RSA: Is it time to move on? A dive into Cryptographic Protocols and their Weaknesses

Andrej Ljubić

November 2021

Abstract

Cryptography involves the development of secure communications and is absolutely fundamental to the world we live in. Every time we connect to a website, send an email or pay using a credit card we are placing our trust in the mathematics underpinning one of humanity's most important inventions.

Whitfield Diffie and Martin Hellman's 1976 paper *New Directions in Cryptography* began "We stand today on the brink of a revolution in cryptography" [1] - a claim proven unquestionably true as the consequences of their work prove more far-reaching than any could have dared to dream at the time. In the 45 years following their paper, mathematicians have developed several cryptographic protocols that revolutionised the landscape. One such cryptographic system is the RSA public-key cryptosystem, perhaps the most famous of all. RSA is still widely used today for everything from Bluetooth to e-banking, but in the last 45 years mathematicians have broken implementations time and time again.

Dan Boneh's paper *Twenty Years of Attacks on the RSA Cryptosystem* shed light on the major implementation flaws that could exist in this ubiquitous protocol [2]. Other attacks such as the ROCA vulnerability of 2017, which led to Estonia suspending 760,000 national ID cards [3], further revealed how difficult it was to safely implement RSA. In the light of these numerous vulnerabilities, and the comparative lack of weaknesses in alternative public-key cryptographic systems, we ask ourselves the question: *Is it time to let RSA go?*

Contents

1 A Brief History of Cryptography

One of the earliest known forms of cryptography is the **Caesar Cipher**, supposedly used by Julius Caesar in his private messages. The aim of the cipher is simple - to be unreadable to anyone except the intended recipient. First, place two lists of all alphabet characters below one another. Now we *shift* the bottom list by a certain factor. This means we move the letters to the left, with any characters moved past the right end being looped back to the right. Let's say we pick a shift of 7 (which we can call the *key*):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

Now we can choose a message, let's say **ATTACK**. What we would do is replace each character in **ATTACK** with the corresponding letter in the bottom list. For example, **A** would be replaced by **H**. If we continue through the word **ATTACK**, the *encrypted* version of the text is **HAAHJR**.

But the Caesar Cipher has several weaknesses. The first is, logically, that there are only a total of 25 *unique ciphertexts* (encrypted versions of the *plaintext*, the original message) we can possibly generate as, after we pass a shift of 25, we essentially just start again. This makes it susceptible to a **brute-force attack**, where we compute all possible shifts from 0 to 25 and analyse them to find the original message. This attack is somewhat mitigated by other ciphers that followed from Caesar (for example any **homophonic substitution cipher**, where each letter is replaced by another letter, but the replacements are in no specific order, e.g. **A** is replaced by **E** and **K** is replaced by **B**, because there are far more possibilities). The more pressing issue is the fact that this is a **symmetric** encryption scheme.

Symmetric cryptographic systems utilise the same *key* (in this case the shift) for both **encryption** and **decryption**. This means that someone who can encrypt the message can also decrypt it. The downside, however, is that for both parties to be able to communicate, they need to have **the same key** - and this leads to the problem of **key distribution**. How does Alice, if she wants to send a message, communicate her key to Bob? Her messages are sent over a public channel - sending the key along the same channel completely compromises the security of the communication, defeating the purpose. The only way for Alice and Bob to agree upon a private key is to use a private channel for communication - meeting in person to exchange the key, perhaps. Unsurprisingly, in the context of the world wide web, this approach is impractical. Nowadays, the symmetric cipher of choice is AES, the *Advanced Encryption Standard*, a cipher which has withstood years of attacks of all kinds.

2 The Fundamentals of Algebra and Number Theory

We say a is divisible by b if there exists an integer k such that $kb = a$. We denote this as $b \mid a$, and $b \nmid a$ if b does not divide a.

The Greatest Common Divisor of a and b is the largest integer k such that $k \mid a$ and $k \mid b$ and is denoted $gcd(a, b)$.

2.1 The Euclidean Algorithm

The Euclidean Algorithm gives us a way to efficiently calculate the greatest common divisor of two numbers a and b by forming a series of equations linking the two [4]. To start with, we write a as a product of b and a remainder added on:

$$a = b \cdot q + r$$

Where $q, r \in \mathbb{Z}, r < b$. Essentially, b divides into a q times with a remainder of r - and this unlocks a neat trick which makes this approach work.

Let's say $g = gcd(a, b)$, then we can rewrite a, b as integer multiples of g, i.e. $a = k_1 g, b = k_2 g$:

$$k_1 g = k_2 g + r$$

But now we can rewrite this equation for r:

$$r = k_1 g - k_2 g = (k_1 - k_2)g$$

And we can notice that $g \mid r$. We realise that this means $gcd(a, b) = gcd(b, r)$! From here, we can do write the same equation, but this time in terms of b and r:

$$b = r_0 \cdot q_0 + r_1$$

We essentially repeat this process several times

$$\begin{aligned} a &= b \cdot q_0 + r_1 \\ b &= r_1 \cdot q_1 + r_2 \\ r_1 &= r_2 \cdot q_2 + r_3 \\ r_2 &= r_3 \cdot q_3 + r_4 \end{aligned}$$

But when do we stop? Well, when $r_n = 0$ we know that $r_{n-1} \mid r_{n-2}$ and we can take r_{n-1} as the GCD! An example can be found in Appendix A.1.

2.2 Extended Euclidean Algorithm

We can take the Euclidean Algorithm a step further and calculate, in addition to the GCD, $u, v \in \mathbb{Z}$ for a, b which sum to the GCD [5], i.e.

$$au + bv = gcd(a, b)$$

This extension of the algorithm is invaluable for calculating **modular inverses** of numbers and is based on using the Euclidean Algorithm to calculate the GCD then writing it in terms of other numbers, repeating the process for the smallest non-GCD number until we reach an equation with only the GCD and the two starting numbers. An example can be found in Appendix A.2.

2.3 Continued Fractions

Continued Fractions are a way of representing real numbers as a sequence of positive integers. Let's say we have a real number α_1. We can form a sequence from α_1 in this way:

$$\alpha_1 = a_1 + \frac{1}{\alpha_2} \qquad \text{where } a_1 = \lfloor x \rfloor$$

For example, let's say $\alpha_1 = 1.5$. We can say that

$$\alpha_1 = 1 + \frac{1}{2}, \text{with } \alpha_2 = 2$$

The trick here is that if $\alpha_2 \notin \mathbb{Z}$, we can continue this exact process with α_2 and keep the continued fraction going.

$$\alpha_1 = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}$$

2.3.1 Convergents and their Properties

The kth convergent of a continued fraction is the approximation of the fraction we gain by truncating the continued fraction and using only the first k terms of the sequence, for example the 2nd convergence of $\frac{17}{11}$ is $1 + \frac{1}{1} = \frac{2}{1} = 2$ while the 3rd would be $1 + \frac{1}{1+\frac{1}{1}} = 1 + \frac{1}{2} = \frac{3}{2}$.

One of the obvious applications of these convergents is as **rational approximations to irrational numbers**. The convergents also have a series of interesting properties:

- As a sequence, they have a limit
    - This limit is α_1, the real number you are attempting to approximate
- They are alternately greater than and less than α_1

3 An Introduction to Modular Arithmetic

Modular Arithmetic is an incredibly important aspect of practically all asymmetric cryptography. A common way to refer to it is as clock arithmetic - imagine it's eleven o'clock right now. Three hours later it'll be two o'clock, right? Well we do the same thing in modular arithmetic:

$$11 + 3 = 14 \equiv 2 \mod 12$$

Essentially we take the sum of 11 and 3 and then divide by 12 and keep the remainder. Here, the 12 is called the **modulus**. The triple equals $\equiv$ denotes a congruence, and we say 14 is congruent to $2 \mod 12$.

We can think of this another way and say that two numbers a and b are congruent modulo m if their difference $a - b$ is divisible by m. For example, 26 and 2 are congruent modulo 12 because $26 - 2 = 24$ which is divisible by 12.

3.1 Modular (Multiplicative) Inverses

In normal maths, every number has an inverse - another number that, when multiplied with them, equals 1. In maths terms, we would say that for every number a there is another b where $ab = 1$. Generally, we can say that $b = \frac{1}{a}$ as that fits the equation - the inverse of 7, for example, is $\frac{1}{7}$.

In modular arithmetic, there are no fractions. Instead of finding b such that $ab = 1$, we instead find b such that $ab \equiv 1 \mod m$. This means that for different values of m there are different inverses for a. For example, the inverse of $3 \mod 5$ is 2, because $3 \cdot 2 \equiv 1 \mod 5$. This inverse is called the *modular multiplicative inverse.* We can therefore say that $3^{-1} \equiv 2 \mod 5$.

Yet not every number a has an inverse modulo m - for example, 2 has no inverse modulo 6. Why? This is because $gcd(2, 6) \neq 1$. Let's prove this in the general case and see why that is.

Lemma 3.1. *Given $a \mod m$, the modular multiplicative inverse b of a exists if and only if $gcd(a, m) = 1$.*

Proof. Assume there is b such that $ab \equiv 1 \mod m$. This would mean that there also exists c such that $ab = cm + 1$ and therefore $ab - cm = 1$. It follows therefore that both sides must be divisible by $gcd(a, m)$, meaning $gcd(a, m) = 1$ as RHS is 1.

As a result, if there is an inverse of a modulo m, such an inverse can only exist if $gcd(a, m) = 1$. □

3.2 The Chinese Remainder Theorem

The Chinese Remainder Theorem gives a unique solution to simultaneous linear congruences with coprime moduli [6]. Gives a variable x and a set of remainder and moduli, the theorem calculates the lowest possible value of x.

For example, given that:

$$\begin{aligned} x &\equiv 1 \mod 3 \\ x &\equiv 4 \mod 5 \\ x &\equiv 3 \mod 7 \end{aligned}$$

In this set of congruences, the lowest possible value of x is $x = 94$, which you can see fits all the congruences. There exists a simple algorithm for solving a small system of congruences by hand, which can be found in Appendix A.3.

3.3 Rings

In modular arithmetic we work modulo some modulus m, and we can think of the numbers we are able to use this way as a group of numbers:

$$\mathbb{Z}/m\mathbb{Z} = \{\, 0, 1, ..., m - 1 \,\}$$

This ring $\mathbb{Z}/m\mathbb{Z}$ is the *ring of integers modulo m.* We are able to add and multiply elements of this ring, then divide by m and take the remainder to obtain an element in $\mathbb{Z}/m\mathbb{Z}$. There are a set of axioms that define a ring, which can be found in Appendix A.4.

Units of a Ring

As we discussed, $a \in \mathbb{Z}/m\mathbb{Z}$ has a modular multiplicative inverse if $gcd(a, m) = 1$. The set of all numbers with modular inverses are denoted as $(\mathbb{Z}/m\mathbb{Z})^\star$ - this is called the *group of units modulo m.* Numbers which have inverses are called *units.* For example:

$$(\mathbb{Z}/12\mathbb{Z})^\star = \{ 1, 5, 7, 11 \}$$

You can think of the group of units modulo m as essentially a set containing all numbers less than m which are **coprime** with it.

3.4 Fields

If every number in $\mathbb{Z}/m\mathbb{Z}$ has a modular inverse, it is promoted from a ring to a field (a field is a ring in which division is possible) and can be denoted $\mathbb{F}_m$. Note that the only values of m where this are possible are values which are prime, which is why these fields are usually denoted $\mathbb{F}_p$. A finite field (also called a **Galois field**) is a field with a finite number of elements, such as those used in modular arithmetic. As a result, the term finite field will come up quite often to describe $\mathbb{F}_p$.

3.4.1 Euler's Totient Function

Euler's totient function returns the number of elements in the group of units modulo m [7]. Mathematically, this means

$$\phi(m) = \#(\mathbb{Z}/m\mathbb{Z})^\star = \# \{ 0 \leq a < m : gcd(a, m) = 1 \}$$

This function has a huge array of applications and further rules which allow us to do some pretty awesome things, and is a key piece of the RSA cryptosystem.

3.4.2 Computing the Euler Totient

If we say that the prime factorisation of n is $p_1^{e_1}, p_2^{e_2}, ..., p_k^{e_k}$ then

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

Note that if p is prime, $\phi(p) = p - 1$.

3.4.3 Euler's Formula

Euler's Formula states that if $gcd(a, p)$ then

$$a^{\phi(p)} \equiv 1 \mod p$$

This is perhaps the most important formula for the RSA cryptosystem, which we will get to soon. Note that this actually tells us a little more in the case modulo a prime , a case named after Fermat.

3.4.4 Fermat's Little Theorem

Since $\phi(p) = p - 1$, Fermat's Little Theorem states that [8]:

$$a^{p-1} \equiv 1 \mod p$$

This was stated by Fermat over 100 years before Euler's Formula. You may note that this also gives us a quick way to compute the modular multiplicative inverse of a in $\mathbb{F}_p$:

$$a^{p-1} \equiv 1 \mod p$$
$$a^{p-1} \cdot a^{-1} \equiv 1 \cdot a^{-1} \mod p$$
$$a^{p-2} \equiv a^{-1} \mod p$$

4 RSA - An Introduction

RSA is the most widely-used asymmetric cryptographic system in use, and the mathematics behind it are deceptively simple.

4.1 Encryption

Alice first generates primes p and q and N such that $N = pq$. This is half of the public key, with the other half being the **public exponent** e, which should be **coprime** to $\phi(N)$ (meaning $gcd(\phi(N), e) = 1$). These two values are then made public.

If Bob wants to send a message to Alice using RSA, he calculates:

$$c \equiv M^e \mod Nr$$

Where c is now the ciphertext and is sent to Alice.

4.2 Decryption

Alice calculates $\phi(N) = (p-1)(q-1)$ and then calculates the **modular multiplicative inverse** d of $e \mod N$. d is the **private key** used for the decryption which is known only to Alice.

To decrypt, the Alice calculates again:

$$M = c^d \mod N$$

The proof for this is based on **Euler's Formula**. Because we calculated d such that $de \equiv 1 mod \phi(N)$, then we know that there is an integer k such that . Also note that $c^d \equiv (M^e)^d \equiv M^{de}$:

$$M^{de} \equiv M^{k\phi(N)+1} \equiv M^{k\phi(N)} \times M \equiv (M^{\phi(N)})^k \times M \equiv 1^k \times M \equiv M \mod N$$

Meaning we get back to the original plaintext M.

5 RSA - Attacks

RSA has been extensively studied since it was first published in 1977, and since then countless of ingenious attacks have been developed that compromise its security. There are a host of ways in which the implementation of the algorithm can fail, from choosing a public exponent which is too small, to one that is to big, to making p and q too close numerically. These implementation flaws constantly crop up, even in large libraries used in the most important systems in the world. Even OpenSSL, the cryptographic suite of tools used by the majority of websites and developed by professionals, has been found to have flaws within the last few years. In this section, we will go over different types of attacks, and outline examples to show how RSA can be made insecure with such few implementation mistakes.

5.1 Small Public Exponent - Håstad's Broadcast Attack

Commonly, a user may want to send the same message to multiple people. In this scenario, the public exponent e stays the same (it is often standardised) but the public key N may vary.

Håstad's Broadcast Attack can be used when the same message m is sent to at least e people with the public modulus $N_1, N_2, ..., N_e$. For example, if we say $e = 3$, then we have the following three congruences:

$$\begin{aligned} m^e &\equiv c_1 \mod N_1 \\ m^e &\equiv c_2 \mod N_2 \\ m^e &\equiv c_3 \mod N_3 \end{aligned}$$

The **Chinese Remainder Theorem** is a technique used for solving a set of simultaneous linear congruences where the moduli are **coprime** - in this example, it's a way of finding $m^e \mod N_1N_2N_3$. However, note that m is less than the smallest N_x we have, therefore if we have e public moduli then $m^e < N_1N_2...N_e$. This means we can simply take the regular eth root of m^e rather than the modular root as it is impossible for m^e it to be greater than the modulus.

An example can be found in Appendix A.5.

5.2 Large Public Exponent - Wiener's Attack

Conversely, if e is too large, this also creates a vulnerability as its inverse mod $\phi(N)$ d is necessarily small. If d is sufficiently small ($d < \frac{1}{3}N^{\frac{1}{4}}$) then we can retrieve the private key using the theory of continued fractions [2]. Wiener's Attack utilises the convergents of the continued fraction expansion of $\frac{k}{d}$ to attempt to guess the decryption exponent.

We can say that

$$\begin{aligned} \phi(N) &= (p-1)(q-1) \\ &= pq - (p+q) + 1 \\ &\approx N \end{aligned}$$

We can also say that since $ed \equiv 1 \mod \phi(N)$, we can rearrange this to say that $ed = k\phi(N) + 1$:

$$\begin{aligned} ed &= k\phi(N) + 1 \\ ed - k\phi(N) &= 1 \\ \frac{e}{\phi(N)} - \frac{k}{d} &= \frac{1}{d\phi(N)} && \text{divide by } d\phi(N) \end{aligned}$$

Now since $d\phi(N)$ is likely to be huge, we can say it's almost zero, and also use the approximation from before to say that

$$\frac{e}{N} \approx \frac{k}{d}$$

Note that $\frac{e}{N}$ is comprised of entirely **public information**, and gives us a good approximation of $\frac{k}{d}$. which is entirely **private information**.

We can represent $\frac{e}{N}$ as a continued fraction. If we go through the convergents of this fraction, we may come across $\frac{k}{d}$ since $\frac{e}{N} \approx \frac{k}{d}$. This is more likely to work when d is smaller due to **Legendre's Theorem in Diophantine Approximations**, specifically $d < \frac{1}{3}N^{\frac{1}{4}}$ [2].

Once we list the convergents, we iterate through and there are a few checks we can make to determine whether or not it's the correct convergent:

1. As $ed \equiv 1 \mod \phi(N)$ and $\phi(N)$ is even, d must be odd, so we can discard convergences with even denominators.
2. Since $\phi(N)$ must be a whole number, we can compute $\frac{ed-1}{k}$ and see if it returns an integer or not - if not, we can discard the convergent.

Once we find a convergent we don't discard, we can assume it's $\frac{k}{d}$ and see if the resultant private key yields a valid result or not. This can take a lot of decryption attempts to work successfully however - we can speed up the "checking" process using a quadratic equation to give us p and q out of the box, which is more efficient.

If we say that $N = pq$, it follows that:

$$\begin{aligned}\phi(N) &= (p-1)(q-1)\\ \phi(N) &= N - (p+q) + 1\\ p+q &= N - \phi(N) + 1\end{aligned}$$

If we now consider the quadratic equation $(x-p)(x-q) = 0$ with roots p and q being the prime factors of N, we can expand this and substitute:

$$\begin{aligned}(x-p)(x-q) &= 0\\ x^2 - (p+q)x + N &= 0\\ x^2 - (N - \phi(N) + 1)x + N &= 0\end{aligned}$$

If our value of $\phi(N)$ is correct, we can substitute this into the equation and solve it for two integer values p and q. If the values are not integer, the result can be discarded. If they are, we can use d to decrypt the data from before.

5.3 Prime Number Choice - p and q numerically close

If p and q are numerically close, we can use an approach called **Fermat Factorisation** to factorise N. During Fermat Factorisation, we hope to find a and b such that

$$a^2 - b^2 = N$$

Because we can then factorise the left-hand expression into

$$(a+b)(a-b) = N$$

And thus get p and q as $a+b$ and $a-b$. The reason we use this when p and q are numerically close is because the closer they are to each other the closer they are to $\sqrt{N}$. If we say $a = \sqrt{N}$ rounded up to the nearest number, we can calculate $b^2 = a^2 - N$ (as rearranged from before) until b is a whole number, after which we've solved the equation. If p and q are not close, we would require far too many iterations of a and b for this to be feasible.

6 The Diffie-Hellman Key Exchange

We will now introduce cryptographic protocols that are alternatives to the RSA cryptosystem. Unlike RSA, where you can send messages of your choice, the DHKE is used to generate a secret number shared between Alice and Bob. This shared secret is then used as the key for a symmetric cryptosystem like the Advanced Encryption Standard (AES), the foremost symmetric cryptosystem [9].

6.1 The Key Exchange

A large prime p and a generator $g \in \mathbb{F}_p, g \neq 0$ are made public. Alice and Bob choose their secret integers a and b respectively. They then compute the following:

$$\begin{aligned} A &= g^a \mod p \\ B &= g^b \mod p \end{aligned}$$

These numbers A and B are exchanged between Alice and Bob over a public channel. Once the other person's number is received, they then put it to the power of their secret integer, i.e. Alice computes B^a and Bob computes A^b, both modulo p. Note that:

$$B^a = (g^b)^a = g^{ab} = (g^a)^b = A^b$$

This means that once they do this, they are in possession of the same number, which they can then use as a shared secret.

6.2 The Discrete Logarithm Problem

The safety of the Diffie-Hellman Key Exchange is grounded on the difficulty of solving the discrete logarithm problem - the difficulty of computing x given

$$g^x \equiv a \mod p$$

You can see this in the overview of the key exchange, presented above - the values $g^a \mod p$ and $g^b \mod p$ are sent over a public channel, but because we cannot solve the DLP efficiently an attacker is unable to retrieve a or b. We say that the DLP is DHKE's **trapdoor function**. As a result, many attacks on Diffie-Hellman rely on situations in which you can efficiently compute the discrete logarithm, the most notable of which is when in the finite field $\mathbb{F}_p$ and $p-1$ is a **smooth number**, meaning it has a large number of small factors. The **Pohlig-Hellman algorithm** efficiently solves the DLP in when such a prime is used.

6.3 The Pohlig-Hellman Algorithm

Recall from Euler's Theorem that $a^{\phi(N)} \equiv 1 \mod N$. Let $\phi(N) = pq$, where $GCD(p,q) = 1$ (but they are not necessarily prime). We can now attempt to solve the discrete logarithm problem $a^x \equiv b \mod N$:

Let $x = a_0 + a_1 p$. Then:

$$\begin{aligned} a^x &\equiv b \mod N \\ a^{qx} &\equiv b^q \\ a^{a_0 q + a_1 pq} &\equiv b^q \\ a^{a_0 q} \cdot a^{a_1 pq} &\equiv b^q \\ (a^{a_0})^q \cdot (a^{pq})^{a_1} &\equiv b^q \\ (a^{a_0})^q \cdot 1^{a_1} &\equiv b^q && \text{Euler's Theorem} \\ (a^q)^{a_0} &\equiv b^q \end{aligned}$$

Since a^q and b^q are known quantities that can be easily calculated, we can determine a_0 through trial and error. We could have done this originally for x, but note that as $x = a_0 + a_1 p$, $a_0 < p$ and therefore there are far fewer possibilities to try for a_0 as we only have to go up to p as opposed to N if we had tried with x.

Once we find a_0, since $x = a_0 + a_1 p$ we know that $x \equiv a_0 \mod p$, giving us a congruence. Repeating this the other way by putting both sides to the power of p will yield us a second congruence modulo q, and we can combine the two and solve for x using the Chinese Remainder Theorem.

The "smoother" the prime p is (the smaller the factors of $p-1$ are) the more efficient the Pohlig-Hellman algorithm is for solving the Discrete Logarithm Problem, as the smaller the prime factors of $\phi(N)$ the faster the calculations can be made.

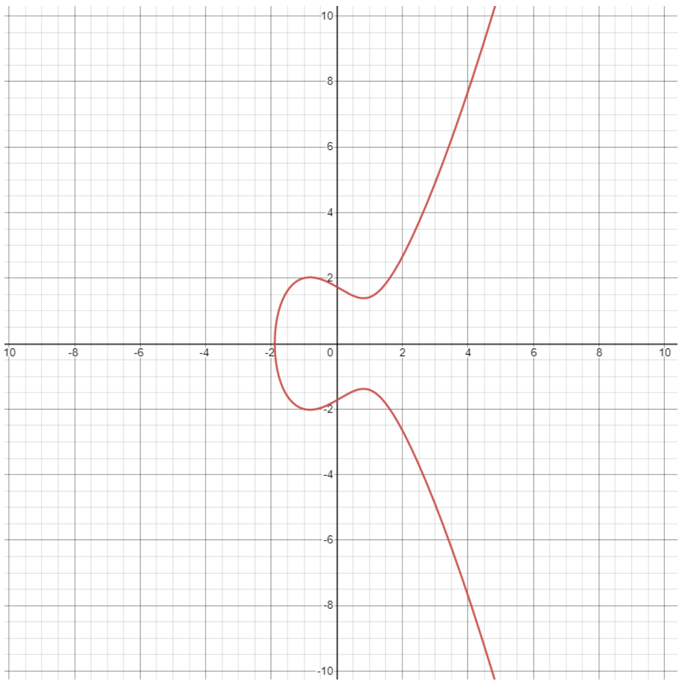
This implementation of the Pohlig-Hellman algorithm works perfectly for modular arithmetic, but there is in fact a **generic** Pohlig-Hellman algorithm that works for *any group*, and we will see this later. In the generic algorithm, it is the **order of the group** (the number of elements in the group) that needs to be smooth, and this is also true in this implementation - when working modulo p, we are working in the field $\mathbb{F}_p$, which has $p-1$ elements (all numbers $1, 2, \ldots, p-1$; 0 is not included in $(\mathbb{Z}/p\mathbb{Z})^*$) and therefore order $p-1$.

7 Elliptic Curve Cryptography

Another alternative cryptographic system involves the use of elliptic curves for the trapdoor function. Confusingly, elliptic curves actually have nothing to do with the concept of an **ellipse** - they are completely distinct, but unfortunately named. Elliptic Curves are, quite simply, a special type of curve. Here we'll be talking about curves in **Weierstrass Form**, which looks like the following:

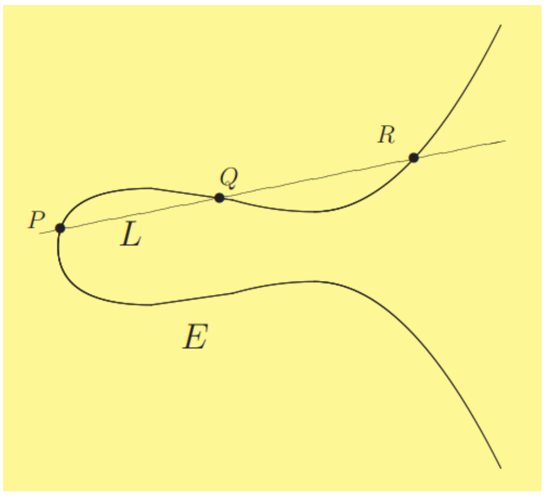
$$y^2 = x^3 + ax + b$$

Where a and b are integers. A good example is the following, from the equation $y^2 = x^3 - 2x + 3$:

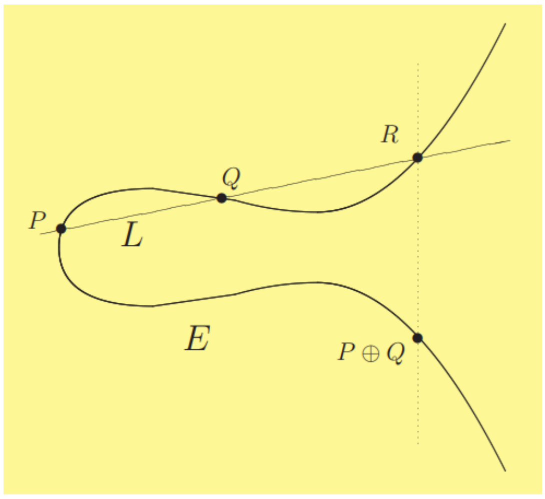


The **rational points** on an elliptic curve are all the points on the curve with rational x and y values. The beauty of these curves is that the rational points form a **group**, meaning they are all linked by an operation. If we choose any two rational points, drawing a line between them intersects the curve at a third point R. If we reflect R in the x-axis, we get the point R' and we say this point R' is the result of the **point addition** of P and Q. This is the operation of the group.

We can also **double** a point, meaning we compute $P + P$. Remember that in this context (and from now on) the $+$ symbol is not regular addition but our new operation **point addition**, which given two points on the curve returns a third. How we do that may not be immediately obvious, for surely there are an infinite number of lines passing through P? What we do in this case is take the **tangent to the curve**. The tangent is a line which only touches the curve at one point - you can think of it as being exactly parallel to the specific point you are drawing it at. Once we find the point of intersection of the curve, we again reflect it in the x-axis to calculate $2P$.

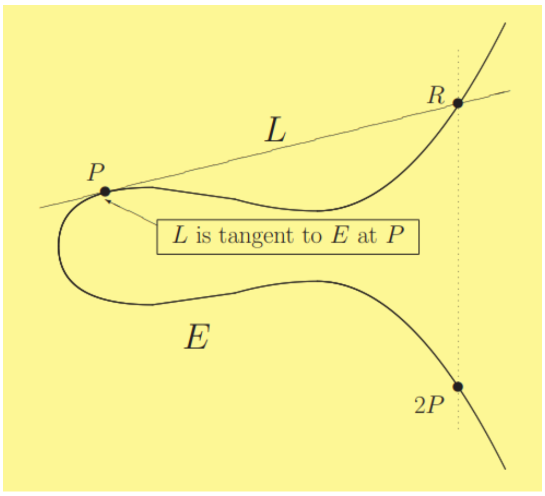


(a) Finding the third point R



(b) Finding $P + Q$

Figure 1: Images courtesy of Joseph Silverman's 2006 presentation slides



(a) Finding $P + P = 2P$

Figure 2: Image courtesy of Joseph Silverman's 2006 presentation slides

Calculating the tangent involves some **implicit differentiation**:

$$\begin{aligned} y^2 &= x^3 - 2x + 3 \\ 2y\frac{dy}{dx} &= 3x^2 - 2 \\ \frac{dy}{dx} &= \frac{3x^2 - 2}{2y} \end{aligned}$$

And substituting the point $P = (x, y)$ yields the gradient at the point P.

If we attempt to calculate $P + (-P)$, i.e. the point on the curve with the same x-coordinate but the negative y-coordinate, we have a slight problem - clearly, if we go through both points, there is no third intersection with the curve, and point addition fails. We say the result of this addition is $\mathcal{O}$, the **point at infinity**. If you want to add this point to any point X, we say that $X + \mathcal{O} = X$ and as such $\mathcal{O}$ is the identity element of the group.

It is noticeable that the numbers could get **very** big **very** quickly, and square rooting these huge numbers would be extremely resource-intensive. Instead, when we use ECC practically, we use modular arithmetic and work over $\mathbb{F}_p$ and reduce the x and y coordinates of the points modulo p. The operation of point addition and the structure of the group still holds even for elliptic curves over a finite field, which we denote $E(\mathbb{F}_p)$.

7.1 The Key Exchange Protocol

If Alice and Bob wish to generate a shared secret, they first choose an elliptic curve $E(\mathbb{F}_p)$. The curve can be made public, but this doesn't compromise the security of the exchange. They next choose a rational point $G \in E(\mathbb{F}_p)$, which we call the **generator** point. This point is also made public.

Now Alice and Bob generate a private number each, and we'll call these numbers a and b respectively. Alice then calculates the rational point A, which is done by adding the generator G to itself a times, i.e. $A = aG$. Bob does the same with *his* private number to calculate $B = bG$. Remember that even though we write bG as if it's multiplication, it's actually just b series of **point** additions (not regular addition!) of G to itself! The multiplication notation is just a handy shortcut. Once this is done, Alice sends A to Bob and Bob sends B back.

$$\begin{aligned} A &= aG && \text{Alice} \\ B &= bG && \text{Bob} \end{aligned}$$

Let's say that $a = 1000$ and $b = 800$. Alice will receive $B = 800G$ from Bob while Bob receives $A = 1000G$ from Alice. The trick here is that Alice can now add B to G another a (1000) times, that is calculate $B + G + G + ... + G$ until she calculates $S_a = B + 1000G = 800G + 1000G = 1800G$, the equivalent of performing a **point addition** of G to itself a total of 1800 times. Bob, in turn, adds A to G another b (800) times. Note that this means Bob computes $S_b = A + 800G = 1000G + 800G = 1800G$. But that means $S_a = S_b$, as both are the result of adding G to itself a total of 1800 times!

The incredible thing here is that the value S is a **shared secret**, a value known only to Alice and Bob. The x-coordinate of S can be used as the private key for a symmetric protocol, with which Alice and Bob can communicate efficiently in secret.

It is also noteworthy that if the operations were not done over $\mathbb{F}_p$, points added to themselves more times would have more extreme x and y values, meaning that by doing the arithmetic modulo p we eliminate the possibility of analysing the point values.

7.2 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

The ECDLP essentially comes down to:
Given $G, A \in E(\mathbb{F}_p)$ and $A = aG$, is it possible to retrieve a?

This problem is deceptively hard, and requires far more than a simple rearrangement - remember that it is not basic multiplication, but rather point addition that is the operation in question here. You may also notice the phrase *Discrete Logarithm Problem* again, which is the trapdoor function of the Diffie-Hellman Key Exchange. In fact, this is essentially the same problem, just with a different operation - point addition rather than multiplication. As such, the problem is really **abstracted away into the group**, and the generic Pohlig-Hellman algorithm can once again be used on the curve $E(\mathbb{F}_p)$ provided that the **order of the curve is smooth**.

8 A Comparison

8.1 Implementation Security

As can be seen in previous sections, as well as more completely in Boneh's *Twenty Years of Attacks on the RSA Cryptosystem*, the RSA cryptographic protocol is vulnerable to a wide array of attacks [2]. This makes RSA fiendishly complex to implement correctly, and even major libraries such as OpenSSL (the most widely-used cryptographic library) can fail to do it correctly [10]. In 1998, Daniel Bleichenbacher discovered a vulnerability in the way RSA was implemented in SSL (the predecessor of TLS, the protocol responsible for securing HTTPS communications throughout the entirety of the internet) by crafting around a million messages and analysing error codes. This attack has been refined in recent years, and websites such as Facebook were shown to be vulnerable to a variant of Bleichenbacher's Attack (called the ROBOT attack) as recently as 2017 [11].

By contrast, protocols such as the Diffie-Hellman Key Exchange and Elliptic Curve Cryptography are vulnerable to far fewer such attacks, and with the notable exception of the Pohlig-Hellman Algorithm (which, as stated before, efficiently solves the discrete logarithm when the order of the group is smooth) the most dangerous such attacks are man-in-the-middle attacks such as the Logjam Attack, where an attacker positions themselves to intercept communications between the two parties in order to downgrade the security of the Diffie-Hellman Key Exchange [12]. Mathematical attacks on the protocols themselves are fairly trivial to prevent - in the case of Pohlig-Hellman, one simply makes the public prime p a **safe prime**, a prime number in the form $p = 2q + 1$ where q is also a prime number.

8.2 Relative Efficiency

One important statistic is the **bit security** of an algorithm - how many bits of data is required to produce how many bits of security. This number represents the order of magnitude of the amount of resources needed to break a crypto primitives' security [13]. If an algorithm has n-bit security, that means we believe it would take 2^n operations to break the security of the protocol.

In this regard, RSA severely lags behind alternatives such as ECC - 1024-bit RSA keys give 80 bits of security [14] while 2048-bit RSA keys give 112 bits of security [14]. ECC, by contrast, only require 160 or 192 bits of data (depending on the implementation) to provide the same level of security as RSA-1024 [14] [15]. Additionally, computational power is increasing rapidly, meaning the security of our algorithms will have to increase too. Increasing key sizes has a much greater effect on ECC security than RSA. In order to provide 128 bits of security, RSA requires a key size of 3072 bits; in order to provide 256 bits of security, RSA requires a key size of 15360 bits [14]. A doubling of bit security requires a key of 5 times the bits. ECC requires 256 bits for 128-bit security and 512 bits for 256-bit security [14] [16] - which is much more proportional and scalable, meaning an increase in security is far easier to achieve.

Distinct from the mathematical security of the algorithm, another consideration is the speed itself. ECC's use of smaller keys means less data is transferred between the server and the client during key exchanges such as the TLS handshake [16]. Additionally, ECC requires far less processing power and memory, resulting in faster operation speeds, which is obviously ideal for making operations such as connecting to a website securely as far as possible for the end user [16].

8.3 Perfect Forward Secrecy

The security of a set of communications is crucial, but another important factor is the security of **past communications**. It is not uncommon for attackers (including governments!) to store the data from key exchanges and the resulting communications for a period of time and then compromise it when the technology becomes powerful enough [11] - once computers become fast enough to break the system, we can decrypt messages from 5 years ago. This may not seem like a threat to the average user, but even 5 years down the line private information such as credit card details may be valid. Military and state secrets are *definitely* going to be valid. Ultimately, it is a risk few can afford to take. We say traditional RSA lacks **perfect forward secrecy**, as a single compromise at any point can decrypt all messages ever sent. Why? Well, if we do manage to factor N, we can calculate the private key d. This private key is valid for **all communications that ever took and will take place**.

The Signal protocol, which encrypts communications in everything from WhatsApp to Google's Allo messenger [17], uses the Diffie-Hellman Key Exchange with perfect forward secrecy. To achieve perfect forward secrecy, both sides of the protocol generate a **new key for every message they send**. This means that if one key gets broken, only a *single* message can be read using that information. In order to read important data, the keys would have to be broken again and again. Perfect forward secrecy doesn't suit RSA as the key generation is computationally expensive, while for protocols like the Diffie-Hellman Key Exchange and Elliptic Curve Cryptography the generation is incredibly simple. This lack of perfect forward secrecy led to TLS 1.3 removing support for RSA key exchanges, opting instead for algorithms that effectively support the feature [11] [18].

8.4 The Quantum Computing Era and the Effects on Cryptography

All the protocols we have analysed so far rely on **trapdoor functions** - functions that are easy to perform in one direction, but difficult to reverse. For RSA, the trapdoor function is multiplication of large primes; for DHKE, the DLP; for ECC, the ECDLP.

All three of these trapdoor functions are indeed difficult to reverse - for a classical computer. In 1954, Peter Shor proposed what is now known as *Shor's Algorithm*, an algorithm for quantum computers that enables them to factor integers and solve the DLP and the ECDLP in polynomial time [19]. While there are no quantum computers in existence that are capable of executing such algorithms at the scale required to compromise security, their very existence is a warning to the cryptography community that new protocols have to be found and accepted relatively soon. NIST (the National Institute of Standards and Technology), the body responsible for standardising and recommending cryptosystems, started a competition to find an accepted and quantum-safe cryptosystem. Four algorithms made it to Round 3 of the NIST standardisation process [20]. While these four protocols have yet to be broken, the Rainbow signature scheme (a finalist for the Digital Signature Algorithms category) was broken in February 2022, nearly five years after its submission [21]. This shows the rigour required to come to an accepted protocol - the world's leading mathematicians are submitting systems that pass a huge number of initial validation checks only to eventually be shown to be insecure.

8.5 Conclusion

In conclusion, RSA is a poor cryptosystem to use in the modern world. It it outshone in performance efficiency by protocols such as the Diffie-Hellman Key Exchange and Elliptic Curve Cryptography, while also lacking the benefits of Perfect Forward Secrecy and being fiendishly difficult to implement correctly. Protocols such as TLS have already recognised this and begun to phase out RSA in favour of better alternatives, which will ultimately be beneficial. All of these alternatives, however, will also have to be replaced in the very near future in preparation for the onslaught of quantum supremacy; cryptosystems such as NTRU will become the new standard. One way or another, the conclusion is clear: *it is time to let RSA go.*

A Appendix

A.1 Euclidean Algorithm

Let's say we want to find the GCD of 8075 and 16283. First, we can write it in the form $a = b \cdot q + r$:

$$16283 = 8075 \cdot 2 + 133$$

And now we attempt to calculate the GCD of 8075 and 133.

$$\begin{aligned}
8075 &= 133 \cdot 60 + 95 \\
133 &= 95 \cdot 1 + 38 \\
95 &= 38 \cdot 2 + 19 \\
38 &= 19 \cdot 2 + 0
\end{aligned}$$

Therefore the GCD of 16283 and 8075 is 19.

A.2 Extended Euclidean Algorithm

Let's work with the example from Appendix A.1, writing an equation for the GCD:

$$19 = 95 - 38 \cdot 2$$

Now 38 is the smallest non-GCD number, and we can write it in terms of the larger number in the sequence of equations written in Section 2.1.1, then repeat for the next smallest:

$$\begin{aligned}
19 &= 95 - 38 \cdot 2 \\
&= 95 - (133 - 95 \cdot 1) \cdot 2 \\
&= 95 - (133 \cdot 2 + 95 \cdot -2) \\
&= 95 \cdot 3 + 133 \cdot -2 \\
&= (8075 + 133 \cdot -60) \cdot 3 + 133 \cdot -2 \\
&= 8075 \cdot 3 + 133 \cdot -180 + 133 \cdot -2 \\
&= 8075 \cdot 3 + 133 \cdot -182 \\
&= 8075 \cdot 3 + (16283 + 8075 \cdot -2) \cdot -182 \\
&= 8075 \cdot 3 + 16283 \cdot -182 + 8075 \cdot 364 \\
&= 8075 \cdot 367 + 16283 \cdot -182
\end{aligned}$$

Therefore our equation $au + bv = gcd(a, b)$ is as follows:

$$16283 \cdot -182 + 8075 \cdot 367 = 19$$

A.3 Solving the System of Congruences

First, we begin with the largest modulus, $x \equiv 3 \mod 7$. We rewrite this as

$$x = 7j + 3 \tag{1}$$

Substitute this into the next congruence $x \equiv 4 \mod 5$:

$$\begin{align} x &\equiv 4 \mod 5 \tag{2}\\ 7j+3 &\equiv 4 \mod 5 \tag{3}\\ 7j &\equiv 1 \mod 5 \tag{4}\\ j &\equiv 3 \mod 5 \tag{5}\end{align}$$

Now we do the same thing for j and write it as

$$j = 5k + 3 \tag{6}$$

Substitute this back into (1), we get

$$x = 7(5k + 3) + 3 = 35k + 24 \tag{7}$$

Now we substitute *this* into the final congruence

$$\begin{align} x &\equiv 1 \mod 3 \tag{8}\\ 35k+24 &\equiv 1 \mod 3 \tag{9}\\ 2k &\equiv 1 \mod 3 \tag{10}\\ k &\equiv 2 \mod 3 \tag{11}\end{align}$$

Now we can write k as

$$k = 3l + 2 \tag{12}$$

And substitute it back into (6):

$$x = 35(3l + 2) + 24 = 105l + 94 \tag{13}$$

Note that 105 is divisible by all moduli, so the $105l$ will disappear in all congruences. This leaves 94 as the solution for x that fits all congruences. Note that that means $x \equiv 94 \mod 105$, so there are infinite possible solutions. The Chinese Remainder Theorem will always solve the congruences to the lowest common multiple of all moduli.

A.4 Ring Axioms

Certain rules define a ring R:

Firstly, R is an **abelian group** under addition, meaning that

1. R is closed under addition ($\forall a, b \in R, a + b \in R$)
2. There is an additive identity (0)
3. Every element has an additive inverse ($\forall a \in R, \exists -a \in R$)
4. Addition is associative ($\forall a, b, c \in R, (a + b) + c = a + (b + c)$)
5. Addition is commutative ($\forall a, b \in R, a + b = b + a$)

Secondly, R is closed under multiplication, which is associative.

Thirdly, Multiplication distributes over addition:

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

[22]

A.5 Håstad's Broadcast Attack

Let's say $e = 3$, and

$$\begin{aligned} m^3 &\equiv 63 \mod 131 \\ m^3 &\equiv 13 \mod 137 \\ m^3 &\equiv 83 \mod 191 \end{aligned}$$

We can use the Chinese Remainder Theorem to calculate that the solution for this system of congruences is $m^3 \equiv 1442897 \mod 3427877$. Since m^3 is guaranteed to be less than 3427877, we can say that $m^3 = 1442897$. We can then simply say that $m = \sqrt[3]{1442897} = 113$, retrieving the secret message m.

References

[1] Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE TRANSACTIONS ON INFORMATION THEORY* IT-22.6 (1976). URL: https://ee.stanford.edu/~hellman/publications/24.pdf.

[2] Dan Boneh. “Twenty Years of Attacks on the RSA Cryptosystem”. In: *NOTICES OF THE AMS* 46 (Feb. 2002). URL: https://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf.

[3] BBC, ed. *Security flaw forces Estonia ID 'lockdown'.* Nov. 2017. URL: https://www.bbc.co.uk/news/technology-41858583.

[4] *Euclidean Algorithm.* URL: https://brilliant.org/wiki/euclidean-algorithm/.

[5] *Extended Euclidean Algorithm.* URL: https://brilliant.org/wiki/extended-euclidean-algorithm/.

[6] *Chinese Remainder Theorem.* URL: https://brilliant.org/wiki/chinese-remainder-theorem/.

[7] *Euler's Totient Function.* URL: https://brilliant.org/wiki/eulers-totient-function/.

[8] *Fermat's Little Theorem.* URL: https://brilliant.org/wiki/fermats-little-theorem/.

[9] *Advanced Encryption Standard (AES).* Nov. 2001. URL: https://www.nist.gov/publications/advanced-encryption-standard-aes.

[10] *CVE-2018-0737.* URL: https://www.cvedetails.com/cve/CVE-2018-0737.

[11] Nick Sullivan. *A Detailed Look at RFC 8446 (a.k.a. TLS 1.3).* Nov. 2018. URL: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/.

[12] David Adrian et al. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *22nd ACM Conference on Computer and Communications Security* (Oct. 2015). DOI: 10.1145/2810103.2813707.

[13] Joël Alwen. *The Bit-Security of Cryptographic Primitives.* June 2017. URL: https://wickr.com/the-bit-security-of-cryptographic-primitives-2/.

[14] Kerry Maletsky. *RSA vs. ECC Comparison for Embedded Systems.* URL: http://ww1.microchip.com/downloads/en/DeviceDoc/00003442A.pdf.

[15] *Size considerations for public and private keys.* URL: https://www.ibm.com/docs/en/zos/2.2.0?topic=certificates-size-considerations-public-private-keys.

[16] Wayne Thayer. *Benefits of Elliptic Curve Cryptography.* URL: https://pkic.org/2014/06/10/benefits-of-elliptic-curve-cryptography/.

[17] Andy Greenberg. Nov. 2016. URL: https://www.wired.com/2016/11/what-is-perfect-forward-secrecy/.

[18] Patrick Crowley. Feb. 2018. URL: https://blogs.cisco.com/security/tls-1-3-and-forward-secrecy-count-us-in-and-heres-why.

[19] *Shor's algorithm.* URL: https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm.

[20] *Round 3 Submissions.* URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[21] Ward Beullens. In: (Feb. 2022). URL: https://eprint.iacr.org/2022/214.pdf.

[22] *Ring Theory.* URL: https://brilliant.org/wiki/ring-theory/.