Encryption on the Internet

Eirik Albrigtsen, 0630552

April 9, 2010

Contents

Chapter 1

Introduction

1.1 What is Cryptography?

Cryptography comes from the Greek words *kryptos*, meaning hidden, and *graphia*, meaning writing. It is the practice of concealing information and as an academic discipline build solidly upon mathematics and computer science. Encryption is the cryptographic operation dealing with the actual transformation of information, to render it hidden in some sense, whereas decryption is the reverse operation. Cryptography is an ancient art that has in recent years blossomed into a rigorous subject, withstanding both wide publicity and expert scrutiny. This runs counter to the historical account of cryptography as a practice based entirely on secrecy, for it did not use to be this way.

Dating all the way back to Julius Caesar, we find examples of cryptography in practice. We will discuss these methods in more detail later, but generally they are to cryptography today no more than pig-latin is to English. They remain as simple methods that are easy to remember, and rely entirely on the secrecy of the method.

Modern cryptography must obey the following well known principle [8] lest implementing programmers can go mad with power.

Definition 1.1. (Kerchoff's Principle) The security of a cryptosystem must not depend on keeping the cryptographic algorithms secret, the security must only depend the secrecy of the key.

Currently, this key allowing the decryption operation to be performed can take the form of an electronic unit supplied by a bank, or a remembered password. However, in most cases you will probably not have any idea of what it is. Our electronic lives are actually infused with cryptographic methods of such complexity, that they transcended the need for human interaction.

This statement may sound outlandish, and it is why we have chosen this topic. To demonstrate that it is indeed possible to operate such machinery with minimal interaction from the user, we will explore the mathematics behind some of the most commonly used cryptographic methods, as well as examining a promising candidate.

While cryptography may seem like a very well understood problem, there are in fact many pitfalls surrounding implementation. In particular, when the underlying device contains security flaws, or when the implementations feature seemingly innocuous error messages that can cleverly be

used to verify decryptions, i.e. act as a *decryption oracle*. For an interesting, but more advanced account on modern pitfalls in cryptography, we refer to the Nate Lawson's insightful presentation in [16]. An interesting analogy from this is worth noting: "*crypto is strong, but it is fragile. Think of it like carbon fiber; if you bend it the right way it snaps.*"

It is for these reasons that more sophisticated notions the security of cryptographic systems have recently been invented. If a system can withstand more sophisticated attacks, then ideally it should be harder to make errors in the implementation stage. Although the ideas of security proofs are relatively new, even for a young subject like computer aided cryptography, they have become widely popular. This is because they allow results to come about more easily in a simplified model, under the assumption that one or two problems are hard to solve. Clearly, the use of a simplified model has negative implications in this context. Essentially, they boil down to saying that a system is hard to invert, unless someone finds a way to do it, in which case it isn't. By *proving* something mathematically, we do not usually expect this kind of ambivalence.

It is therefore natural to follow the standard of stating the assumptions made about each system as simply and concisely as possible, in the interest to further their study, and to consequently either increase their believed veracity, or to reject them. Koblitz has an interesting historical account on the topic of security proofs here [1], and a more technical one here [17] for the interested reader.

Before we get started, it is important to note that cryptography is a vast subject. There is an enormous amount of papers being released on evermore complicated cryptographic schemes and methods, necessitated by a growing need for better security. Thus, in the interest of talking more solidly about certain parts, we will move quickly over other essential topics of cryptography, like message authenticity, and digital signatures. Our report will focus largely on the mathematics behind certain cryptographic systems, and methods for reliably generating large prime numbers for these systems. These methods will later be rigorously compared in terms of efficiency.

1.2 Efficiency

1.2.1 A fundamental limitation

In the early days of cryptography, developers of encryption methods were concerned with the notion of *perfect security*. This is system where, even if an encrypted message was intercepted, one would learn nothing from the it, and no method could exist to obtain information about the original message - without the secret key.

Today, we know an example of such a system; the One-Time Pad. This works by converting a message into a stream of bits, combining it with a key of equal length, and XORing them together. XOR is short for the logical exclusive OR operator. It is is performed bitwise (i.e. match up the ith bit of your message with the ith bit of the key and XOR these), and is commonly denoted by the additive $\oplus$ symbol, as it is mathematically equivalent to addition modulo 2. For decryption, simply XORing the encrypted message with the key again, will reveal the message. This method of encryption is has the *one-time* prefix because it can be used securely only once with the same password. To see this, consider two messages m_i and corresponding encrypted text $c_i = m_i \oplus k$ for some key k. By taking $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$ we can subtract the key from the combined message. For such a bitwise sum of two messages, there are bound to be statistical patterns from the overlying language of the messages. These can be exploited to reveal

information about the original messages.

Unfortunately, this is not its most significant problem. The key has to be as least as long as the message, as otherwise it will be susceptible to a class of attacks, described in a later section about the Vigenère cipher. This will become cumbersome when trying to encrypt large amounts of data. Just imagine the preposterous solution of having to store a 40GB key for encrypting 40GB of data. There is also the potential problem of transporting the key to the receiver. If this cannot be delivered in person, would one have to encrypt this key as well? Should one engage in a cycle of encrypting keys of keys until an adversary becomes bored? Clearly, this is not a good solution.

The fundamental problem is a mathematical one. It is possible to show that any cryptosystem offering perfect security, must have a key length at least as long as the message itself [7]. Therefore, this notion of perfect security has been left for us as an unfortunate historical warning; any encryption method must have the possibility of being broken to be practical.

To overcome this fundamental limitation, we will need to lay the groundwork for what it means for a mathematical problem to be hard.

1.2.2 Big O

Following Dietzfelbinger's approach [2] we define $\mathcal{O}$ (big O) as follows.

Definition 1.2. For a function $f : \mathbb{N} \to \mathbb{R}^+$ we define $\mathcal{O}(f)$, or *big O of f* as

$$\mathcal{O}(f) = \{g \mid g : \mathbb{N} \to \mathbb{R}^+, \exists C > 0, \exists n_0 \forall n \geq n_0 : g(n) \leq C \cdot f(n)\}.$$

Informally, we may think of $\mathcal{O}(f)$ as the set of functions g, that grows at no faster rate than f.

Example 1. Suppose $g(n) \leq 15 \log n + 5(\log \log n)^2$ for all $n \geq 50$, then $g \in \mathcal{O}(\log n) \subset \mathcal{O}((\log n)^2)$.

The subset statement aims to clarify that we are indeed dealing with sets, and a statement about a function using big O, might not represent the strongest possible bound for growth. However, we will still stick to the conventional definition that $g(n) = \mathcal{O}(f(n))$ if $g \in \mathcal{O}(f(n))$, where the equality sign is used very abusively.

This approximation to the growth of a function, will be used to describe the running time of an algorithm $\mathcal{A}$. We will say that $\mathcal{A}$ *is* $\mathcal{O}(f(n))$, if the running time of $t_{\mathcal{A}}(n)$ of the algorithm on input n satisfies $t_{\mathcal{A}}(n) = \mathcal{O}(f(n))$. This is also sometimes called the complexity of $\mathcal{A}$. Using these simplifications, and other similar grammatical abbreviations, we can with reasonable accuracy say much about the growth rate without being too verbose. Following is a clear lemma that we will state without proof.

Lemma 1.3. *Suppose* $g_1(n) = \mathcal{O}(f_1(n))$ *and* $g_2(n) \in \mathcal{O}(f_2(n))$. *Then*

1. $g_1(n) + g_2(n) = \mathcal{O}(\max\{f_1(n), f_2(n)\})$.
2. $g_1(n)g_2(n) = \mathcal{O}(f_1(n)f_2(n))$.

Note that while this works for fixed size sums (independent of n), we can not induct with this statement. Say $f_i(n) = n$ for $i \in \{1, \ldots, n\}$ then $\sum f_i(n) = n^2 \notin \mathcal{O}(n)$.

1.2.3 Complexity

To properly analyze mathematical methods, or algorithms, it is important to have a measure of how big the input is. In our case, a one parameter input is sufficient, but these concepts can be easily generalized (see [6] for this and for more details on this subsection). We let $\|n\| := \lfloor \log_2 n \rfloor + 1$ be the number of bits of n. It is reasonable to use this a measure for input size because of how computers operate. It gives the ability to count the number of cycles needed to run a certain algorithm in terms of *bit operations*.

Secondly, to be able to use big O to analyze the hardness of a problem, we need to classify what hard is. A simple way of doing this is to divide functions up into so called complexity classes, by grouping them together by their smallest known running time into particular forms of big O estimates.

Definition 1.4. An algorithm $\mathcal{A}$ is said to be of polynomial complexity if its running time on input n is $\mathcal{O}((\log n)^z)$ for some $z \in \mathbb{N}$.

This is equivalent to the requirement that its bitwise complexity, i.e. its running time in terms of an input of k bits, is $\mathcal{O}(k^z)$. This is where the polynomial name comes from. If we can express the running time of of an algorithm in terms of a polynomial of the input bit size, then it satisfies this definition for some $z \in \mathbb{N}$, and we call it succinctly a polynomial time algorithm. We want encryption to be such a polynomial time algorithm, while decryption should not be so without the key. In fact ideally, we want decryption to be fundamentally hard and belong to a different complexity class.

Definition 1.5. An algorithm $\mathcal{A}$ is said to be of exponential complexity if its running time on input n is $\mathcal{O}(n^z)$ for some $z \in \mathbb{N}$.

One can also make the equivalent definition in terms of bit length, to obtain the reason behind the name, exponential (bit) complexity. A problem that can only be solved by algorithms in this class is thought of as being truly hard. However, the common definition is a little more forgiving.

Definition 1.6. A problem is said to intractable if no polynomial time algorithms exist to solve it.

In this report we will simply call any such problem hard, and any algorithm that runs in polynomial time, an efficient one. The latter is usually only done with the added informal requirement that there are no unreasonable constants in this polynomial, to render the the method useless even at large inputs. This actually calls out a rather big flaw in this simplifying nature of big O; while certain algorithms might beat others in complexity, they might not actually be faster until the input is reaches thousands of digits. This also helps the somewhat intuitive, but informal notion, that the smaller big O bound, the more complex the actual algorithm is likely to be (relative to the original bound from the original approach).

1.2.4 Complexity of Basic Operations

Continuing Dietzfelbinger's approach, we obtain a simple big O estimate of the basic mathematical operations.

Lemma 1.7. *Let $m, n \in \mathbb{N}$ be two numbers . Then*

1. *Adding or subtracting takes $\mathcal{O}(\|n\| + \|m\|) = \mathcal{O}(\log n + \log m)$ bit operations.*

2. *Multiplication takes* $\mathcal{O}(\|m\| \cdot \|n\|) = \mathcal{O}(\log n \cdot \log m)$ *bit operations.*

3. *Computing the quotient of* n *divided by* m *(denoted* n *div* m*) and the remainder* n *mod* m *takes* $\mathcal{O}((\|n\| - \|m\| + 1) \cdot \|m\|)$ *bit operations.*

Proof. Primary school methods adopted to binary all yield the claimed bounds. Note that in the big O notation, the base of the logarithm is irrelevant. □

There are other asymptotically faster ways to multiply two numbers of k bits. One example is the Karatsuba algorithm, with complexity $\mathcal{O}((k)^{\log_2 3})$, and another, is SchonhageStrassen that clocks in at $\mathcal{O}(k \log k \log\log k)$ [6]. Now a somewhat surprising result is that these algorithms also immediately generate same complexity algorithms for divisions. To see this we must pull out Newton-Raphson from our old bag of tricks [18], to help us find the reciprocal of a rational $q \neq 0$. For this to work we need a function with a zero at $1/q$, and by cleverly using the function $f(x) = \frac{1}{x} - q$, we obtain the Newton-Raphson iterate

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{\frac{1}{x} - q}{\frac{-1}{x_k^2}} = x_k + x_k(1 - qx_k) = x_k(2 - qx_k),$$

by only using multiplication and subtraction. The general quadratic convergence properties of this method, should also ensure that we do not have to do too many steps for this to be a sufficient approximation of $1/q$.

Exactly how useful these multiplication and division algorithms are in practice, is another question entirely. The documentation from MAGMA claim that these algorithms have a crossover points in efficiency (for Schonhage−Strassen vs. Karatsuba) at somewhere around an input of 2^{15} bits [19]. It is unlikely that this is going to be useful in cryptography soon. However, with Moore's law looming on the horizon, it would be unwise to discard these completely. It is clear that combining these methods with more complicated algorithms encountered later, will reduce their complexity. While this could be useful for the later discussed primality proving algorithms at the extreme ends, they are out of the range of numbers encountered in cryptography today, and will not be discussed any further.

We leave this peripheral section as a word of caution about big O; it can be terribly oversimplifying to not know the constants, but just the growth of an algorithm's running time. A more amusing side of this is obtained from writing out the Schonhage−Strassen complexity in the unsimplified non-bit form to get the preposterous $\mathcal{O}(\log n \log\log n \log\log\log n)$. Such monstrosities are not uncommon in this field, but to omit them, is to miss the chance of retelling another terrible joke.

What sound does a drowning number theorist make? *loglogloglogloglog..*

1.2.5 Fast Modular Exponentiation

A basic operation required for cryptography is the ability to raise things to large powers, in some finite ring, say a^n mod m. If we simply did n straight multiplications, then this would be an exponential time algorithm. The cleverer approach is to realize that we can repeatedly square an element most of the time. For instance $a^{100} = a^4 \cdot a^3 2 \cdot a^6 4$, and this decomposition is fully determined by the binary expansion of $100_{10} = 1100100_2$. For instance, if the last bit is a zero, $n = 0 \bmod 2$, so we can remove that bit and look at the remaining power after squaring a once.

If the last bit is a 1 somewhere in this process, then the remaining power is odd, so we need to multiply by a once to make it even. This fully determines an inductive step, and we compose the following algorithm from the less clear [4][p114].

```
Fast Modular Exponentiation
Input: Integers a, n, m, n ≥ 0
Output: a^n mod m
Method:
0   r ← 1;
1   while(n>0)
2       if (n mod 2 = 1) then
3           r ← (ar) mod m;
4           n ← n − 1;
5       else
6           a ← a^2 mod m;
7           n ← n/2;
8   return r;
```

Running through for a^{100} we see that r gradually increases from $1, a^4, a^{36}$ to finally a^{100} in the iterations when a is not squaring itself. It's clear that this would work without the modulus taken into account, but the modular version has bounded output and does not suffer from overflow issues. By taking moduli, we can additionally bound the running time of this algorithm.

Lemma 1.8. *Fast modular exponentiation runs in $\mathcal{O}(\log n)$ arithmetic operations, and $\mathcal{O}(\log n(\log m)^2)$ bit operations.*

Proof. Look at the binary representation of n. If the least significant bit is a one, the while loop needs an additional run to get rid of that bit. In the worst case the while loop needs $2\|n\| = \mathcal{O}(\log n)$ iterations. Now multiplication mod m, is always done with two numbers smaller than m, so that is $\mathcal{O}((\log m)^2)$, and by plugging a number $n < m^2 \bmod m$ into the basic complexity lemma, we get that reduction mod m also is

$$\mathcal{O}((\|n\| - \|m\| + 1) \cdot \|m\|) = \mathcal{O}((2\log m - \log m + 1)\log m) = \mathcal{O}((\log m)^2).$$

Thus, summing these complexities using the earlier lemma, give the desired overall bound. □

Chapter 2

Number Theory

To be able to formally discuss cryptography properly, we do need to lay some mathematical groundwork in number theory. Most of this is expected to be familiar, so it is run through fairly quickly. The goal of the first section is to prove a complexity bound on the Euclidean Algorithm.

2.1 Divisibility and the Euclidean Algorithm

2.1.1 G.C.D. and Divisibility in $\mathbb{Z}$

This section presents a very quick rundown of only the necessary lemmas from [2, pp23-32] about the greatest common divisor and divisibility required to continue.

Definition 2.1. For an integer n, let $D(n)$ denote the set of non-negative divisors of n.

Definition 2.2. For two integers m, n, not both zero, we define their *greatest common divisor* of n and m to be the maximal element of $D(n) \cap D(m)$. Additionally, we say m, n are *coprime* or *relatively prime* if $\gcd(m, n) = 1$.

Note that this is well defined as: the set is non-empty (it contains 1), it's a finite set whenever n and m are not both zero (by convention $D(0) = \mathbb{N}$ but $\gcd(0, 0) = 0$) and hence it has a maximum.

The gcd function have a very important property, a type of modular invariance. The corollary of this, essentially proves the validity of the basic Euclidean Algorithm.

Lemma 2.3. *For all integers n, m, x we have* $\gcd(n, m) = \gcd(n + mx, m)$

Proof. Assume without loss of generality that $m \neq 0$. Then $D(m)$ is a finite set, and it suffices to show $D(n) \cap D(m) = D(n + mx) \cap D(m)$.
($\subseteq$) If $n = du$ and $m = dv$, then $n + mx = d(u + vx)$, hence $n + mx$ is a multiple of d as well.
($\supseteq$) If $n + mx = du$ and $m = dv$ then $n = d(u - vx)$, hence n is a multiple of d as well. □

Corollary 2.4. *If $m \in \mathbb{N}$, then $\forall n \in \mathbb{Z}$* $\gcd(n, m) = \gcd(n \bmod m, m)$.

Proposition 2.5. *For $m, n \in \mathbb{Z}$ there exists integers x and y such that* $\gcd(n, m) = nx + my$.

Proof. This is not necessary to prove for the Euclidean Algorithm, and will in fact follow from its extended version. □

Proposition 2.6. *For integers m and n we have*
$\gcd(m, n) = 1 \iff \exists x, y \in \mathbb{Z}$ *such that* $nx + my = 1$.

Proof. First direction follows from previous lemma, so suppose $nx+my = 1$. Then n and m cannot both be zero, and every common divisor of m and n also divides 1. Hence $\gcd(n, n) = 1$. □

The following lemma was simply included for completeness, and states that $\mathbb{Z}$ is a Euclidean Domain. Feel free to jump to the next section.

Lemma 2.7. *(Integer Division with Remainder). Let $n \in \mathbb{Z}$ and $d \in \mathbb{N}$. Then there exist unique integers q (the quotient) and r (the remainder) such that $n = dq + r$ and $0 \leq r < d$.*

Proof. (Existence) Let q be the maximal integer such that $qd \leq n$. Since $(-|n|)d \leq n < (|n|+1)d$, such q exists and can in principle be found by searching through a finite set of integers. The choice of q implies that
$qd \leq n < (q+1)d$. We define $r := n - qd$, and conclude that $0 \leq r < d$.
(Uniqueness) Assume $n = qd + r = q'd + r'$, for $0 \leq r, r' < d$. By symmetry, we may assume that $r' - r \geq 0$, and then $0 \leq r' - r < d$.
Thus $0 = (q'-q)d+(r'-r)$, so $r'-r$ is a multiple of d. Now the only multiple of d in $\{0, 1, \ldots, d-1\}$ is 0, so we must have $r' = r$ and consequently $qd = q'd$. Since d is non-zero, we get $q = q'$. □

2.1.2 The Euclidean Algorithm

The Euclidean Algorithm, the perhaps oldest known algorithm, is a way of finding the gcd of two integers by repeatedly applying above corollary, combined with the stopping condition that $\gcd(r, 0) = r$. As this is fairly well known, we simply leave it here in the unusually elegant 1 line recursive form with a little more advanced pseudocode.

`function` $\gcd(n, m)$ { `return` $m \;=\; 0$ `?` n `:` $\gcd(m, n$ `mod` $m)$`;` }

Here the ternary operator ? is for the non-programmers a simplified *if-else* statement, with the checked condition before the operator, and the two outcomes (*true* result first) are separated by the semicolon. The function calls itself recursively until the last call returns a number, which is then passed on through all these calls to the original. The benefit of writing an algorithm down recursively is that it is easily analyzed mathematically by an induction argument. It is clear that this algorithm generates a sequence of pairs

$$(n, m) = (a_0, b_0), (a_1, b_1), \ldots, (a_t, b_t) \tag{2.1}$$

where $a_i = b_{i-1}$ and $b_i = a_{i-1} \bmod b_{i-1}$ for $1 \leq i \leq t$.

Lemma 2.8. *(Efficiency of EA) The Euclidean Algorithm does at most $2\min\{\|n\|, \|m\|\} = \mathcal{O}(\min\{\log n, \log m\})$ recursive calls, and the number of bit operations made is $\mathcal{O}(\log n \log m)$.*

Proof. From [2]. It is clear that the sequence of pairs (a_i, b_i) generated above is strictly decreasing in both variables, and while (a_i) converges to the gcd, b_i converges to 0. Note that (a_i) and (b_i) converge simultaneously.
If $b_{i+1} > \frac{1}{2}b_i = \frac{1}{2}a_{i+1}$ then $b_{i+2} = a_{i+1} - b_{i+1} < \frac{1}{2}b_i$. On the other hand,
if $b_{i+1} \leq \frac{1}{2}b_i = \frac{1}{2}a_{i+1}$ then $b_{i+2} = a_{i+1} \bmod b_{i+1} < b_{i+1} \leq \frac{1}{2}b_i$.
So in either case, the b_i's are decreasing in bitlength at least every other iteration, so it would take at most $2\min\{\|n\|, \|m\|\}$ iterations for it to converge.
Now to divide a_i by b_i t times, we need at most $\mathcal{O}((\|a_i\| - \|b_i\| + 1) \cdot \|b_i\|)$ bit operations. Note that $b_i = a_{i+1}$ for $0 \leq i < t$ and $b_t = 0$. Thus

$$(\|a_i\| - \|b_i\| + 1) \cdot \|b_i\| = \|a_i\|\|b_i\| - \|a_{i+1}\|\|b_i\| + \|b_i\| \leq (\|a_i\| - \|a_{i+1}\|)\|b_0\| + \|b_i\|.$$

So we need

$$\sum_{0\leq i<t} \mathcal{O}((\|a_i\| - \|b_i\| + 1) \cdot \|b_i\|) = \mathcal{O}(\sum_{0\leq i<t} (\|a_i\| - \|a_{i+1}\|)\|b_0\| + \|b_i\|)$$

and, by splitting the sum and canceling telescoping terms, this is

$$= \mathcal{O}(\|b_0\|(\|a_{t-1}\| - \|a_0\|) + t\|b_0\|) = \mathcal{O}(\|b_0\|\|a_0\| + t\|b_0\|) = \mathcal{O}(\|m\|\|n\|).$$

□

The step where $\mathcal{O}$ is taken outside a sum makes sense as this is really shorthand for saying that we are summing a set of functions that are $\mathcal{O}(g_i(n))$. The resulting function is clearly $\mathcal{O}(\sum g_i(n))$.

This is the familiar (basic) Euclidean Algorithm. However, we do need to modify this to obtain integers x, y such that $\gcd(m, n) = mx + ny$. With a preference for the recursive formulation established, we present the Extended Euclidean Algorithm in this format.

```
function extended_gcd(n, m)    {
    if (n mod m = 0) return {0, 1};
    {x, y} := extended_gcd(m, n mod m);
    return {y, x − y ∗ (n div m)};
}
```

It is clear that we are not really doing anything more complex than we were already performing, in the normal Euclidean Algorithm. We are simply keeping track of different variables, and this does not affect the $\mathcal{O}(\log n \log m)$ bit complexity. Most importantly, however, it works.

Lemma 2.9. *The Extended Euclidean Algorithm outputs a pair (x, y) such that $nx + my = \gcd(n, n)$.*

Proof. Let $d := gcd(n, m)$, and note that if $m \mid n$ then $n \equiv 0 \bmod m$ and so $d = 0n + 1m$. The recursive call in line two is applied to the pair $(m, n \bmod m)$ so we can assume inductively that $mx + (n \bmod m)y = d$. Consequently,

$$mx - m(n \operatorname{div} m)y + (n \bmod m)y + m(n \operatorname{div} m)y = d.$$

Noticing that for any $a, b \in \mathbb{Z}$ the relation $a = a \bmod b + b(a \operatorname{div} b)$ holds, the main expression thus contracts to

$$m(x - (n \operatorname{div} m)y) + ny = d.$$

This shows that the new pair also satisfies the desired relation, completing the induction argument. □

Typically this is presented as a long, non-recursive function keeping track of one additional value (the gcd). This is clearly much more aesthetically pleasing, and one can obviously reclaim the gcd from m, n, x, y.

As an indication of the how common this algorithm is, these elegant recursive formulations are rewritten from versions, that were already very pleasing on wikipedia. The proof is still mathematically solid, but the point was merely to show the concept simply, as there are other faster implementations of the Euclidean Algorithm available (see for instance the quite recent recursive binary gcd [22]).

2.2 Finite Groups

2.2.1 Cyclic Groups and The Chinese Remainder Theorem

Many cryptographic systems builds on finite groups, or fields, and the notions of the orders of elements. We will not define everything required from group theory, but rather give a quick rundown on the most important notions, and some of their proofs. This will be done regardless of how deeply these proofs depend on auxiliary statements within group theory. The rest of this chapter will serve strictly as preliminary number theory, and will not really talk about cryptography at all. Their use will only become apparent later, while discussing certain methods or cryptosystems. We begin with defining the finite cyclic group, and its multiplicative group.

Definition 2.10. Let $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ be the cyclic group of n elements.

Definition 2.11. For $n \geq 1$, we define the the multiplicative group of $\mathbb{Z}_n$ to be

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \gcd(a, n) = 1\}. \tag{2.2}$$

Using this simplified notation for $\mathbb{Z}/n\mathbb{Z}$ might run the risk of confusion, but there will be no references to $p-$adic integers in this report. Now, we have not shown that the defined version of the multiplicative group actually is a group yet. We thus present the following three results from [2, pp34-38], with perhaps slightly shorter proofs.

Proposition 2.12. *$\mathbb{Z}_n^*$ is the multiplicative group modulo n of all invertible elements in $\mathbb{Z}_n$.*

Proof. We will rely heavily on Proposition 2.6. If $a, b \in \mathbb{Z}_n^*$ then $1 = ax + ny = bu + nv$ for some integers x, y, u, v. Then

$$(ab) \cdot (xu) = (ax)(bu) = (1 - ny)(1 - nv) = 1 - n(y + v - nyv).$$

Hence $\gcd(ab \bmod n, n) = \gcd(ab, n) = 1$ showing that $\mathbb{Z}_n^*$ is closed under multiplication. Clearly $1 \in \mathbb{Z}_n^*$, so it also forms a group. We will show that its elements are invertible, and no other invertible elements exist in $\mathbb{Z}_n$. So we suppose $a \in \mathbb{Z}_n^*$, so $1 = ax + ny$ for some integers x, y. Then $ax - 1 = -ny \equiv 0 \bmod n$, and $ax \equiv 1 \bmod n$. Thus $x = a^{-1}$ in $\mathbb{Z}_n^*$.
Conversely, if $b \in \mathbb{Z}_n$ with $ab \equiv 1$, then $ab - 1 = nx$ for some x, which implies $\gcd(b, n) = 1$, so $b \in \mathbb{Z}_n^*$. □

This shows that we can compute the inverses of elements in $\mathbb{Z}_n$ with help of the extended Euclidean Algorithm.

Theorem 2.13. *(Chinese Remainder Theorem) Let $n = n_1 \cdots n_r$ for $n_1, \ldots, n_r$ relatively prime integers. Then*

$$\mathbb{Z}_n \cong \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r} \tag{2.3}$$

Proof. We will show that the map

$$\Phi : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r},\ a \mapsto (a \bmod n_1, \ldots, a \bmod n_r)$$

is an isomorphism. It is clear that this is a homomorphism, whose kernel consists of elements divisible by all the n_i. But since these are all coprime, the kernel must be in the ideal generated by n, and this exactly what it means for Φ to be injective. Now, since $n = |\mathbb{Z}_n| = n = n_1 \cdots n_r = |\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r}|$, it must also be true that Φ is also surjective. □

The Chinese Remainder Theorem (CRT) is important for many cryptographic application because it means that operations in $\mathbb{Z}_n$ for n as a product of two primes, can be performed in the isomorphic ring $\mathbb{Z}_p \times \mathbb{Z}_q$. Many schemes use this, or some modification of this ring. Another interesting consequence of CRT, is that it actually provides a bijection between the corresponding multiplicative groups.

Lemma 2.14. *Let $n = n_1 n_2$ be the product of two coprime integers, and $\Phi(a) = (a_1, a_2)$ from CRT. Then $a \in \mathbb{Z}_n^* \iff a_1 \in \mathbb{Z}_{n_1}^*, a_2 \in \mathbb{Z}_{n_2}^*$.*

Proof. Still relying heavily on Proposition 2.6, we suppose $a \in \mathbb{Z}_n^*$, thus $1 = ax + ny$ for some $x, y \in \mathbb{Z}$. Now $a = a_1 + kn_1$ for some k, so $1 = (a_1 + kn_1)x + ny = a_1 u + (kx + n_2 y)n_1$ which implies $a_1 \in \mathbb{Z}_{n_1}^*$. The proof for a_2 is similar.
Conversely, suppose $a_1 \in \mathbb{Z}_{n_1}^*$ and $a_2 \in \mathbb{Z}_{n_2}^*$. Find integers x_1, x_2, y_1, y_2 such that $a_1 x_1 + n_1 y_1 = 1 = a_2 x_2 + n_2 y_2$. By CRT the congruences $x \equiv x_1 \bmod n_1$ and $x \equiv x_2 \bmod n_2$ admits a solution $x \in \mathbb{Z}_n$ and since

$$ax \equiv a_1 x_1 \equiv 1 \bmod n_1, \text{ and } ax \equiv a_1 x_1 \equiv 1 \bmod n_2,$$

the uniqueness of this solution in fact forces $ax \equiv 1 \bmod n$ which means that a is an invertible element in $\mathbb{Z}_n$. □

2.2.2 Euler Phi

We proceed to the Euler phi, also called the Euler Totient fuction.

Definition 2.15. (Euler Totient) For all $n \in \mathbb{N}$ the Euler function ϕ is defined by $\phi(n) := |\mathbb{Z}_n^*|$.

With the above groundwork, certain famous results about Euler phi become trivial.

Corollary 2.16. *(Properties of the Euler ϕ function)*

1. *If $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.*
2. *If p is prime, then $\phi(p^r) = p^r\left(1 - \frac{1}{p}\right)$.*
3. *$\sum_{d|n} \phi(d) = n$*

Proof. (1) From [2]. Lemma above says that $|\mathbb{Z}_{mn}^*| = |\mathbb{Z}_m^*| \cdot |\mathbb{Z}_n^*|$. We calculate (2) as follows

$$\phi(p^r) = |\{a | 1 \leq a < p^r, p \nmid a\}| = p^r - |\{a | 1 \leq a < p^r, p | a\}| = p^r - p^{r-1} = p^r\left(1 - \frac{1}{p}\right).$$

(3) is less trivial and taken from [3]. Denote $f(n) = \sum_{d|n} \phi(d)$, and note that for any prime p

$$f(p^r) = \sum_{d|p^r} \phi(d) = \sum_{i=0}^{r} \phi(p^i) = 1 + \sum_{i=0}^{r} (p^i - p^{i-1}) = p^r.$$

Now $\phi(n)$ is multiplicative (i.e. property (1) holds), and it suffices to show that $f(n)$ is as well. To see this, note that any divisor $d \mid mn$ (when n, m are coprime) can be factored uniquely into d_1 and d_2 where $d_1 \mid m$, $d_2 \mid n$. Since $\gcd(d_1, d_2) = 1$, we have $\phi(d) = \phi(d_1)\phi(d_2)$ by (1). We get all possible divisors d of mn by taking all possible pairs (d_1, d_2) where d_1 is a divisor of m and d_2 is a divisor of n. Thus

$$f(mn) = \sum_{d_1|m} \sum_{d_2|n} \phi(d_1)\phi(d_2) = \left(\sum_{d_1|m} \phi(d_1)\right)\left(\sum_{d_2|n} \phi(d_2)\right) = f(m)f(n).$$

□

Corollary 2.17. *If $n \in \mathbb{N}$ has prime factorization $n = p_1^{a_1} \cdots p_k^{a_k}$, then*

$$\phi(n) = \prod_{i=1}^{k} p_i^{a_i - 1}(p_i - 1) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right). \tag{2.4}$$

2.2.3 Euler's Theorem

With Euler Phi fully described, the following famouse theorem says that if you raise any invertible element of $\mathbb{Z}_n$ to the power of the order of the multiplicative group, you get 1. Consequently, the uniqueness of inverses gives another way to obtain inverses via $a^{-1} \equiv a^{\phi(n)-1}$.

Theorem 2.18. *(Euler's Theorem) Let a, n be coprime integers. Then $a^{\phi(n)} \equiv 1$ mod n.*

Proof. From [4, p112]. Let $r_1, \ldots, r_{\phi(n)}$ denote elements in $\mathbb{Z}_n^*$. Then $ar_1, \ldots, ar_{\phi(n)}$ are all invertible, and since a is invertible modulo n, no two elements are congruent (for $ar_j \equiv ar_i \implies r_j \equiv r_i$ when a is invertible). We now have two different representations of $\mathbb{Z}_n^*$, and so

$$ar_1 \cdots ar_{\phi(n)} \equiv r_1 \cdots r_{\phi(n)} \bmod n.$$

Equivalently, by rearranging

$$a^{\phi(n)} r_1 \cdots r_{\phi(n)} \equiv r_1 \cdots r_{\phi(n)} \bmod n.$$

Which yields the result after cancelation of the invertible r_i. □

The famous corollary to this goes by the name of Fermat's little theorem. This is simply the special case of $n = p$ prime, so $\phi(p) = p - 1$.

2.2.4 Finite Fields and Primitive Roots

From the above results on $\mathbb{Z}_n$ it is clear that this is a field if and only if n is prime. For this reason, it makes sense to define $\mathbb{F}_q$ to be *the* finite field of q elements. From Galois theory, we know that q can either be a power of a prime, or simply a prime (in which case it coincides with $\mathbb{Z}_q$), but we will not prove this here.

The the multiplicative group $\mathbb{F}_q^*$ of the finite field $\mathbb{F}_q$, similarly contains all the invertible elements of $\mathbb{F}_q$. If we recall as well that the order of an element in a group is the least positive power of that element that gives 1, then by the definition of a field, this must be the $q - 1$ non-zero elements. Thus, we will simply state the following elementary group theoretic lemma in the special case of $\mathbb{F}_q^*$.

Lemma 2.19. *(Lagrange's Theorem) The order of any $a \in \mathbb{F}_q^*$ divides $q - 1$.*

Definition 2.20. A generator g of a finite field $\mathbb{F}_q$ is an element of order $q - 1$. In other words, the powers of g run through all the elements of $\mathbb{F}_q^*$.

The following are handpicked and proved from [3, pp33-43]. These will provide just the necessary machinery to prove certain theorems about quadratic residues.

Proposition 2.21. *Every finite field has a generator. If g is a generator of $\mathbb{F}_q^*$, then g^j is another generator if and only if* $\gcd(j, q - 1) = 1$.

Proof. We prove the slightly more general version of the second statement first: If a has order d, then a^j is another element of order d if and only if $\gcd(j,d) = 1$. By the lemma above, we note that our d must divide $q-1$.
If $\gcd(j,d) > 1$, then $(a^j)^{d/\gcd(j,d)} = 1$ and $d/\gcd(j,d) < d$ proving the forward implication. Conversely, pick j with $\gcd(j,d) = 1$ and suppose for the sake of contradiction that the order of a^j is $l < d$. Since j and d are coprime we find integers x, y such that $1 = xj + yd$. Thus $a^l = a^{l(xj+yd)} = (a^{lj})^x \cdot (a^{ld})^y = 1$, contradicting the minimality of the order d of a.
We have shown that for each $d \mid q-1$ there are either $\phi(d)$ or 0 elements of order d. Fortunately, every element has some order $d \mid q-1$, and by Corr 2.16 we know that $\sum_{d|q-1} \phi(d) = q-1 = |\mathbb{F}_q^*|$. So the only way that every element can have some order $d \mid q-1$ is if there are always $\phi(d)$ elements of order d. □

In particular there are $\phi(q-1)$ different generators of $\mathbb{F}_q^*$.

Proposition 2.22. *Let g be a generator of $\mathbb{F}_q^*$. Then g^j is an nth root of unity if and only if $nj \equiv 0 \bmod q-1$. The number of nth roots of unity is $\gcd(n, q-1)$. In particular, $\mathbb{F}_q$ has a primitive nth root of unity (i.e. an element ζ such that all the powers of ζ run through the nth roots of unity) if and only if $n \mid q-1$. If ζ is a primitive nth root of unity in $\mathbb{F}_q$, then ζ^j is also a primitive nth root if and only if $\gcd(j,n) = 1$.*

Proof. Any element of $\mathbb{F}_q^*$ can be written as a power g^j of the generator g. A power of g is 1 if and only if the power is divisible by $q-1$. Thus, an element g^x is an nth root of unity if and only if $nx \equiv 0 \bmod q-1$. Next, let $d = \gcd(n, q-1)$. The equation $nx \equiv 0 \bmod q-1$ is equivalent to the equation $\frac{n}{d}x \equiv 0 \bmod \frac{q-1}{d}$. Since n/d and $(q-1)/d$ are coprime, the latter congruence is equivalent to requiring x to be a multiple of $(q-1)/d$. In other words, the d distinct powers of $g^{(q-1)/d}$ are precisely the nth roots of unity. There are n such roots if and only if $d = n$, i.e. $n \mid q-1$. Finally, if $n \mid q-1$, let $\zeta = g^{(q-1)/n}$. Then $\zeta^j = 1$ if and only if $n \mid j$. The kth power of ζ^j equals 1 if and only if $kj \equiv 0 \bmod n$. It is easy to see that ζ^j has order n if and only if j and n are coprime. Thus, there are $\phi(n)$ different primitive nth roots of unity if $n \mid q-1$. □

2.3 Quadratic Residues

A particularly interesting topic in number theory, is the classification of quadratic residues. One is tempted to think that this topic is the typical theoretical abstraction with very few applications in cryptography. In fact, quoting Samir Siksek in a lecture at University of Warwick about quadratic residues he goes, albeit slightly paraphrased, *"But Samir, why would this theorem ever be useful? Well. I will tell you. One year from now, when you are still queuing up at the employment office for jobs, you will look around you, see all these other people, and sneer at them. These people do not even know the law of quadratic reciprocity... But other than that.."*

2.3.1 The Legendre Symbol

We follow the approach of Koblitz [3, pp43-47] work with Legendre symbol.

Definition 2.23. We say a non-zero integer a is a quadratic residue mod n if there exists a solution to the equation $x^2 \equiv a \pmod{n}$. If no such x exist, we say a is a quadratic non-residue mod n.

We are usually interested determining whether of not a is a quadratic residue in the case that n is prime.

Lemma 2.24. *Let p be a prime. Then $x^2 \equiv y^2 \pmod{p} \iff x \equiv \pm y \pmod{p}$.*

Proof. If $x^2 \equiv y^2 \pmod{p}$ then $p \mid (x^2 - y^2) = (x - y)(x + y)$. But p is prime, so $p \mid (x - y)$ or $p \mid (x + y)$. Thus $x \equiv \pm y \pmod{p}$. □

This means that every square has exactly two solutions to $x^2 \equiv a \pmod{p}$, so by squaring all non-zero elements of the field $\mathbb{F}_p$ we get exactly $(p-1)/2$ quadratic residues, and $(p-1)/2$ quadratic non-residues. For brevity, we let $Q_n$denote the set of quadratic residues mod n, and conversely $\overline{Q}_n$the set of quadratic non-residues. We exclude zero from the set Q_n, despite the fact that $0^2 = 0$, to obtain the nice relation $|\overline{Q}_p| = |Q_p|$.

Note that these sets are informally considered as equivalence class representatives under the equivalence relation congruence. I.e. by $a \in Q_p$, we mean that the remainder of $a \pmod{p}$ is one of these $(p-1)/2$ elements less than p, that have a square root in $\mathbb{F}_p$.

Definition 2.25. For a prime $p > 2$ and an integer a we define the *Legendre Symbol* $\left(\frac{a}{p}\right)$ to be

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p \mid a \\ 1 & \text{if } a \in Q_p \\ -1 & \text{if } a \in \overline{Q}_p \end{cases} \tag{2.5}$$

Theorem 2.26. *(Euler's Criterion)*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}. \tag{2.6}$$

Proof. If p divides a then both sides are zero mod p, so suppose $p \nmid a$. By Fermat's Little Theorem $a^{(p-1)/2}$ squares to one, so $a^{(p-1)/2}$ is ± 1 by the lemma above. Let g be a generator of $\mathbf{F}_\mathbf{p}^*$ and find j such that $a \equiv g^j \pmod{p}$. There are $(p-1)/2$ elements in Q_p, and by considering all the even powers of the generator, we can find them all.
Now $g^{j(p-1)/2} = 1 \iff j(p-1)/2$ is a multiple of $p-1 \iff j$ is even. So $a^{(p-1)/2} = g^{j(p-1)/2} = 1 \iff j$ is even $\iff a \in Q_p$. □

Proposition 2.27. *(Properties of the Legendre Symbol)*

1. *If $a \equiv b \pmod{p}$ then $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$*
2. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$
3. $\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$ *for b and p coprime.*
4. $\left(\frac{1}{p}\right) = 1$ *and* $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2} = \begin{cases} 1 & \text{if } p \equiv 1 \pmod 4 \\ -1 & \text{if } p \equiv 3 \pmod 4 \end{cases}$
5. $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8} = \begin{cases} 1 & \text{if } p \equiv 1 \text{ or } 7 \pmod 8 \\ -1 & \text{if } p \equiv 3 \text{ or } 5 \pmod 8 \end{cases}$

Proof. Part 1) is clear by the informal definition of Q_p, and part 2) and 4) follows from the representation in the right hand side of Euler's Criterion. Part 3) follows immediately from part 3) by noting that $(\frac{b^2}{p}) = 1$ as any (non-zero) square is a quadratic residue.

Part 5) is non-trivial. We start by defining $f(n) = (-1)^{(n^2-1)/8}$ for odd n, and zero for even n. We will show that $(\frac{2}{p}) = f(p)$. Since $p^2 \equiv 1 \pmod 8$ for all odd primes p we know by Prop. 2.22 that the field $\mathbb{F}_{p^2}$ contains a primitive 8th root of unity. We let ζ denote this primitive root, and define $G = \sum_{j=0}^{7} f(j)\zeta^j$. We have $G = \zeta - \zeta^3 - \zeta^5 + \zeta^7 = 2(\zeta - \zeta^3)$ by the definition of f and by noting that $\zeta^4 = -1$. Thus $G^2 = 4(\zeta^2 - 2\zeta^4 + \zeta^6) = 8$. We can therefore write G^p as

$$G^p = (G^2)^{(p-1)/2}G = 8^{(p-1)/2}G = \left(\frac{8}{p}\right)G = \left(\frac{2}{p}\right)G.$$

Note we have used Euler's Criterion to rewrite the first factor and part 3) to get rid of a power of two. Now our function f satisfies $f(j)^p = f(j)$, as $(-1)^p = (-1)$ for p odd, and less trivially $f(j) = f(p)f(pj)$. We can verify the latter by verifying the equivalent logical statement

$$(p \equiv \pm 1 \wedge pj \equiv \pm 1) \vee (p \equiv \pm 3 \vee pj \equiv \pm 3) \iff (j \equiv \pm 1)$$

using the mod 8 version of $f(j)$. I.e. the terms on the right hand side are either both 1 or both -1 if and only if the left hand side is 1. Additionally, in our finite field $\mathbb{F}_{p^2}$, which has characteristic p, we know $(a+b)^p = a^p + b^p$ (see 4.8). So by direct calculation, we now obtain a different expression for G^p by

$$G^p = \sum_{j=0}^{7} f(j)\zeta^{pj} = \sum_{j=0}^{7} f(p)f(pj)\zeta^{pj} = f(p)\sum_{k=0}^{7} f(k)\zeta^k = f(p)G.$$

The first step uses the distributivity of the p'th power and the last substitution is valid as if j runs through $0, 1, ..., 7$ so does pj mod 8. Comparing these two expressions for G^p, and dividing by the non-zero element G (as $G^2 = 8$) reveals $f(p) = \left(\frac{2}{p}\right)$ proving part 5. □

2.3.2 Quadratic Reciprocity

The last part of the theorem was really a lot harder than the other properties of the Legendre symbol, and it is in fact often associated with the much stronger law of quadratic reciprocity. The proofs of these are a bit long, but we have modified them to a much clearer format, compared to how they were presented in [3]. Additionally, we would like to point out that of the three different variants that we have come across, this is by far the cleanest.

Theorem 2.28. *(Law of Quadratic Reciprocity) Let p, q be odd primes. Then*

$$\left(\frac{q}{p}\right) = (-1)^{(p-1)(q-1)/4}\left(\frac{p}{q}\right) = \begin{cases} \left(\frac{p}{q}\right) & \text{if } p \equiv q \equiv 3 \pmod 4 \\ -\left(\frac{p}{q}\right) & \text{if } p \text{ or } q \equiv 1 \pmod 4 \end{cases} \tag{2.7}$$

Proof. Let $f = q - 1$. By by 2.22 we can find a primitive q-th root of unity ζ in $\mathbb{F}_{p^f}$. Similarly to the last Legendre property proof, we define $G = \sum_{j=0}^{q-1}\left(\frac{j}{q}\right)\zeta^j$. Noting that we can truncate the first term, we get

$$G^2 = \left(\sum_{j=1}^{q-1}\left(\frac{j}{q}\right)\zeta^j\right)\left(\sum_{k=1}^{q-1}\left(\frac{k}{q}\right)\zeta^k\right) = \left(\sum_{j=1}^{q-1}\left(\frac{j}{q}\right)\zeta^j\right)\left(\sum_{k=1}^{q-1}\left(\frac{-k}{q}\right)\zeta^{-k}\right)$$

$$= \sum_{j=1}^{q-1}\sum_{k=1}^{q-1}\left(\frac{j}{q}\right)\zeta^{j}\left(\frac{-k}{q}\right)\zeta^{-k} = \left(\frac{-1}{q}\right)\sum_{j=1}^{q-1}\sum_{k=1}^{q-1}\left(\frac{jk}{q}\right)\zeta^{j-k}$$

We have substituted k for $-k$ in one copy of G in the first line, used property 2 of the Legendre symbol in line 2. Now substitute k for kj in the inner sum. This is valid as the inner sum depends only on the residue mod q, and when k runs through these, so does kj. Thus

$$= (-1)^{(q-1)/2}\sum_{j=1}^{q-1}\sum_{k=1}^{q-1}\left(\frac{j^2k}{q}\right)\zeta^{j(1-k)} = (-1)^{(q-1)/2}\sum_{k=1}^{q-1}\left(\frac{k}{q}\right)\sum_{j=1}^{q-1}\zeta^{j(1-k)}$$

$$= (-1)^{(q-1)/2}\sum_{k=1}^{q-1}\left(\frac{k}{q}\right)\sum_{j=0}^{q-1}\zeta^{j(1-k)} = (-1)^{(q-1)/2}\left(\frac{1}{q}\right)\sum_{j=0}^{q-1}\zeta^{0} = (-1)^{(q-1)/2}q.$$

Here we changed the order of summation, and in the second line added $\sum_{k=1}^{q-1}\left(\frac{k}{q}\right)$ to the sum to get the index of j down to 0 (note this addition is zero as $|Q_q| = |\overline{Q}_q|$). The final step comes from recognizing that the sum of the distinct powers of all the primitive roots is always zero. We are half-way done.

We now work out two expressions for G^p along the lines of property 5. Using our expression for G^2 and Euler's Criterion (working in a field of characteristic p justifies the mod p comparison) we have

$$G^p = (G^2)^{(p-1)/2}G = \left((-1)^{(q-1)/2}q\right)^{(p-1)/2}G$$

$$= (-1)^{(p-1)(q-1)/4}q^{(p-1)/2}G = (-1)^{(p-1)(q-1)/4}\left(\frac{q}{p}\right)G.$$

The second expression, we have yet to work out uses the distributivity $(a+b)^p = a^p + b^p$ in characteristic p again, and the trivial observation that $\left(\frac{j}{p}\right)^p = \left(\frac{j}{p}\right)$ as such

$$G^p = \left(\sum_{j=1}^{q-1}\left(\frac{j}{q}\right)\zeta^{j}\right)^p = \sum_{j=1}^{q-1}\left(\frac{j}{q}\right)\zeta^{(jp)} = \left(\frac{p}{q}\right)\sum_{j=1}^{q-1}\left(\frac{pj}{q}\right)\zeta^{(jp)} = \left(\frac{p}{q}\right)G.$$

This uses property 2 and 3 of the Legendre symbol, while the very last step changes the variables $j' = pj$ in the sum to obtain G. Now simply compare the two expressions for G^p, and cancel the non-zero element G (as $G^2 = \pm q$) to obtain the result. □

2.3.3 The Jacobi Symbol

Finally, we generalize the construction to pairs a, n where n is not assumed to be prime.

Definition 2.29. (Jacobi Symbol) Let $n = p_1^{a_1}\cdots p_k^{a_k}$ and $a \in \mathbb{Z}$ We define the Jacobi Symbol of a over n to be

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{a_1}\cdots\left(\frac{a}{p_k}\right)^{a_k} \tag{2.8}$$

Note that this construction does not have the same property as the Legendre symbol with respect to determining whether or not a is a quadratic residue mod n. For instance, with n composite, there is no guarantee that because $\left(\frac{a}{n}\right) = 1$, we have $a \in Q_n$. However, with this construction all the nice properties of the Legendre symbol carry over. Most of them do so trivially, so we will only explain the quadratic reciprocity laws.

Proposition 2.30. *Let a be an integer and n an odd positive integers. Then*

1. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$
2. $\left(\frac{m}{n}\right) = (-1)^{(m-1)(n-1)/4}\left(\frac{n}{m}\right)$

Proof. (1) Let $f(n) = (-1)^{(n^2-1)/8}$ and note like earlier that $f(nm) = f(n)f(m)$. Thus

$$f(n) = f(p_1)^{a_1} \cdots f(p_k)^{a_k} = \left(\frac{2}{p_1}\right)^{a_1} \cdots \left(\frac{2}{p_k}\right)^{a_k} =: \left(\frac{2}{n}\right).$$

For (2) we assume without loss of generality that $\gcd(m, n) = 1$ (otherwise both sides are zero). Write $m = p_1 \cdots p_r$ and $n = q_1 \ldots q_s$ where the some prime factors may be repeated. We now convert $\left(\frac{m}{n}\right) = \prod_{i,j}\left(\frac{p_i}{q_j}\right)$ to $\left(\frac{n}{m}\right) = \prod_{i,j}\left(\frac{q_j}{p_i}\right)$ by applying the quadratic reciprocity law rs times. The number of sign changes we get is the number of both p_i and q_j that are $\equiv 3 \bmod 4$, i.e. it's the product of primes $\equiv 3 \bmod 4$ in the factorization of m and in the factorization of n. Thus, $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$ unless there are an odd number primes $\equiv 3 \bmod 4$ in both factorizations, in which case $\left(\frac{m}{n}\right) = -\left(\frac{n}{m}\right)$. But a product of odd primes is $\equiv 3 \bmod 4$ if and only if it contains an odd number of primes which are $\equiv 3 \bmod 4$, so our proof is complete. □

Chapter 3

Cryptography

The process of encryption, also called enciphering, is running a plaintext $P \in \mathcal{P}$ through an algorithm $f_E : \mathcal{P} \to \mathcal{C}$. The process of decryption, or deciphering, is running a ciphertext $C \in \mathcal{C}$ through an algorithm $f_D : \mathcal{C} \to \mathcal{P}$. The sets $\mathcal{P}, \mathcal{C}$ are called the sets plaintexts and ciphertexts respectively, and depending on the system, these will vary. We will use an informal version of Katz and Lindell's [7] definition of a cryptosystem, and explain some of the key nuances as we go along.

Definition 3.1. (Cryptosystem) A cryptographic system is a set of probabilistic or deterministic algorithms for encryption, decryption and for the generation of keys. The encryption and decryption algorithms will be denoted (f_E, f_D) and will be referred to as the cipher algorithms or cipher functions with the requirement that $f_D \circ f_E = 1_{\mathcal{P}}$.

The latter requirement states that decryption should be the right inverse of encryption, and $f_D(f_E(P)) = P$ for all plaintexts $P \in \mathcal{P}$. However, this notation is slightly abusive in a general context, when dealing with complicated probabilistic algorithms. Nevertheless, for the cryptographic systems we will be dealing with, this notation will suffice. Additionally, it is worth noting that as functions, encryption and decrypting pass through the ciphertexts and the plaintexts sets, that may or may not be different depending on your algorithms. However, generally they operate on integers, or integers modulo some integer. For example, we can represent one character using a number and encrypt each letter separately. Alternatively, we can consider larger fixed length blocks of characters, that will be padded with zeroes to fit. The latter will be demonstrated to be the better solution in the next section, where we will drop the $\mathcal{P}, \mathcal{C}$ notation in favour of the actual sets.

For the most of this chapter, we will ignore the topic of key generation and assume that we have efficient and reliably random methods for computing the necessary keys, to work with the described cryptosystems. We also note that this definition says nothing about the strength or security of the system. Initially, this will be discussed informally with some simple systems, but will be approached more formally towards the the end, before we start dealing with serious cryptographic systems.

Recalling Kerchoff's principle, it is important to note that there may still be benefits, from secrecy. For instance one can mask flaws that one does not know how to correct immediately. However, these bonuses are considered null and void for the academic purposes of this report, and we will

focus on the provable security features of a system, rather than the so called security through obscurity, that this practice amounts to.

3.1 Private Key Cryptography

Before we go into public key systems, we will start with some simple examples of private key cryptosystems, as an illustration of the common hazards of cryptography. For a detailed account of this, we refer to Beutelspacher[8][pp1-66], but we will present some of the more important ideas from his book.

A private key cryptosystem, also called a symmetric cryptosystem or cipher, is simply a cryptosystem with the same key used for encryption and decryption. An example would be the shift cipher, which is commonly referred to as Caesar's cipher, when dealing with individual letter encryption.

Definition 3.2. (Shift ciphier) Let $s, N \in \mathbb{N}$. The shift cipher algorithms is the pair consisting of $f_E(P) = P + s \bmod N$ and $f_D(C) = C - s$)

Here the secret key is the integer $s < N$, and for $N = 26$ we are simply shifting letters around the alphabet, and the result looks like gibberish, as it should. Unfortunately, this is a useless construction in terms of secrecy, as a simple guess among the 26 different values of s grant the secret key. It is also possible to merely look at the frequency of the ciphertext letters, and try to match them up with the known distribution of letters in a given language. Clearly a requirement for a strong cryptosystem, therefore must be that it has a large keyspace, and one cannot infer anything about the plaintext from the ciphertext (in polynomial time). If the latter is possible, then we say the cipher can be broken by a known ciphertext attack, and it is placed pretty much on the lowest echelon of security. On the other hand, if this is proven infeasible, then the cipher is said to be semantically secure.

One possible way to try to improve this, is to encrypt a k letter block and work modulo N^k for integers of the form $\sum_{i=1}^{k} a_i N^{k-i}$, where a_i denotes the ith letter in the plaintext. However, this is also has problems when the number of characters available per letter (N) is known (reasonable to assume by Kerchoff). For instance, taking the ciphertext mod N gives the last letter in each k letter block, so given more ciphertexts, we can potentially work out s mod N by looking at letter frequencies [3, p61]. By increasing the powers of N, it is possible to work out s iteratively this way. Thus this cipher offers no real security, and its use should be highly discouraged.

In 1586, Blaise de Vigenère published an enhancement of the shift cipher (for single letter encryption), that is now popularly referred to as the Vigenère cipher [8, pp27-44]. The main idea is use a keyword, of say length l, turn it into numbers via the natural correspondence $\{a, b, \ldots, z\} \leftrightarrow \mathbb{Z}_{26}$, and shift the ith letter of a plaintext by the ($i \bmod l$)th keyword number. Essentially, this means we combine l shift ciphers (whose shift factors are described by the keyword) to create a conceivably stronger cipher. It is clear that by going from a length one keyword (an ipso facto shift cipher) to a length $l > 1$, letter frequencies will be equalized somewhat (compared to the plaintext) as the most common letters get shifted around to more than one letter, depending on the keyword.

Unfortunately, the security of this cipher also depends highly of the keeping the key length l secret, as for if this is known, then every letter that lie in the same residue class mod l, use the same

shift; thus, it can be analyzed via letter frequencies to work out a letter in the key. One method for finding l, is to look at repeating series of digraphs (2 letter blocks), and trigraphs (3 letter blocks), and look at multiples of distances between their repetitions. The only way this cipher would be considered secure nowadays, is if we used a key as long as the plaintext (eliminating attacks to obtain l), and that this keyword was a random stream of letters (not a long English paragraph as the letter frequency of this would rub of in the ciphertext somehow). Clearly this just creates more problems than it solves, and we will not investigate this further.

The main lesson from this section on Vigenère, is that just because a cipher seems complicated, does not in any way make it strong. Most of the ciphers we will discuss in the next chapter are based on a mathematical problem, that have been considered to be hard for many centuries.

3.2 Public Key Cryptography

A public key cryptosystem, also called an asymmetric cryptosystem or cipher, is a cryptosystem with a publicity available encryption key, and a secret decryption key, which should be unobtainable. Essentially, we follow the definition of a cryptosystem, but now with modification that the encryption algorithm and all its parameters are publicly available. This is an analogue of mailbox security; everyone can drop a message into your mailbox, but only you have the right key to open it.

One important note about public key cryptography, is that if an adversary has intercepted a message, and has access to the encryption function (here a reasonable assumption), then he must not be able to apply brute force encryption to the message space to find a corresponding ciphertext. Such an attack is commonly called a chosen plaintext attack.

An obvious example of attempt at a public key cryptosystem, is to modify the old shift variant and give the dead horse a few more hits with the metaphorical paddle.

Definition 3.3. (Affine cipher) Let $N \in \mathbb{N}$. The affine cipher algorithms are the functions

1. $f_E(P) = aP + b \bmod N$,
2. $f_D(C) = a'C - b' \bmod N$.

If we ignore all the aforementioned problems with frequency analysis and shifting as a enciphering method in general, then this would be a good first stab at a public key cryptosystem. The key idea here, is that it is not immediately clear what the secret keys (the parameters of the decrypting function) would be from just seeing the encryption function. Unfortunately, it does not require a considerable amount of work to be of any use [3], as if $P = f_D(f_E(P))$ then $a'C + b' = a'(aP + b) + b' = P$, and so we require $a' = a^{-1}$ and $b' = -a^{-1}b$. Obtaining the modular inverse of a is done by using the Extended Euclidean Algorithm in $\mathcal{O}((\log(N))^2)$ bit operations. Thus, an adversary can also do this in polynomial time and we deduce that this public key cryptosystem is broken under Kerchoff's principle.

Let us look at this implicit requirement more closely. We want to create a method from which we can only go back if we have the key, or if not, have to subject ourself to a calculation that scales infeasibly with input. Such a function is called a trapdoor function.

Definition 3.4. A function $f : A \longrightarrow B$ is said to be a trapdoor function if f, f^{-1} can be efficiently computed with some extra information (called the trapdoor), but f^{-1} is infeasible to compute without this extra information.

Note, that no trapdoor functions have yet to be proven to exist, but it is widely believed that the encryption function for RSA (in the next section) has this property. A related concept is a one-way function.

Definition 3.5. A function $f : A \longrightarrow B$ is said to be a one-way function if f can be efficiently computed, but f^{-1} is infeasible to compute even with extra information.

A good candidate for such a function, could be the function $(p, q) \mapsto pq$ that maps primes to its product. This relies on the assumption, that factorizing integers into their prime components, is a hard problem, which it has been believed to be for centuries. Just as an example of the kind of work necessary to factor large integers, the biggest current factored number of the RSA-challenge (a number not of any particular form to allow for faster specialized factoring methods) is RSA-768 consisting of 768 bits [10]. Here a number field sieve was used to filter out obvious non-prime factor candidates, and quoting "We spent half a year on 80 processors on polynomial selection [used to optimize the sieve]. This was about 3% of the main task, the sieving, which was done on many hundreds of machines and took almost two years. On a single core 2.2 GHz AMD Opteron processor with 2 GB RAM, sieving would have taken about fifteen hundred years."

By contrast, the possibly largest number proved to be prime with the popular Elliptic Curve Primality Proving (ECPP) algorithm, is the so called Mills' prime, with a ridiculous $20,562$ digits. This was proved on a cluster of six 2.6 GHz Xeon biprocessors running for 68 days[11].

In summary, we present the briefly discussed types of attacks and corresponding notions of security, along with a few extras. A word of caution here; these definitions are lose translations of much more rigorous statements from complexity science. The security definitions are vast simplifications.

Definition 3.6. (Types of Attacks)

1. A ciphertext only attack (COA) tries to find information about a plaintext by making use a set of ciphertexts.

2. A known plaintext attack (KPA) tries to find information about a plaintext by making use of a set of known correspondences of ciphertexts and plaintexts.

3. A chosen plaintext attack (CPA) tries to find information about a plaintext by making use of the encryption function interactively.

4. A chosen ciphertext attack (CCA) tries to find information about a ciphertext by making use of the decryption function with certain restrictions.

Note that the only difference in (KPA) and (CPA), lies in the ability to generate more ciphertext-plaintext pairs as the process of attacking goes along. For a public key cryptosystem, (CPA) security should really be a requirement, as by definition the encryption function is public. However, this also requires encryption to be probabilistic rather than deterministic, for if not, set phrases and locations might be easily picked out of the ciphertext. One might also wonder at what time could possibly a system be under a (CCA) attack? This is a good question, as this attack model was considered a theoretical cul-de-sac for quite some time, until a sophisticated attack on SSL was demonstrated as possible in 1998 [12].

Definition 3.7. (Security of a Cryptosystem) A cryptographic system is said to be

1. (SS) Semantically secure if no polynomial time adversary can attain significant information about the plaintext of a given ciphertext.
2. (IND) Indistinguishable if any two ciphertexts are indistinguishable to a polynomial time adversary.
3. (NM) Non-malleable if given a ciphertext of a plaintext any adversary cannot construct another ciphertext whose plaintext is meaningfully related to the initial one.
4. (IND-CPA) Indistinguishable under a CPA attack if (IND) holds even with full access to f_E.
5. (IND-CCA1) Indistinguishable under a CCA attack if (IND) holds even with access to f_D prior to obtaining two ciphertexts.
6. (IND-CCA2) Indistinguishable under a CCA attack if (IND) holds even with full access to f_D except for on the two ciphertexts to be distinguished.

Indistinguishable here means that ciphertexts cannot be distinguished consistently with a probability of more than 1/2, by any polynomial time algorithm, and a polynomial time adversary refers to an adversary that can run any polynomial time algorithm.

Note that (1) and (2) are equivalent definitions (in the sense that $(IND{-}CPA) \iff (SS{-}CPA)$ and $(IND-CCAi) \iff (SS-CPAi)$ where the SS definitions are similar) as demonstrated in [9], where other interesting hierarchies and implications of combinations of security notions are also summarized. IND-CCA2 is effectively the new high standard to strive for in modern cryptosystems, and there are actually a few cryptosystems that satisfy this.

3.2.1 Rivest-Shamir-Adleman

We start out with the perhaps most well known mathematical cryptosystem, namely RSA.

Definition 3.8. (RSA) Let p, q be two large distinct prime numbers. We compute $n = p \cdot q$, an element e coprime with $\phi(n)$ and the modular inverse $d = e^{-1} \bmod \phi(n)$. The RSA cipher algorithms are the deterministic functions

$f_E(m) = m^e \bmod n$,

$f_D(c) = c^d \bmod n$.

We can verify its validity as such [4][p138]

Lemma 3.9. *(Validity of RSA)* $f_E = f_D^{-1}$ *in* $\mathbb{Z}_n$.

Proof. We aim to verify that $m^{ed} \equiv (m^e)^d \equiv m \bmod n$ for all $m \in \mathbb{Z}_n$. First let $\gcd(m, n) = 1$. Since $\phi(n) \mid ed - 1$, using Euler's Theorem, we see that $m^{ed-1} = m^{k\phi(n)} \equiv 1 \bmod n$. From this it is clear that $m^{ed} \equiv m \bmod n$.
Now suppose $\gcd(m, n) > 1$, i.e. m contains p or q as a factor. Without loss of generality suppose $m = pk$ with $\gcd(k, q) = 1$. Then $m^{ed} \equiv 0 \bmod p$. Now $\phi(n) = (p-1)(q-1)$ so $q - 1 \mid ed - 1$. Therefore, $m^{ed} \equiv m^{ed-1}m \equiv m \bmod q$, as $\gcd(m, q) = 1$. Put together the two congruences $m^{ed} \equiv 0 \equiv m \bmod p$ and $m^{ed} \equiv m \bmod q$ to get $m^{ed} \equiv m \bmod n$. □

Here n is the public modulus, but it is required that the numbers p, q and $\phi(n)$ are kept secret, because otherwise the private decryption exponent d can be computed via the Euclidean Algorithm. This is RSA in its most basic form, and its security relies on the following two (widely believed assumptions).

1. Factorization is an intractable problem.
2. f_E is a trapdoor function.

It is clear by the above paragraph that inverting f_E is at least as easy as factorizing n, but it also conceivable that it is easier. This is the curse of security proofs.

RSA in this form is deterministic, hence offer no CPA security, and consequently as a public deterministic system, neither offer semantic security. However, there are other methods to couple with to achieve this. A method called Optimal Asymmetric Encryption Padding (OAEP) invented by Mihir Bellare and Phillip Rogaway in 1995 [13] and later refined by Victor Shoup in 2001 [14] have effectively modernized RSA by adding an element of randomness and proving that RSA with this system, is actually IND-CCA2 secure. On should note that OAEP follows a specific scheme that and is intended to work with any trapdoor function, not just RSA's encryption function. However, going into the details of this scheme is a bit too peripheral for this report and we will instead concentrate on some interesting mathematics behind another less known cryptosystem.

3.2.2 Goldwasser-Micali

Finally, we get to play around with an application of quadratic residues. The following cryptosystem was considered the first probabilistic encryption scheme, and it was invented by Shafi Goldwasser and Silvio Micali in 1982, and we will abbreviate it by GM.

Before we dive into it directly, we will have a brief discussion about the Jacobi symbol. For this composite $n = pq$, an integer a is a quadratic residue modulo n if and only if $a \in Q_p \cap Q_q$, i.e. it is a quadratic residue with respect to both p and q.

Definition 3.10. (GM) Let p, q be two large distinct prime numbers, and x a quadratic non-residue modulo both p, q. The public key is (x, n), the private key is (p, q), and the system operates bitwise via the following probabilistic cipher algorithms on a message $m = m_1 \cdots m_k$ here decomposed into k bits.

$f_E(m_i) = r^2 x^{m_i} \bmod n$, for $m < n$ and a random $r_i \in \mathbb{Z}_n^*$.

$$f_D(c_i) = \begin{cases} 0 & \text{if } c_i \in Q_n \\ 1 & \text{if } c_i \in \overline{Q_n} \end{cases}$$

The definition of x means that the Jacobi symbol $\left(\frac{x}{n}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right) = (-1)^2 = 1$. One way to find such x is to restrict the primes to those satisfying $p \equiv q \equiv 3 \bmod 4$, as then $x = -1$ would satisfy this by the fourth property of the Legendre symbol.

Lemma 3.11. *(Validity of GM) For all $m_i \in \mathbb{Z}_2$ we have $f_D(f_E(m_i)) = m_i$.*

Proof. Homemade. Suppose $r^2 x^{m_i} \in Q_n$. Then $x^{m_i} \in Q_n$, and by the property of x, we find

$$\left(\frac{x^{m_i}}{p}\right)\left(\frac{x^{m_i}}{q}\right) = \left(\frac{x}{p}\right)^{m_i}\left(\frac{x}{q}\right)^{m_i} = (-1)^{m_i}(-1)^{m_i}.$$

By assumption, both these factors must be 1, so m_i is even and $\equiv 0 \bmod 2$. The other way is similar. □

Using the private factorization of $n = pq$, one can quickly determine whether each ciphertext is a quadratic residue, and in fact, we will describe an algorithm to do so in the next chapter. However, when the factorization is unknown this is widely believed to be a hard problem, and we will state it as one of the two assumptions on which the system relies.

1. Factorization is an intractable problem.
2. Determining whether an integer a is in Q_n or $\overline{Q_n}$ for $n = pq$ when p, q is not known is an intractable problem.

The latter is in fact called the quadratic residuosity problem (QRP). In the original paper [20], a theorem by Rabin states that if for a $1/\log n$ fraction of the quadratic residues q mod n one could find a square root of q, then one could construct a probabilistic algorithm to factor n in polynomial time.

This of course speaks about actually finding the square roots, whereas determining Legendre symbols, just shows whether or not such a root exists. However, this not a very difficult problem. In fact, if we know the factorization, and by sticking to the convenient option of letting p and q be $\equiv 3 \bmod 4$, we can actually just plug into the formula $x \equiv a^{(p+1)/4} \bmod p$, do the same mod q, and use CRT. The general method (for arbitrary primes) is not particulary hard either [3, pp48-50]. Note that probabilistic here means that the algorithm has a small chance of failing, but this result it still so significant it is believed that the two assumptions above are of equivalent.

Furthermore, as this system features probabilistic encryption, it satisfies the SS security notion that was actually proposed by the authors in the original paper. It also satisfies the somewhat different notion of homomorphic encryption.

Definition 3.12. Encryption in a public key cryptosystem is said to have homomorphic encryption if there exists a homomorphism $\Gamma : \mathcal{P} \longrightarrow \mathcal{C}$ in terms of some binary operations $\circ_{\mathcal{P}}, \circ_{\mathcal{C}}$.

GM has this property, as given two message bits m_1 and m_2, the products of the $f_E(m_i)$ satisfy

$$f_E(m_1)f_E(m_2) = (r_1^2 x^{m_1})(r_2^2 x^{m_2}) = (r_1 r_2)^2 x^{m_1+m_2} = f_E(m_1 \oplus m_2).$$

Note that unpadded RSA clearly satisfies this as well, but the stronger RSA-OAEP does not as IND-CCA2 security is equivalent to NM-CCA2 [9]. A homomorphic system is by definition malleable.

Malleability is often thought of as an undesirable property, as theoretically, this could be used to flip bits used in money transfers to change the amount of pounds transferred using a so called a man in the middle attack. However, when the malleability arises from a homomorphism, it could be useful for certain esoteric applications like electronic voting. See [21] for more information on this.

Unfortunately, the GM cryptosystem suffer from a couple of serious ailments. Each individual bit of the message to be encrypted, will be mapped an arbitrary element in $\mathbb{Z}_n$. This causes the ciphertext to be several orders of magnitude larger than the plaintext. This is called message expansion, and is considered a serious inefficiency trait.
Several others systems have since been invented to try to fix this problem. We will go one of these that have gained quite a bit of popularly in recent years.

3.2.3 Paillier

The Paillier cryptosystem was invented in 1999 by Pascal Paillier. This system uses ideas from a wide range of cryptosystems including RSA and GM, but also other probabilistic systems like ElGamal . It uses a typical RSA style modulus $n = pq$, the product of two large primes. Whereas GM was a probabilistic system based on quadratic residues, Paillier is a probabilistic system based on nth residues mod n^2.

Definition 3.13. An integer z is said to be an nth residue mod n^2 if there exists $y \in \mathbb{Z}_{n^2}^*$ such that $z \equiv y^n \bmod n^2$.

The main idea, is that it is hard to determine whether an arbitrary element in $\mathbb{Z}_{n^2}^*$ is an nth residue mod n^2 without the underlying factorization. This is called the decisional composite residuosity assumption (DCRA), and Paillier's original paper clearly explains the different systems and attempts that led to this idea [5]. We will follow this paper almost entirely in this section, but also resort to the occasional wiki help - and consequent double checking - to fill in some of the gaps. To study this system we need to include some more number theoretic results for the special case of $n = pq$. First on the list is Carmichael's theorem, where we improvise a quick proof for this case.

Theorem 3.14. *(Carmichael's Theorem) Let ω, n be coprime, and $\lambda = lcm(p-1, q-1)$. Then*

1. $\omega^\lambda \equiv 1 \bmod n$
2. $\omega^{n\lambda} \equiv 1 \bmod n^2$

Proof. (1) It's clear that $\omega^{p-1} \equiv 1 \bmod p$ and $\omega^{q-1} \equiv 1 \bmod q$. From this we see that $\omega^\lambda - 1$ is a common multiple of p and q, so $\omega^\lambda \equiv 1 \bmod n$.
(2) Using (1) we know that $\omega^\lambda = 1 + kn$, for some $k \in \mathbb{Z}$, so

$$\omega^{n\lambda} = (1+kn)^n \equiv 1 + \binom{n}{1} kn \equiv 1 \bmod n^2,$$

by reducing the binomial expansion mod n^2. □

To continue, we need to investigate a little bit about the structure of nth residues, in particular the roots of them.

Proposition 3.15. *Let y be an nth residue mod n^2 and $\omega \in \mathbb{Z}_{n^2}^*$ such that $\omega^n \equiv y \bmod n^2$. Then the nth roots of y are the elements $\{(1+xn)\omega \bmod n^2 : x \in \mathbb{Z}_n\}$, and there is only one such root smaller than n.*

Proof. Since $(1+n)^x \equiv 1 + xn \bmod n^2$ it is clear that these elements all satisfy $(1+xn)^n \equiv 1 \bmod n$. We will skip the proof that only these elements satisfy $x^n \equiv 1 \bmod n^2$.
We now find the smallest such solution root. First note that $\omega \in \mathbb{Z}_{n^2}^* \Rightarrow \omega \bmod n \in \mathbb{Z}_n^*$. Hence, we can write $\omega = a + bn$ where $a \in \mathbb{Z}_n^*$, i.e. we can also find $a^{-1} \in \mathbb{Z}_n^*$. The desired root is the root with $x \equiv -(a^{-1})b \bmod n$ as then

$$(1+xn)\omega = (1+xn)(a+bn) \equiv a + (b+ax)n = a + (b - baa^{-1})n = a \bmod n^2.$$

Since the inverse of a is unique, so is this solution. □

Definition 3.16. We define ε_g to be the map $\mathbb{Z}_n \times \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_{n^2}^*$ defined by $(x, y) \mapsto g^x y^n \bmod n^2$.

This is going to operate as the encryption function in the Paillier cryptosystem.

Definition 3.17. Let $\mathcal{B}_\alpha$ denote the subset of $\mathbb{Z}_{n^2}^*$ containing all elements of order $n\alpha$, and $\mathcal{B}$ denote the union of the $\mathcal{B}_\alpha$ for $\alpha \in \{1, \ldots, \lambda\}$

The number g is called the base, and when the base has one of the allowed orders, we get the following very interesting relation.

Lemma 3.18. *If $g \in \mathcal{B}$ then ϵ_g is a bijection.*

Proof. First note that $\phi(n^2) = \phi(p^2)\phi(q^2) = p^2q^2\left(1-\frac{1}{p}\right)\left(1-\frac{1}{q}\right) = n\phi(n)$ using the proven properties of the Euler function. Consequently, by the very definition of ϕ, we see that the domain and range have the same cardinality, so it suffices to show that $\mathcal{B}_g$ is injective. Suppose $g^{x_1}y_1^n \equiv g^{x_2}y_2^n \bmod n^2$. Then

$$g^{x_2-x_1}\left(\frac{y_2}{y_1}\right)^n \equiv 1 \bmod n^2, \tag{3.1}$$

so we can invoke Carmichael's theorem to obtain $g^{\lambda(x_2-x_1)} \equiv 1 \bmod n^2$ by taking λ'th powers on both sides. Now $\lambda(x_2 - x_1)$ must be a multiple of the order of g, and $g \in \mathcal{B}$, so $n \mid \lambda(x_2 - x_1)$. But $\gcd(\lambda, n) = 1$ as neither $p-1$ nor $q-1$ can be a factor of n, so we must in fact have $n \mid (x_2 - x_1)$, i.e. $x_2 \equiv x_1 \bmod n$. This also forces $\left(\frac{y_2}{y_1}\right)^n \equiv 1 \bmod n^2$ by 3.1, with the obvious solution $y_2 \equiv y_1 \bmod n$, and this is the only such solution by 3.15. □

Thus, it makes sense to define the class number to be the exponent of the base g.

Definition 3.19. (Class of ω) For $g \in \mathcal{B}$ and $\omega \in \mathbb{Z}_{n^2}^*$ we define the nth residue class of ω to be the unique integer $x \in \mathbb{Z}_n$ for which there exists $y \in \mathbb{Z}_n^*$ such that $\epsilon_g(g, y) = \omega$. Denote this $[\![\omega]\!]_g$.

Proposition 3.20. *(Class Properties) Let $g, g_i \in \mathcal{B}$. Then $\forall \omega, \omega_i \in \mathbb{Z}_{n^2}^*$*

1. $[\![\omega]\!]_g = 0 \Leftrightarrow \omega$ *is an n-th residue mod n^2.*
2. $[\![\omega_1\omega_2]\!]_g = [\![\omega_1]\!]_g + [\![\omega_2]\!]_g$
3. $[\![\omega]\!]_{g_1} = [\![\omega]\!]_{g_2}[\![g_2]\!]_{g_1}$.
4. $[\![g_1]\!]_{g_2} = [\![g_2]\!]_{g_1}^{-1}$.

Proof. (1) If $[\![\omega]\!]_g = 0$ then $\omega \equiv g^0y^n \bmod n^2$ which implies $\omega \equiv y^n$. Conversely, suppose ω is an n-th residue mod n^2, i.e. $\omega \equiv y^n \bmod n^2$. Since $y \in \mathbb{Z}_{n^2}^*$, it must be of the form $y = a + bn$ with $a \in \mathbb{Z}_n$. Then $\omega = y^n = (a+bn)^n = a^n \bmod n^2 = g^0a^n$, so $[\![\omega]\!]_g = 0$.
(2) Given $\omega_1 = g^xy^n \bmod n^2$ and $\omega_2 = g^ab^n \bmod n^2$, then $\omega_1\omega_2 = g^{a+x}(yb)^n$. Now we can write $a + x = (a + x \bmod n) + cn$, so finally

$$\omega_1\omega_2 = g^{a+x \bmod n}(ybg^c)^n \bmod n^2.$$

Evaluating class numbers of these values yield the result.
(3) Given $\omega_1 = g_1^ab^n \bmod n^2$ and $g_2 = g_1^cd^n \bmod n^2$, we can write $\omega = g_1^{cx}(d^xy)^n \bmod n^2$. This implies $[\![\omega]\!]_{g_1} = cx$, $[\![\omega]\!]_{g_2} = x$ and $[\![g_2]\!]_{g_1} = c$.
(4) Set $\omega = g_1$ in (3) to get $[\![g_1]\!]_{g_2}[\![g_2]\!]_{g_1} = [\![g_1]\!]_{g_1} = 1 \bmod n$. □

Note (2) is saying that the map $(\mathbb{Z}_{n^2}^*, \times) \longrightarrow (\mathbb{Z}_n, +)$, $\omega \mapsto [\![\omega]\!]_g$ is a group homomorphism (in fact this also causes homomorphic encryption). Property (3) says that if you can evaluate the class number for one particular base g, then you can evaluate it for any other base. This property is referred to as random self-reducibility, and a physical interpretation of this is that any choice of g is as good (as hard for an adversary) as any other in terms of g as a public parameter.

These abstract properties can be a bit overwhelming at first, but essentially they are statements about the equivalence classes of residues. If we were talking about quadratic residues, then there would only be two residue classes, namely Q_n and $\overline{Q}_n$. For nth residues there are n such classes (and only one of them corresponds to the case ω is an nth residue) and we can actually show the correspondence of the class numbers in a more clear way.

Lemma 3.21. *ω_1 and ω_2 lie in the same residue class if and only if $\omega_1\omega_2^{-1}$ is an nth residue mod n^2*

Proof. Homemade proof. Let $g \in \mathcal{B}$. Translating this statement in the language above we need to show $[\![\omega_1\omega_2^{-1}]\!]_g = 0 \iff [\![\omega_1]\!]_g = [\![\omega_2]\!]_g$. First note that if $\omega = g^r y^n \bmod n^2$ then $\omega^{-1} = g^{-r}(y^{-1})^n \bmod n^2$. Thus $[\![\omega^{-1}]\!]_g = -[\![\omega]\!]_g$. Now $[\![\omega_1]\!]_g = [\![\omega_2]\!]_g \iff [\![\omega_1]\!]_g - [\![\omega_2]\!]_g = 0$ and this last expression is $[\![\omega_1]\!]_g + [\![\omega_2^{-1}]\!]_g = [\![\omega_1\omega_2^{-1}]\!]_g$. □

Definition 3.22. We define the subset $S_n = \{u < n^2 : u \equiv 1 \bmod n\}$ of $\mathbb{Z}_{n^2}^*$, and on it, a map $L : S_n \longrightarrow \mathbb{Z}_n$, $u \mapsto \frac{u-1}{n}$.

It's clear that S_n is in fact a subgroup, and that the L map is well defined on it. More interestingly, this map has the following abstract property relating what we have worked on so far.

Lemma 3.23. *$\forall \omega \in \mathbb{Z}_{n^2}^*$ $L(\omega^\lambda \bmod n^2) = \lambda[\![\omega]\!]_{1+n}$*

Proof. It's clear that $1+n \in \mathcal{B}$ so by the bijectivity of ε_g there exists a unique pair $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ such that $\omega = (1+n)^a b^n \bmod n^2$. Consequently $\omega^\lambda \equiv (1+n)^{a\lambda} \equiv 1 + an\lambda \bmod n^2$, and thus

$$L(\omega^\lambda \bmod n^2 = a\lambda = [\![\omega]\!]_{1+n}\lambda.$$

□

We are almost done. But before we state what our decryption function is, we first state the definition of the full system, so that the calculation makes sense.

Definition 3.24. (Paillier) Let p, q be two large distinct prime numbers, and $g \in \mathcal{B}$ be an arbitrary base for $n = pq$, and $\lambda = lcm(p-1, q-1)$. The Paillier cryptosystem has public key (g, n) and private key λ, and operates via the following probabilistic cipher algorithms.

$f_E(m) = g^m c^r$, for $m < n$ and a random $r < n$.

$f_D(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)}$ for $c < n^2$.

Lemma 3.25. *$f_D(f_E(m)) = m$ on $\mathbb{Z}_n$*

Proof. Using property (4) of the class number we have $[\![g]\!]_{1+n} = [\![_{1+n}]\!]_g^{-1} \bmod n$. Now λ is coprime to n and hence invertible, so in fact the whole $L(g^\lambda \bmod n^2) = \lambda[\![g]\!]_{1+n}$ is invertible. Thus the decryption function is well defined and it evaluates at

$$\frac{L(\omega^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} = \frac{\lambda[\![\omega]\!]_{1+n}}{\lambda[\![g]\!]_{n+1}} = [\![\omega]\!]_{1+n}[\![1+n]\!]_g = [\![\omega]\!]_g \bmod n.$$

□

Clearly, knowledge of the factorization of n can make you break the system, as does being able to evaluate class numbers efficiently. These are our only two assumptions however.

1. Factorization is an intractable problem.
2. Determining the class number of an element is an intractable problem without knowing p, q.

Under these assumptions the system satisfies IND-CPA security [5, p7]. The homomorphic property means that we will not be able to enjoy the higher IND-CCA2 tier of security. This is because of the aforementioned equivalence between the two notions IND-CCA2 and NM-CCA2, and because Paillier is a homomorphic scheme.

Paillier also benefits from the interesting self-blinding property. In particular, two ciphertexts can be publicity changed into another without changing its plaintext. This is also a desirable property of voting and money transactions, where one would like methods of ensuring validity of such an action, but not implicity disclosing the identity of the person to whom it belongs. To see that this property holds, consider a random $r < n$. Then we can scale the ciphertext by r^n in $\mathbb{Z}_{n^2}$ as $f_D(f_E(m)r^n \bmod n^2) = m$ by noting that the scaling does not change the base exponent, and hence not the class number.

3.3 Key Exchange

As a more practical segment about the usefulness of public key cryptography, consider the following. Two parties wanting to communicate, can with one of the above ciphers both create a public key, send these to each other and communicate securely under simple assumptions. In practice, public key ciphers are often slower than their symmetric counterparts [3]. As a consequence, symmetric cihpers are often used after having simply received a symmetric key securely by another method. One obvious such method, is to send it via a public key system, but there is another mathematically pleasing way, of doing so.

3.3.1 Diffie-Hellman

Definition 3.26. (DH Key Exchange) Let g be a generator of $\mathbb{F}_q$, and A and B denote the two communicating parties. The key exchange procedure works as follows,

A finds a random element $a \in \mathbb{F}_q^*$, computes g^a and sends (g, g^a, q) to B.

B finds a random element $a \in \mathbb{F}_q^*$, computes g^b and sends (g^b) back to A.

Both A and B can now compute the secret key $k := g^{ab}$

We choose g to be a generator, as this allows g to run through any element of $\mathbb{F}_q^*$, and ensures more randomness of the key. One is tempted to think that, surely it is possible to derive the key, when almost all the information have crossed paths like this, and of course, it is. The problem is that it is really hard to do so.

Definition 3.27. (Diffie-Hellmann assumption) In $\mathbb{F}_q$, it is computationally infeasible to compute g^{ab} knowing only g^a and g^b.

It is widely believed that this is as hard as computing discrete logarithms (i.e. finding a from g^a in a finite field) [3, pp98-99]. Clearly, a method for solving discrete logs efficiently would also

violate the DH assumption.

A successful implementation of this ensures that in many secure encounters with cryptography online, you will have generated keys you are not even aware of. The system can essentially run itself between two parts that support this protocol.

3.4 Extra Considerations

In practice, there are more things to consider, than what cipher to use, and what key length to use for this cipher. A big problem on the internet is to prove that the person you are talking to, or sending your information to, really is who or what he claims to be. Another, is the possibility of tampering.

3.4.1 Authenticity & Integrity

If a message was intercepted, tampered with, and then forwarded to you. Determining this, is determining message authenticity, and is in practice solved by a message authentication code (MAC). This also solves the other problem of integrity.

One method is utilizing a type of one-way function H called a hash function, that is applied to the plaintext, and sent along with the message in the form of the pair $(c, h) = (f_{E_A}(m), f_{D_B}(H(m)))$. Here the message is encrypted with the public key of the recipient, but the hash is *decrypted* (we say the message is signed) with the private key of the sender. In certain cryptosystems like RSA (although the original is not the best for this particular implementation [25]) $f_E = f_D^{-1}$ and encryption of a decrypted message, is inherently just a different form of decrypting encryption. The receiver can verify (with the senders public key) that $f_{E_B}(h) = H(f_{D_A}(c))$, i.e. the hash that was encrypted with the senders private key is the same as hash from the decrypted message.

This works because an interceptor can change the first parameter and generate a new hash value accordingly (as the senders public key, and the hash function is assumed to be known), but they do not possess the sender's private key and the hashes will disagree in the verification stage.

The soundness of this construction notwithstanding, this process is typically verified using a hash function relying on a shared secret key. For with such a system, a man in the middle cannot alter the hash without the secret key, and the extra layer of encryption is redundant. The message is authenticated if and only if the hash corresponds to the hash of the received decrypted message. One typical such construction is the Keyed-Hash Message Authentication Code (HMAC) [7], [27]. In the case of, for instance, an electronic transaction, then the server must prove its authenticity to you. This is commonly done the Secure Sockets Layer protocol. If the server has an SSL certificate issued by a third party (whose identity can be determined by in the browser), the authenticity of this certificate can be verified live by an encryption with the issuers public key. This procedure is similar to the one described above.

Chapter 4

Key Generation

4.1 Pseudo-Random Number Generators

The process of actually implementing a cryptographic system on a computer is an intricate and interesting problem. The seemingly innocuous statement that *"pick a random $r < n$"*, can have widely different interpretations about what is good practice for this. After all, the entire cryptographic model that we assume, that a system can be only be broken after an infeasible amount of computation, only holds if the pool of numbers we are generating from, cannot be exhausted faster than this computation. This is the notion of entropy of pseudo-random number generator (PRNG).

A PRNG is found in every basic programming language, but the very basic implementations of such generators often show significant statistical patterns, that would combined with cryptographic use be a major security flaw. Thankfully, there are more secure solutions made for cryptographic implementation. Thus, it is of the utmost importance that these are used. Unfortunately, this is not always the case [16].

4.1.1 Blum-Blum-Shub

In more recent texts [7] PRNGs are given the same rigorous treatment as you would expect from any other area of cryptography. A good PRNG is defined as one that cannot be distinguished from a truly random source in polynomial time. One example would be the Blum-Blum-Shub algorithm which is defined by $x_{k+1} \equiv x_k^2 \mod n$ for $n = pq$, and the output would be the least significant bit of this calculation. Here we require $p \equiv q \equiv 3 \mod 4$ which ensures that each quadratic residue has exactly one square root that is also a quadratic residue (see [26] for a proof of this). Since determining modular square roots is hard (as discussed in the Goldwasser-Micali section), this PRNG actually has a security proof, related to the hardness of integer factorization. However, we will not go into more details on this.

4.2 Tests for Primality

As we saw in the last chapter, all the cryptosystems we discussed relied on primes for strong security (although this is not always the case). Thus, it is important that we are able to generate new large primes for each encounter. We generally follow [2, pp85-94] for the rest of this chapter.

The typical mental way of determining if a number is prime, is by checking if it has any divisors in the set of prime numbers smaller than $\sqrt{n}$. A simple algorithm for checking this for an integer n, can be written as follows.

```
1    i ← 2;
2    while i^2 ≤ n repeat;
3        if i divides n then return false;
4        i ← i + 1;
5    return true;
```

This cycles through all the divisors (not just primes here for algorithm simplicity) and returns *false* if we found a divisor, or eventually *true* by exhausting the pool of integers. Since we have to expect to exhaust the pool to find a prime, this algorithm runs in $\mathcal{O}(\sqrt{n})$, i.e. exponential time.

The reason for its horrific speed, is that it actually does much more than is required; it reveals the factorization of n. As this is believed to be an intractable problem, we cannot realistically expect such an algorithm to ever operate efficiently. We therefore go through some interesting and unexpected ways, to check if a number is prime in polynomial time.

As an introduction, the first method that perhaps comes to mind is to test whether Fermat's little theorem holds. For if n is prime, then $a^{n-1} \equiv 1 \bmod n$ for all a, so by testing a large amount of elements $a \in \mathbb{Z}_n$ perhaps this would be enough to show n is prime. The process of doing so is called Fermat testing the number, and in most cases this works quite well. For most n it will give you a false positive (called an F-liar) for at most half of the a values [2]. We prove the following.

Lemma 4.1. *Let $n \geq 2$ be an integer. If $a^{n-1} \equiv 1$ for all $a \in \mathbb{Z}_n$, then n is prime.*

Proof. Note that $a^r \equiv 1$ for some $r \geq 1$ implies that a has an inverse, namely a^{r-1}. Thus, if $a^{n-1} \bmod n = 1$ for all non-zero $a \in \mathbb{Z}_n$, then $\mathbb{Z}_n^* = \mathbb{Z}_n \setminus \{0\}$, and so $\phi(n) = n - 1$. □

This means there will always be *some* F-witnesses for an odd composite number n, and the test will succeed. In fact, the $n - 1 - \phi(n)$ elements of $\{a \in \mathbb{Z}_n : \gcd(a, n) > 1\}$ cannot satisfy $a^{n-1} \bmod n = 1$. Unfortunately, this set sparse when n has few prime factors. For example, if $n = pq$ where p and q are distinct primes, then there are exactly $p + q - 2$ such numbers in $\mathbb{Z}_n$.

To make matters worse, there are pathological cases where n is not prime, but one receives a false positive for every number in $\mathbb{Z}_n^*$ (see for instance [3, pp126-129]). Such a number is called a Carmichael number, and for certain such numbers the proportion of invertible elements can be very large. For instance, the Carmichael number $n = 651693055693681$ has $\phi(n)/n > 0.99996$ [2]. For this reason the Fermat test cannot be relied upon in general, without actually testing all the values.

4.2.1 Soloway-Strassen

The perhaps most surprising algorithm is the one developed by Robert M. Solovay and Volker Strassen, based on the theory of quadratic residues. Following [2, pp85-94] in this section, we begin by a quick guide to the quadratic reciprocity law that we proved earlier for the Jacobi symbol. If given an odd integer $n \geq 3$ and an integer a, then we can perform a systematic reduction of Jacobi symbol $\left(\frac{a}{n}\right)$ by the following key steps.

1. If a is not in the interval $\{1, \ldots, n-1\}$ the result is $\left(\frac{a \bmod n}{n}\right)$.
2. If $a = 0$ the result is 0.
3. If $a = 1$ the result is 1.
4. If $4 \mid a$ the result is $\left(\frac{a/4}{n}\right)$.
5. If $2 \mid a$ and $n \in \{1, 7\}$, the result is $\left(\frac{a/2}{n}\right)$.
6. If $2 \mid a$ and $n \in \{3, 5\}$, the result is $-\left(\frac{a/2}{n}\right)$.
7. If ($a > 1$ and) $a \equiv 1$ or $n \equiv 1 \pmod 4$, the result is $\left(\frac{n \bmod a}{a}\right)$
8. If $a \equiv 3$ and $n \equiv 3 \pmod 4$, the result is $-\left(\frac{n \bmod a}{a}\right)$

This procedure can be followed directly, stopping only once the numerator becomes 0 or 1, or we can convert this into a more streamlined computer algorithm following this procedure.

```
Input: Integer a, odd integer n ≥ 3.
Output: (a/n)
Method:
0   b,c,s: integer;
1   b ← a mod n; c ← n; s ← 1;
2   while b ≥ 2 repeat
3       while 4 | b repeat b ← b/4;
4       if 2 | b then
5           if c mod 8 ∈ {3,5} then s ← (−s);
6           b ← b/2;
7       if b = 1 then break;
8       if b mod 4 = c mod 4 = 3 then s ← (−s);
9       (b,c) ← (c mod b, b);
10  return s · b;
```

A quick run through these lines reveal that s keeps track of all the sign changes from the $\left(\frac{2}{p}\right)$ rule in line 5, and the quadratic reciprocity law usage in line 8. On a computer, division, or checking divisibility by 2 or 4, can be done by simply treating the binary representation as a string, and check for, or remove trailing zeroes.

Interestingly, the outer while loop behaves like the Euclidean Algorithm, in that by taking the residues of the opposite inputs, the generated decreasing sequence must terminate in $2\|n\| = \mathcal{O}(\log n)$ steps. Since the only expensive operations found herein is division with remainder, the Jacobi Algorithm also runs in $\mathcal{O}((\log n)^2)$ bit operations, exactly like the Euclidean Algorithm - truly an amazing consequence of the quadratic reciprocity law.

The following trivial lemma, will form the basis for the primality test.

Lemma 4.2. *If p is an odd prime number, then for all $a \in \mathbb{Z}_p$*

$$a^{(p-1)/2} \cdot \left(\frac{a}{p}\right) \ mod\ p = 1. \tag{4.1}$$

Proof. By Euler's Criterion, the two factors are either both 1 or -1. □

Using the contrapositive turnaround of this, we get that if $n \geq 3$ is odd and $a \in \mathbb{Z}_n$ does not satisfy this identity, then n cannot be prime. This leads to the essential question: how likely is it for a composite number to pass this test? To answer this question accurately, we will need a little group theory.

Definition 4.3. Let n be an odd composite number, then $a \in \mathbb{Z}_n$ is called an E-witness for n if the identity 4.1 is satisfied. a is called an E-liar otherwise.

Lemma 4.4. *Let $n \geq 3$ be an odd composite number. Then the set of E-liars for n form a proper subgroup of $\mathbb{Z}_n^*$.*

Proof. If $a \in \mathbb{Z}_n$ an E-liar, then 4.1 holds and $\left(\frac{a}{n}\right) \in \{\pm 1\}$ (i.e. not zero), so

$$1^2 = \left((a^{(n-1)/2}\left(\frac{a}{n}\right)\right)^2 = a^{n-1} = 1 \bmod n,$$

So a is also an F-liar. Consequently, $a^{-1} = a^{n-2}$ so a is an invertible element.
We show the liars form a subgroup. Clearly 1 is an E-liar, so suppose $a, b \in \mathbb{Z}_n^*$ are two E-liars. Then

$$(ab)^{(n-1)/2}\left(\frac{ab}{n}\right) = a^{(n-1)/2}\left(\frac{a}{n}\right) b^{(n-1)/2}\left(\frac{b}{n}\right) = 1 \cdot 1 = 1,$$

so ab is another E-liar. The hard part is showing that the set of E-witnesses in $\mathbb{Z}_n^*$ is non-empty. Consider first the case when n is square free, and we can write $n = mp$ for an odd $p \nmid m$ and an odd $m \geq 3$. We let $b \in \overline{Q}_p$. Now use CRT to obtain a solution a to the congruences $a \equiv b \bmod p$ and $a \equiv 1 \bmod m$. Since this a is not divisible by p and is coprime to m it must actually be in $\mathbb{Z}_n^*$. Now the symbol can be evaluated

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{m}\right) = \left(\frac{b}{p}\right)\left(\frac{1}{m}\right) = (-1) \cdot 1 = -1.$$

On the other and if $a^{(n-1)/2} \equiv -1 \bmod n$ then this would mean $a^{(n-1)/2} \equiv 1 \bmod m$, a contradiction. Thus, 4.1 is violated and a is an E-witness.

In the second case, suppose there exists a prime p such that $p^2 \mid n$. Then if we let $a = (1 + \frac{n}{p})$ it is clear from the binomial theorem that $a^p = 1 \bmod n$ so that a is invertible, and in particular the order of a is p. Thus $a^{n-1} \neq 1 \bmod n$ since $p \nmid n - 1$. On the other hand, the symbol satisfies

$$\left(\frac{a}{n}\right) = \left(\frac{1 + n/p}{n}\right) = \left(\frac{1 + n/p}{p}\right)\left(\frac{1 + n/p}{n/p}\right) = \left(\frac{1}{p}\right)\left(\frac{1}{n/p}\right) = 1,$$

so 4.1 does not hold in this case either.

□

```
Solovay-Strassen Test
1   while (l > 0)
2       Let a be randomly chosen from {2,...,n-2};
3       if (a^((n-1)/2) · (a/n) mod n ≠ 1) return false;
4       l ← l - 1;
4   return true;
```

This returns false if identity 4.1 is not satisfied for some $a \in \mathbb{Z}_n \setminus \{\pm 1\}$, which only happens if we found an E-witness for n. This test uses the above Jacobi algorithm for the symbol, and it is repeated a fixed amount of times l large enough so that we can with reasonable faith believe the true result. Note also that the l does not depend on n and can technically be dropped from the complexity bound below.

Lemma 4.5. *The Solovay-Strassen test runs with complexity* $\mathcal{O}(l(\log n)^3)$ *and outputs* false *when* n *is composite, and* true *when* n *is prime or* n *is composite with the latter case happening with a probability smaller than* 2^{-l}.

Proof. Since the E-liars for n form a proper subgroup of $\mathbb{Z}_n^*$, this means that the probability of getting a false positive is less than $(\phi(n)/2)/\phi(n) = 1/2$. Thus, running the algorithm for l iterations ensure a failure probability $< 2^{-l}$, and depending only on the Jacobi symbol algorithm and fast modular exponentiation, this test can be achieved in $\mathcal{O}(l(\log n)^3)$ bit operations. □

4.2.2 Miller-Rabin

The following test is largely considered the successor of Soloway-Strassen, and is it relies heavily on a special case of the very simple Lemma 2.24, namely; $a^2 \equiv 1 \bmod p$ if and only if $a \equiv \pm 1$. This means, that if we find a non-trivial square root of unity modulo n, then n is certainly composite.

What leads to the great discovery here, is the fact that we are only interested in an odd $n \geq 3$, so we decompose $n - 1 = u2^k$ for some odd u, and some $k > 0$. With this decomposition, we see that $a^{n-1} \equiv (a^u)^{2^k} \bmod n$. Now consider the possibility that n is prime. By generating the numbers $\{(a^u)^{2^i}\}_{i=0}^k$, we should expect the last value to be 1, as otherwise a is an F-liar for n. More importantly, if the sequence does not start with a 1, then the first 1 should be immediately preceded by $-1 \equiv n - 1$, as otherwise there exists a non-trivial square root of unity. Putting these two observations together, it suffices to check that if the sequence does not start at 1, then it must contain $n - 1$ at some point before k. This is going to be the condition to check for, and the formal definition goes as follows.

Definition 4.6. Let $n \geq 3$ be odd, and write $n - 1 = u2^k$, for some odd u, and some $k > 0$. A number $a \in \{1, \ldots, n-1\}$ is called an A-witness for n if $a^u \neq 1 \bmod n$ and $a^{u2^i} \neq n - 1 \bmod n$ for all $i \in \{0, \ldots, k\}$. If n is composite and a is not an A-witness for n, then a is called an A-liar for n.

Like for Soloway-Strassen, we create a more complete version of the algorithm from [2] to perform the test.

```
Miller-Rabin Test
1   Decompose n − 1 = u2^k;
2   while (l > 0)
3       Let a be randomly chosen from {2, ..., n − 2};
4       b ← a^u mod n;
5       if b ∉ {1, n − 1} then
6           while (k > 1)
7               b ← b^2 mod n;
8               if b = n − 1 then break;
9               if b = 1 then return false;
10              k ← k − 1;
```

```
11          if k = 1 return false;
12      l ← l − 1;
13  return true;
```

The decomposition in line 1 can, as noted earlier, be performed quickly by slashing trailing zeroes in the binary expansion of $n-1$. The test at line 5 investigates whether the initial sequence value will lead to any information at all. If it does, then we enter the squaring loop, and if at any point in the sequence we get $n-1$ before a 1, we drop out of the squaring loop as a has passed the test. However, if we obtain a 1 first, then there exists a non-trivial square root of 1, so a is an A-witness for n. If we run through the entire squaring loop, and have not gotten $n-1$ (or found a witness) before k has reached 1, then the remain cases indicate that n is composite. To see this, simply consider the possibilities. If the last iterate is not 1, a is an F-witness, on the other hand, if the last iterate is 1, then $a^{u2^{k-1}}$ is a non-trivial square root of 1, and a is an A-witness.

This algorithm also runs through l different of values a to test for, and the belief is that having passed the test for all the l random values, should strongly indicate that n is prime. The question again, is how strong is the error bound. By checking both the Fermat condition, and the square root condition, we should expect a result as least as strong as Soloway-Strassen. It turns out, that the chance of getting a false positive is actually less than 4^{-1}. Regrettably, the proof of this is convoluted, and does not contribute much to this section. One version can be found in [3, pp130-134] however, and why yes, it does require *5* pages.

An interesting fact about Miller-Rabin, is that it was originally constructed by Gary Miller as a deterministic primality test - now referred to as Miller's test. The difference in the original version was that under the assumption of the General Riemann Hypothesis (GRH) [2], there is an explicit upper bound on the smallest A-witness at $2(\log n)^2$. This means, that if we run the test for all $a \in \{2, \ldots, 2(\log n)^2\}$, then getting the result true, means that either n is prime, or that GRH is false. This combined with an error bound twice the size of Soloway-Strassen, is why Miller-Rabin is one of the most trusted primality testing algorithms in cryptography (see for instance [25]).

Miller's test was later converted into a probabilistic version by Michael O. Rabin, and it is essentially much faster. Instead of going all the way up to $2(\log n)^2$, we can simply stop at 27, a number which will ensure a failure probability smaller than $4^{-27} \approx 10^{-18}$. This is one in a billion billion, so if we pick two completely arbitrary inhabitants of China, the error probability is smaller than you actually guessing these two. This should be sufficient for most practical purposes.

Miller-Rabin runs in $\mathcal{O}(l(\log n)^3)$ bit operations by noting that the most expensive operation is fast modular exponentiation in line 4, with cost $\mathcal{O}((\log n)^3)$, and that squaring in the k loop is performed for $k \leq \log n$ times with a cost of $\mathcal{O}(k(\log n)^2) = \mathcal{O}((\log n)^3)$. The conditionally deterministic version, thus runs in $\mathcal{O}((\log n)^5)$.

There is also a sequence dedicated to this at [24], which lists the smallest odd number for which Miller-Rabin primality test on bases $\leq n$th prime fails. As expected, it is a rapidly growing sequence.

As a summary to this section, we present the following theorem.

Theorem 4.7. *The Miller-Rabin test runs with complexity* $\mathcal{O}(l(\log n)^3)$ *and outputs* false *when* n *is composite, and* true *when* n *is prime or* n *is composite with the latter case happening with a*

probability smaller than 4^{-l}.

4.2.3 Agrawal-Kayal-Saxena

A newcomer in the field of primality testing is the 2002 AKS algorithm. This is much more complex to discuss, so we will simply note one of the key lemmas being used for testing; a familiar lemma usually only seen in one direction. Here it swings both ways.

Lemma 4.8. *Let* $n \geq 2$*, and* $a < n$ *be two coprime integers. Then*
n *is prime* $\iff$ $(X + a)^n = X^n + a$ *in* $\mathbb{Z}_n[X]$

Proof. Using the Binomial Theorem, we know

$$(X + a)^n = X^n + \sum_{0<i<n} \binom{n}{i} a^i X^{n-i} + a^n.$$

($\Rightarrow$) Suppose n is prime. Then for $1 \leq i \leq n-1$ the binomial coefficient

$$\binom{n}{i} = \frac{n(n-1)\cdots(n-i+1)}{i!}$$

is divisible by n as the numerator is, but not the denominator. So in $\mathbb{Z}_n[X]$ all these coefficients vanish, and so there are no cross terms. Fermat's little finishes this implication by showing $a^n = a$.
($\Leftarrow$) Suppose n is not a prime number. We can then find a factor $p < n$ of n and some $s \in \mathbb{N}$ such that $p^s \mid n$, but $p^{s+1} \nmid n$. We will show that the coefficient of X^{n-p} does not vanish, contradicting the congruence. This coefficient is

$$\binom{n}{p} a^p = \frac{n(n-1)\cdots(n-p+1)}{p!} a^p.$$

In the numerator, n is divisible by p^s, so the other factors must be relatively prime to p. The denominator is divisibly by p, so $\binom{n}{p}$ is not divisibly by p^s. Now a and p were coprime, so p does not divide a^p. Thus the coefficient for X^{n-p} is not divisibly by p^s and is thus not a multiple of n. □

The significance of this algorithm, is that it proves unconditionally, that a prime status can be deterministically proven in polynomial time. Unfortunately, with simple multiplication methods, the algorithm has only a proven complexity bound of $\mathcal{O}((\log n)^{16.5})$. However, this has been lowered in recent years to $\mathcal{O}((\log n)^{6+\epsilon})$ [28], but it is unclear at what point this starts to become efficient.

4.3 Last Words

Having gone through quite a few methods and ideas behind what today constitutes modern cryptography, we hope this text has been interesting and have piqued your interest in cryptography accordingly. This is by no means an exhaustive list, but it goes through what we consider to be the most important ideas. We also believe that the recent Paillier cryptosystem can help provide greater security and anonymity in electronic encryption, in particular in voting applications.

These systems have helped simplify our lives, and go about almost without any notice at all. In that sense, it is a perfect scientific invention, and it should not be taken for granted.

Bibliography

[1] Neal Koblitz, *The uneasy relationship between mathematics and cryptography*, (2007), Notices of the Amer. Math. Society, Vol. 54, 972-979

[2] Martin Dietzfelbinger, *Primality Testing in Polynomial Time*, (2004)

[3] Neal Koblitz, *A Course in Number Theory and Cryptography, 2nd Edition*, (1994)

[4] Ramanujachary Kumanduri and Cristina Romero, *Number Theory with Computer Applications*, (1998)

[5] Pascal Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, Eurocrypt (1999), pp223-238

[6] Song Y. Yan, *Number Theory for Computing*, (2002), 2nd Ed, pp173-202

[7] Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, (2007)

[8] Albrecht Beutelspacher, *Cryptology*, (1994)

[9] Yodai Watanabe, Junji Shikata, and Hideki Imai, *Equivalence between Semantic Security and Indistinguishability against Chosen Ciphertext Attacks*, (2003)

[10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thom, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann *Factorization of a 768-bit RSA modulus* (2010)

[11] Chris K. Caldwell, *Determining Mills' Constant and a Note on Honaker's Problem*, (2005), pp5-6

[12] Daniel Bleichenbacher, *Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS*, (1998)

[13] M. Bellare, P. Rogaway, emphOptimal Asymmetric Encryption - How to encrypt with RSA., Eurocrypt (1994)

[14] Victor Shoup, *OAEP Reconsidered*, (2001)

[15] William Judson LeVeque, *Fundamentals of number theory*, (1996), pp90-91

[16] Nate Lawson, *Crypto Strikes Back!*, (2009), Google Tech Talks, `http://www.youtube.com/user/GoogleTechTalks#p/search/0/ySQl0NhW1J0`

[17] Neal Koblitz and Alfred Menezes, *Another Look at Generic Groups*, (2006)

[18] Alan Parker, *Algorithms and Data Structures in C++*, (1993), p241

[19] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language. J. Symbolic Comput.*, (1997), `http://magma.maths.usyd.edu.au/magma/Features/node86.html`

[20] Goldwasser, S. and Micali, S. *Probabilistic Encryption. Special issue of Journal of Computer and Systems Sciences*, Vol. 28, No. 2, pp270-299, (1984)

[21] Ben Adida, *Theory and Practice of Cryptygraphy: Verifying Elections with Cryptography*, (2007), Google Tech Talks, `http://www.youtube.com/watch?v=ZDnShu5V99s`

[22] Damien Stehl and Paul Zimmermann, *A Binary Recursive Gcd Algorithm*, (2004)

[23] A.O.L. Atkin and F. Morain*Elliptic Curve Primality Proving*, (1991)

[24] Sloane, N. J. A., *Sequence A014233 in "The On-Line Encyclopedia of Integer Sequences."*

[25] FIPS PUB 186-3, *Digital Signature Standard (DSS)*, (2009)

[26] Leonore Blum, Manuel Blum and Michael Shub, *Comparison of two Pseudo-Random Number Generators*, (1986)

[27] FIPS PUB 198-1, *The Keyed-Hash Message Authentication Code*, (2008)

[28] R. Crandall and J. Papadopoulos, *On the implementation of AKS-class primality tests*, (2003)

[29] SSL 3.0 specification, (1996), `http://www.freesoft.org/CIE/Topics/ssl-draft/3-SPEC.HTM`