole idea about RSA is that this number is pretty hard to calculate without knowing p and q (which are also deleted right after the key generation). In fact, it has been proved that calculating $\phi(n)$ is equal to obtaining p and q, hence factoring n:[2, p22]

Theorem 2.2. *Suppose that $n = pq$ is a product of two distinct primes. Then determining $\phi(n)$ is equivalent to factoring n.*

Proof: Knowing the factors p and q we can easily calculate $\phi(n) = (p-1)(q-1)$. On the other hand, if $\phi(n)$ is known then it is also easy to compute the factors p and q. We know that $\phi(n) = (p-1)(q-1) = pq-(p+q)+1 = n-(p+q)+1$. Let $2b = n+1-\phi(n)$. Then we have that $p+q = 2b$ and $pq = n$. Two numbers whose sum is $2b$ and whose product is n must be solutions to the quadratic equation $x^2 - (2b)x + n = 0$, and hence p and q can be found by solving this equation. □

There are no theorems saying that factoring numbers is hard to do, but with the current known methods, factoring numbers of the sizes normally used in RSA, is a *very* time demanding task.

2.2 Elliptic curves - The El-Gamal system

There are lots of other places in number theory that has a one way property like the RSA system, and therefor can be used to build cryptosystems. One of them is raising to a power within a finite field. Suppose we have a finite field $(\mathbb{Z}/n\mathbb{Z})^*$ or $\mathbb{F}_q$, then it is fairly easy to compute b^x even for large values of x, but given an element y which is known to be of the form b^x, then computing $x = \log_b(y)$ is very time consuming. This is known as the discrete log problem. In this section, the elliptic curve analogue to the El-Gamal cryptosystem will be presented. The el-gamal system uses the discrete log problem, and can also be constructed without the use of elliptic curves.

E is an elliptic curve over a finite field $\mathbb{F}_p$. Let P be a point on the elliptic

curve, and suppose that P has prime order n. Now the cyclic subgroup of $E(\mathbb{F}_p)$ generated by P is:

$$\langle P \rangle = \{\mathcal{O}, P, 2P, 3P, \ldots, (n-1)P\}$$

Encrypting and decrypting using the elliptic curve El-Gamal system: Everyone knows the prime p, the equation of the elliptic curve E, and the point P and its order n.

Private key The private key is an integer d that is selected uniformly at random from the interval $[1, n-1]$.

Public key The corresponding public key is $Q = dP$.

Encrypting First represent the plaintext message m as a point M on the elliptic curve (in $E(\mathbb{F}_p)$). Select a $k \in [1, n-1]$. Compute $C_1 = kP$ and $C_2 = M + kQ$; C_1 and C_2 are the ciphertext, which are sent to the recipient.

Decrypting Compute the point $M = C_2 - dC_1$, and return the plaintext m. The reason why this works is because $dC_1 = d(kP) = k(dP) = kQ$

To break the system, one would need to determine the private parameter d given the public available parameters p, E, P, Q, and n. This might seem even easier than in the RSA case with all these parameters known, but in order to determine this d one would have to solve the so called elliptic curve discrete logarithm problem which will be discussed in section 2.3.2.

2.3 Comparison

When choosing an algorithm for cryptography one has to consider things like the security and the performance. How can we be sure that the system is safe? Will the system be usable at all with the desired security level, or will it be too performance demanding? As stated earlier there are no theorems stating anything about either factoring numbers or solving the elliptic curve discrete logarithm problem being hard, so in order to measure and compare the level of security one has to find other methods. Usually what one does is

compare the efficiency of the algorithms by looking at the number of steps needed for an algorithm to do the desired job. That way you can get an upper bound on the time needed for the algorithm to finish, which of course is a function of the key-size. An accepted notation for efficiency of an algorithm with input of bitlength l is using the O-notation[1]:

Definition 2.1 (O-notation)**.** *If f and g are two positive real-valued functions defined on the positive integers, then we write $f = O(g)$ when there exist positive constants c and L such that $f(l) \leq cg(l)$ for all $l \geq L$.*

The above definition means that $f(l)$ grows no faster than $g(l)$ within a constant multiple.

Definition 2.2 (o-notation)**.** *We write $f = o(g)$ if for any positive constant c there exists a constant L such that $f(l) \leq cg(l)$ for $l \geq L$.*

The above definition means that $f(l)$ becomes insignificant relative to $g(l)$ for large values of l.

Definition 2.3. *Let A be an algorithm whose input has bitlength l.*

1. *A is a polynomial-time algorithm if its running time is $O(l^c)$ for some constant $c > 0$.*
2. *A is an exponential-time algorithm if its running time is not of the form $O(l^c)$ for any $c > 0$.*
3. *A is a subexponential-time algorithm if its running time is $O(2^{o(l)})$, and A is not a polynomial-time algorithm.*
4. *A is a fully-exponential-time algorithm if its running time is not of the form $O(2^{o(l)})$.*

What this definition tells us is how fast an algorithm grows, which can then be compared to another algorithm. Let's say we have a polynomial-time algorithm and a fully-exponential-time algorithm. The exponential one might be faster for small inputs, but as the running time of exponential one grows fastest of the two, the polynomial algorithm will be the fastest at some point

when the input is large enough. Of course it's also possible to compare two exponential-time algorithms and so on.

We will know look at the efficiency of some of today's best algorithms for factoring large integers and solving the elliptic curve discrete log problem.

2.3.1 Integer factorization

Breaking an RSA system would mean that one would have to factor the modulus n. These days, the best algorithm for factoring large integers is the Number Field Sieve (NFS), which was first proposed by J. M. Pollard in 1988 [3] and has since been refined. This algorithm currently has a subexponential running time of:[1]

$$L_n[\frac{1}{3}, 1.923].$$

where a subexponential algorithm has a running time of the form

$$L_n[\alpha, c] = O\left(e^{c+o(1)(\log n)^{\alpha}(\log\log n)^{1-\alpha}}\right)$$

The Number Field Sieve is a pretty complicated algorithm, and would take up too much space to explain in detail in this report. Very briefly told, the core of the NFS algorithm, is to find a congruence of squares. A random solution to the congruence $x^2 \equiv y^2 \pmod{n}$ has a 50% chance of producing a non-trivial factor of n. The algorithm consists of two stages [2, 1]:

1. A sieving stage where relations are collected to limit the amount of needed calculations. This can be done as a distributed calculation, e.g. using computers connected via the Internet, and hence saving time.
2. A matrix stage where a large sparse system of linear equations is solved. This is most efficiently done on one (really big) computer capable of performing a lot of parallel calculations, as there will have to be a lot of communications during the calculations.

The largest RSA prize number that has been factored so far is the RSA-640 — a number of 193 decimal digits.:

```
3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609
```

This number was factored on november 2nd 2005 by F. Bahr et al. and gave a money price of $20000[4]. According to the factoring team, the effort took approximately 30 2.2GHz-Opteron-CPU years, and over five months of calendar time. Notice that this is only one $\sim$ 200 digit number being factored, so all the remaining 200 digit modulus does still give "fairly secure" RSA keys.

Today a minimum key-size of 1024 bits is recommended for use with RSA[5]. If we assume that Moore's Law continues to hold, then it would take a $10 million machine 10 days in the year 2015 to factor a 1024 bits RSA-key.[5] Although a $10 million computer will most likely still be considered expensive for a regular person in 2015 (I don't intent to write a big economical discussion), it is obvious that a 1024 bit key will not continue to be considered as secure. We also have to remember that $10 million is a drop in the ocean for a government agency or some large company. Why don't we just move to 2048 bits or 4096 bits or even higher right away then? The answer to this is, that using larger key-sizes also affects the time it takes to encrypt/decrypt messages, and one would usually want these computations to take as little time as possible. For very important data or data that needs to be securely encrypted for several years is already recommended to use larger RSA key-sizes of up to 3072 bits.[5]

The table below [6] shows the estimated computing power required to factor integers with the current version of the Number Field Sieve. A MIPS year is equivalent to the computational power of a computer that is rated at 1 MIPS and utilized for one year. A Pentium 4 processor clocked at 2.8GHz is capable of performing approximately 5300MIPS (according to SiSoft Sandra [7]).

size of n (in bits)	MIPS years
512	$3 \cdot 10^4$
768	$2 \cdot 10^8$
1024	$3 \cdot 10^{11}$
1280	$1 \cdot 10^{14}$
1536	$3 \cdot 10^{16}$
2048	$3 \cdot 10^{20}$

In some special cases the RSA system is very easy to break, but these are well known and documented, so they can be avoided in the key generation.

- If the size of the two primes p and q are roughly the same, then p and q can be found by searching for factors near $\sqrt{n}$. So we need two primes that are both large, but on the other hand not too close to each other.
- If the private exponent d is too small, then the system is easily breakable.[8]
- If one knows of a cleartext m or ciphertext c that isn't coprime to n, then their greatest common divisor can be found using Euclid's algorithm, and the result must be either p or q. But as $\phi(n) = (p-1)(q-1)$, the numbers not coprime to n is extremely small, so the chance of finding one is almost non existing.

2.3.2 Elliptic Curve Discrete Logarithm Problem

The Elliptic Curve Discrete Logarithm Problem (ECDLP) can be formulated in the following way:[1, p. 153]

Definition 2.4 (ECDLP)**.** *Given an elliptic curve E defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q \in \langle P \rangle$, find the integer $d \in [0, n-1]$ such that $Q = dP$. The integer d is called the discrete logarithm of Q to the base P, denoted $d = \log_P Q$.*

When solving the ECDLP, we're trying to find the integer $d \in [1, n-1]$, such that $Q = dP$. In 2003 [1, p. 18] the fastest general algorithm known for

solving the ECDLP was Pollard's rho algorithm[9]. This algorithm has an expected running time of:

$$\frac{\sqrt{\pi n}}{2}$$

The largest ECDLP solved with Pollard's rho algorithm (2003) is for an elliptic curve over a 109-bit prime field. This corresponds to a decimal number of the size (33 digits):

`649037107316853453566312041152659`

Notice that this number is *alot* smaller than the 640 bit RSA prize number.

Where part 2 in the number field Sieve algorithm worked best when implemented on a single computer, Pollard's rho algorithm can easily be distributed onto several computers, as the processors don't have to communicate a lot with each other. This gives a clear advantage for Pollard's rho algorithm.

The table below, shows the computing power required to solve the discrete logarithm using the Pollard rho-method:

size of n (in bits)	MIPS years
160	$9.6 \cdot 10^{11}$
186	$7.9 \cdot 10^{15}$
234	$1.6 \cdot 10^{23}$
354	$1.5 \cdot 10^{41}$
426	$1.0 \cdot 10^{52}$

From this table and the table on page 7, it is pretty obvious why a 160 bit elliptic curve system is believed to provide the same level of security as a 1024 bit RSA system.

Over the past decade, the ECDLP has received considerable attention from mathematicians around the world, and so far, no significant weaknesses have been reported. This doesn't imply that there are no weaknesses though — just think of how long it took before someone proved Fermat's last theorem

(Fermat proposed his marvelous proof in 1637 and Wiles finally proved the theorem in 1994).

The discrete logarithm problem also shows up in the RSA system in the special case where someone would be in possession of a part of both the plain- and cipher text, i.e. knowing both m and c for a number of text blocks. In this case d is the only variable in the equation $m = c^d \bmod n$, and hence d can be found solving the discrete logarithm problem.

2.4 The future

The security of the different cryptosystems is based on the fact that breaking the systems takes so long time with the algorithms known today, that it is a near impossible task. Looking at the development of the algorithms over the last years, we probably shouldn't expect to see an algorithm that will be able to break either of the systems in a very short time. But we know absolutely nothing about what the future can bring, so perhaps we will see an incredible algorithm next year, or perhaps already tomorrow. Maybe it's already there, but being kept secret by intelligence services — who knows?

We can expect the algorithms for factoring integers and solving the ECDLP to become faster and more sofisticated in the coming years. The improvements seems to roughly follow Moore's law [5], which means that we can expect a doubling in speed every 18 months. If the development continues in the same way it's been doing for the last decades, then neither RSA or cryptosystems based on the ECDLP will be threatened in the next many decades. But with some informations that needs to be kept secure for many decades, like very sensitive informations about some person kept by the CIA or similar, very strong encryption is obviously needed. But for standard encryption that just has to last for a few years or less, we just have to remember to raise the security along with the release of faster computers. In order for an algorithm to be a serious threat, it will have to be of polynomial-time, and the algorithms we've seen so far have all been of some kind of exponential time. We can't easily tell if one system is more secure than the other, as both RSA and the elliptic curve variant of the El-Gamal system takes extremely long time to break, but as elliptic curve cryptosystems seems to give same security with much smaller keys this naturally leads to more possible

implementations, e.g. in small mobile devices where the computer power is limited. Funnily enough elliptic curves has proved to be today's best attack on factoring large numbers, so not only does elliptic curves seem to be able to give better security cryptosystems on itself, but it also makes the security of systems like RSA weaker as they now have to use larger keys in order to be secure.

Although both factoring large integers, and solving the ECDLP appears to be very difficult on classic computers, both problems are known to be easy to solve on a so called quantum computer. A quantum computer uses principles from quantum mechanics, and according to the laws of physics, a quantum computer is possible to make, but we probably won't see sophisticated quantum computers in the near future. In 1994, Schor [10] presented a polynomial-time algorithm for computing discrete logarithms and factoring integers using a quantum computer. This has later been extended to solving the ECDLP. An interesting thing is that it seems to be alot easier to solve the discrete log problem than factoring an integer using a quantum computer.[1, p. 196]

References

[1] Darrel Hankerson, Alfred Menezes, and Scot Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[2] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, 2nd edition, 1994.

[3] A. Lenstra and H. Lenstra, editors. *The Development of the Number Field Sieve*. Springer-Verlag, 1993.

[4] RSA Security - The RSA Challenge Numbers. http://www.rsasecurity.com/rsalabs/node.asp?id=2093.

[5] RSA Security - TWIRL and RSA Key Size. http://www.rsasecurity.com/rsalabs/node.asp?id=2004.

[6] A. Odlyzko. The future of integer factorization. *CryptoBytes - The technical newsletter of RSA Laboratories*, 1(2):5–12, Summer 1995.

[7] http://www.sisoftware.net/.

[8] Dan Boneh and Glenn Durfee:. New Results on the Cryptanalysis of Low Exponent RSA.

[9] J. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.

[10] P. W. Schor. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, 1994.