10 Public Key Cryptography: RSA

10.1 Introduction

The idea behind a public-key system is that it might be possible to find a cryptosystem where it is computationally infeasible to determine $d_K$ even if $e_K$ is known. If so, then the encryption rule $e_K$ could be made public by publishing it in a directory. A diagram of a public key cryptosystem was given in Section 1 where we contrasted private and public key cryptosystems.

Private-key cryptosystems have significant problems related to the sharing of the private key, in particular, the key must be securely distributed to the communicating parties. Moreover, if there are $n$ parties, all wanting to communicate securely with a central party, then $n$ different keys must be securely distributed. Public-key cryptosystems overcome these problems, but public-key cryptosystems tend to be significantly slower than private-key systems.

In practice, hybrid systems are often used in which a slower public-key system is used for infrequently occurring operations such as key establishment and the creation of digital signatures, and a faster private-key system is used for frequently occurring operations such as bulk data encryption.

- Public-key/two-key/asymmetric cryptography involves the use of two keys:
  1. a public-key, which may be known by anybody, and can be used to encrypt messages, and verify signatures;
  2. a private-key, known only to the recipient, used to decrypt messages, and create signatures.

- Anyone knowing the public key can encrypt messages or verify signatures, but cannot decrypt messages or create signatures, counter-intuitive though this may seem.

- It works by the clever use of number theory problems that are easy (P type) one way but hard (NP type) the other way, eg exponentiation vs logs, multiplication vs factoring. If some extra information is known, then the hard problem becomes easy.

We will focus on two public key cryptography schemes: RSA (which is based on the difficulty of integer factorisation) and ElGamal (which is based on the difficulty of calculating a discrete logarithm).

10.2 Mathematical background

Both schemes rely heavily on the properties of prime numbers. Here, we give some necessary background.

Remark 10.1 Euclid showed that there are infinitely many prime numbers.

Remark 10.2 The Prime Number Theorem says that the number $\pi(n)$ of primes less than or equal to $n$ is asymptotically equal to $\frac{n}{\ln n}$, as $n$ gets large.

Theorem 10.3 Bertrand’s Postulate: For any integer $n$ there is always a prime number between $n + 1$ and $2n$. 
Remark 10.4 In setting up an RSA or ElGamal cryptosystem, it is necessary to generate one or more “random primes.” From Remark 10.2, we know that the number of primes not exceeding $n$ is approximately $n/\ln n$. Hence, if an integer $x \leq n$ is chosen at random, the probability that it is prime is about $1/\ln n$. For $n = 2^{256} \approx 10^{77}$, we have $1/\ln n \approx 1/177$. That is, on average, of 177 random integers $x$ of the appropriate size, one will be prime. To find a prime, we can start by using probabilistic testing methods, and proceed from there.

The Miller-Rabin Algorithm determines whether or not an integer $p$ is composite with respect to an arbitrary integer $r$. To test for 1024-bit prime numbers, the Miller-Rabin Algorithm is usually run three times with different integers $r$. If the result is that $p$ is not composite with respect to $r$ for each $r$ that is tested, then the probability that $p$ is prime is extremely high.

Definition 10.5 Define $\mathbb{Z}_n^\dagger = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$, and define $\phi(n) = |\mathbb{Z}_n^\dagger|$.

Remark 10.6 Thus, $\phi(n)$ is the number of non-negative integers less than $n$ which are relatively prime to $n$. $\phi(n)$ is called Euler’s Totient Function or Euler’s Phi Function.

If $n$ is an integer with prime factorisation $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$ then

$$\phi(n) = \prod_{i=1}^{k} (p_i^{e_i} - p_i^{e_i-1}) = (p_1^{e_1} - p_1^{e_1-1})(p_2^{e_2} - p_2^{e_2-1}) \ldots (p_k^{e_k} - p_k^{e_k-1}).$$

Exercise 10.7 Calculate $\phi(24)$.

Remark 10.8 Some important values of $\phi(n)$ include:

1. $\phi(p) = p - 1$ if $p$ is prime;
2. $\phi(p^n) = p^n - p^{n-1}$ if $p$ is prime;
3. $\phi(s \cdot t) = \phi(s) \cdot \phi(t)$ if $\gcd(s, t) = 1$;
4. $\phi(p \cdot q) = (p - 1) \cdot (q - 1)$ if $p, q$ are distinct primes.
Exercise 10.9  Prove that $\phi(p^n) = p^n - p^{n-1}$ for any prime $p$ and integer $n$. (This is statement (2) above.)

Theorem 10.10  If $\gcd(a, n) = 1$ then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Corollary 10.11  If $\gcd(a, n) = 1$ then $a^{-1} \equiv a^{\phi(n)-1} \pmod{n}$.

Remark 10.12  Suppose $p$ is prime and $a \in \mathbb{Z}_p$. Then

- $a^{p-1} \equiv 1 \pmod{p}$;
- $a^p \equiv a \pmod{p}$;
- $a^{-1} \equiv a^{p-2} \pmod{p}$.

Exercise 10.13  Calculate the multiplicative inverse of 5 in $\mathbb{Z}_{11}$.
Theorem 10.14 (Chinese Remainder Theorem)
Suppose $m_1, m_2, \ldots, m_r$ are pairwise relatively prime positive integers, and let $a_1, a_2, \ldots, a_r$ be integers. Then the system of $r$ congruences $x = a_i \pmod{m_i}$ has a unique solution modulo $M = m_1 \times m_2 \cdots \times m_r$, which is given by

$$x = \sum_{i=1}^{r} a_i M_i y_i \pmod{M},$$

where $M_i = M/m_i$ and $y_i = M_i^{-1}$ modulo $m_i$ for $1 \leq i \leq r$.

Exercise 10.15 Use the Chinese Remainder Theorem to solve the simultaneous congruences

$$x = 22 \pmod{101}$$
$$x = 104 \pmod{113}$$

Theorem 10.16 $\mathbb{Z}_n^*$ is a multiplicative group (with multiplication modulo $n$).

Remark 10.17 Suppose $p$ is a prime. We use $\mathbb{Z}_p^*$ to denote the set of non-zero elements of $\mathbb{Z}_p$. Let $\times_p$ denote multiplication modulo $p$, then $(\mathbb{Z}_p^*, \times_p)$ is a multiplicative group and has $p - 1$ elements.

A generator (or primitive element) of a multiplicative group $(G, \cdot)$ (having $|G|$ elements) is an element $a \in G$ such that $\{a, a^2, a^3, \ldots, a^{|G|}\}$ contains all the elements of $G$. A group that has a generator is called a cyclic group. For all primes $p$, $(\mathbb{Z}_p^*, \times_p)$ is a cyclic group.

The number of generators of a cyclic group $(G, \cdot)$ is $\phi(|G|)$. Thus the number of generators of the group $(\mathbb{Z}_p^*, \times_p)$ is $\phi(p - 1)$. 

109
Exercise 10.18  Show that 2 is a generator of $(\mathbb{Z}_{11}^*, \times_{11})$ but that 5 and 10 are not.

Let $p$ be a prime and let $p - 1 = q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k}$ be the prime factorisation of $\phi(p) = p - 1$. To find a generator of $(\mathbb{Z}_p^*, \times_p)$:

- choose a random element $\alpha$ of $\mathbb{Z}_p^*$;
  - evaluate $b = \alpha^{p-1/n} \pmod{p}$ for each $q_i$;
  - if $b = 1$ for any $q_i$ then $\alpha$ is not a generator, so choose a new $\alpha$;
- if $b \neq 1$ for any $q_i$ then $\alpha$ is a generator.

10.3 The RSA algorithm

The RSA cryptosystem was proposed by Ron Rivest, Adi Shamir and Len Adleman in 1977 and is in wide-spread use world-wide. It uses computations in $\mathbb{Z}_n$, where $n$ is the product of two large, distinct (odd) primes $p$ and $q$. For such $n$, note that $\phi(n) = (p - 1)(q - 1)$.

Algorithm 10.19  RSA Cryptosystem

Let $n = pq$, where $p$ and $q$ are distinct primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) \mid ab = 1 \mod \phi(n)\}.$$ 

For $K = (n, p, q, a, b) \in \mathcal{K}$, $x \in \mathcal{P}$ and $y \in \mathcal{C}$ define

$$e_K(x) = x^b \text{ modulo } n$$

and

$$d_K(y) = y^a \text{ modulo } n.$$ 

The values $n$ and $b$ are made public, and the values $p$, $q$ and $a$ are kept secret.
The following theorem proves that the RSA algorithm is a valid cryptosystem.

**Theorem 10.20** If \( x \in \mathbb{Z}_n \) then \( (x^b)^a = x \mod n \).

**Proof:** We have \( ab = 1 \mod \phi(n) \) and \( \phi(n) = \phi(pq) = (p-1)(q-1) \), so for some \( t \in \mathbb{Z} \),

\[
ab - 1 = t(p-1)(q-1).
\]

Now, for any \( y \neq 0 \) we know that \( y^{p-1} = 1 \mod p \), by Remark 10.12. So in particular

\[
x(\overline{ab-1}) = x^{t(p-1)(q-1)} = (x^{(t(q-1))})^{(p-1)} = y^{(p-1)}
\]

where \( y = x^{t(q-1)} \). Hence we have \( x(\overline{ab-1}) = 1 \mod p \).

By symmetry we also have \( x(\overline{ab-1}) = 1 \mod q \).

Thus \( x(\overline{ab-1}) - 1 = 0 \mod p \) and \( x(\overline{ab-1}) - 1 = 0 \mod q \). Since \( p \) and \( q \) are relatively prime, we can combine these to say \( x(\overline{ab-1}) - 1 = 0 \mod pq \).

Hence \( x(\overline{ab-1}) = 1 \mod pq \), and multiplying by \( x \) we have

\[
(x^a)^b = x \mod pq.
\]

\( \square \)

Thus we have shown that for any message \( x \), if we encrypt \( x \) and then decrypt the ciphertext according to the rules of RSA, then the original message \( x \) will be recovered.

### 10.4 Setting up and using RSA

**Algorithm 10.21** Setting up RSA

Let \( R \) be the recipient of a message, and \( S \) be the sender.

1. \( R \) generates two large, distinct primes, \( p \) and \( q \) (maybe 100 digits each).
2. \( R \) computes \( n = pq \) and \( \phi(n) = (p-1)(q-1) \).
3. \( R \) chooses a random number \( b \) \((0 \leq b \leq \phi(n))\) such that \( \gcd(b, \phi(n)) = 1 \).
4. \( R \) computes \( a \) such that \( a = b^{-1} \mod \phi(n) \) using the Euclidean algorithm, so \( 0 \leq a < \phi(n) \).
5. \( R \) publishes \( n \) and \( b \) in a directory as their public key, and keeps \( a \), \( p \) and \( q \) secret, as their (private) decryption key. \( \square \)

The key setup is done once (rarely) when a user establishes (or replaces) their public key. The exponent \( b \) is usually fairly small to allow fast computation of \( x^b \mod n \); the only restriction on \( b \) is that it must be relatively prime to \( \phi(n) \).

- Typically \( b \) is the same for all users.
- Originally a value of \( b = 3 \) was suggested, but this is now regarded as too small.
- Values such as \( b = 17 = 10001_2 \) and \( b = 2^{16} + 1 = 65537 = 10000000000000001_2 \) are often used as choosing \( b \) such that its binary representation is composed primarily of zeros significantly improves the performance of exponentiation algorithms.
To pass an encrypted message from the sender to the receiver the following steps are taken.

**Algorithm 10.22 Using RSA**

Let $R$ be the recipient of a message, and $S$ be the sender.

1. $S$ reads the public (encryption) key $K = (b, n)$ from the public directory.
2. To encrypt a message $m$, $0 \leq m < n$, the sender $S$ computes $c = m^b \mod n$.
3. $S$ sends ciphertext $c$ to $R$.
4. To decrypt the ciphertext $c$, $R$ uses their private key $K^{-1} = (a, p, q)$, and computes $m = c^a \mod n$.

**Example 10.23** Suppose Bob chooses $p = 101$ and $q = 113$. Then $n = 11413$ and $\phi(n) = 100 \times 112 = 11200$. Bob also (randomly) chooses $b = 569$. Note that gcd(569, 11200) = 1. Hence Bob’s secret decryption exponent is $a$ where $ab = 1 \mod 11200$; thus $a = 1929$.

Bob publishes $n = 11413$ and $b = 569$ in a public directory. Now if Alice wants to send the plaintext 1234 to Bob, she needs to compute $1234^{569}$ modulo 11413 which is 1932. When Bob receives the ciphertext 1932 he uses his secret key $a$ (and maybe also $p$ and $q$) to compute $1932^{1929}$ modulo 11413 which is 1234.

In RSA, we need to evaluate exponents and perform modular arithmetic on very large numbers. In practice, we can get numbers of length 300 digits (or 1024+ bits), but most computers can’t directly handle numbers larger than 32 or 64 bits. Luckily, there are techniques for speeding up RSA calculations.

**Example 10.24** In Example 10.23, Alice needed to evaluate $1234^{569}$ mod 11413. The integer $1234^{569}$ has 1759 digits, so first calculating that and then reducing modulo 11413 would take many multiplications. So how can this calculation be done efficiently?

Let $m$ denote the message 1234 and $n = 11413$. To evaluate $m^{569}$ we first write 569 as the sum of distinct powers of 2.

$$569 = 512 + 32 + 16 + 8 + 1 = 2^9 + 2^5 + 2^4 + 2^3 + 2^0$$

Then we compute

$$m^2 \mod n, \quad m^4 \mod n, \quad m^8 \mod n, \quad m^{16} \mod n, \quad \ldots \quad m^{512} \mod n$$

and multiply the appropriate values.

\[
\begin{align*}
(1234)^2 \mod 11413 &= 4827 \\
(1234)^4 \mod 11413 &= (4827)^2 \mod 11413 = 5996 \\
(1234)^8 \mod 11413 &= (5996)^2 \mod 11413 = 1066 \\
(1234)^{16} \mod 11413 &= (1066)^2 \mod 11413 = 6469 \\
(1234)^{32} \mod 11413 &= (6469)^2 \mod 11413 = 7903 \\
(1234)^{64} \mod 11413 &= (7903)^2 \mod 11413 = 5473 \\
(1234)^{128} \mod 11413 &= (5473)^2 \mod 11413 = 6017 \\
(1234)^{256} \mod 11413 &= (6017)^2 \mod 11413 = 2253 \\
(1234)^{512} \mod 11413 &= (2253)^2 \mod 11413 = 8637
\end{align*}
\]
Thus \((1234)^{569} = (m^{512} \cdot m^{32} \cdot m^{16} \cdot m^8 \cdot m^1) \mod n\) which is
\[
(8637 \cdot 7903 \cdot 6469 \cdot 1066 \cdot 1234) \mod 11413 \\
= (8471 \cdot 6469 \cdot 1066 \cdot 1234) \mod 11413 \\
= (5086 \cdot 1066 \cdot 1234) \mod 11413 \\
= (501 \cdot 1234) \mod 11413 \\
= 1932
\]
This has been evaluated with only 13 modular multiplications! This method is often referred to as the \textit{square and multiply} method for fast exponentiation.

To decrypt a received message \(c\), the receiver must compute \(c^a \mod n\). This can be done as illustrated above, but if the receiver knows the values of the primes \(p\) and \(q\), then the Chinese Remainder Theorem can be used to speed up the decryption process. Rather than working \(\mod n\), the receiver can use two equations \(\mod p\) and \(\mod q\) respectively. Since \(p\) and \(q\) are each much smaller than \(n\), decryption is much faster.

\textbf{Algorithm 10.25} \hspace{1em} CRT and RSA decryption

To solve the decryption equation \(m = c^a \mod n\):

1. compute \(c \mod p\) and \(a \mod (p - 1)\) and then compute
   \[
   m_1 = (c \mod p)^{a \mod (p - 1)} \mod p.
   \]

2. compute \(c \mod q\) and \(a \mod (q - 1)\) and then compute
   \[
   m_2 = (c \mod q)^{a \mod (q - 1)} \mod q.
   \]

Then, use the Chinese Remainder Theorem to solve the pair of equations
\[
\begin{align*}
m &= m_1 \mod p \\
m &= m_2 \mod q.
\end{align*}
\]

\textbf{Exercise 10.26} In Example 10.23 Bob had to compute \(1932^{1929} \mod 11413\) to decrypt his received message. Use the above algorithm to verify that the message is 1234.
Remark 10.27 RSA makes use of very large integers which have to be stored exactly. These numbers are much larger than any computer can store exactly, without using some special tricks. The Chinese Remainder Theorem can be used to do this.

Exercise 10.28 Assume that a certain (unrealistic) computer can only exactly store numbers less than or equal to 100. Use the CRT to show how numbers much larger than 100 can be stored exactly.

10.5 Using RSA for digital signatures

Suppose that a bank has set up an RSA scheme so that its customers can send encrypted messages to the bank (so the bank holds the private key \((a, p, q)\) and all the customers have access to the public key \((b, n)\)). If the bank wishes to send a message to its customers, it would be nice if the customers could verify that the message originated from the bank and is not a hoax message. RSA can be used for this purpose.

Let \(m\) be the message that the bank wishes to send to its customers. The bank uses the private key \((a, p, q)\) to compute \(r = m^a \pmod{n}\) and sends \(r\) to its customers. Now a customer, knowing that this transmission comes from the bank, looks up the bank’s public key \((b, n)\) and computes \(r^b \pmod{n}\). Because of the symmetry of the cryptosystem, we can see from the proof of Theorem 10.20 that \(r^b = m \pmod{n}\) and hence the customer will receive an intelligible message. Since the bank is the only holder of the private key \(a\), the message must have originated there. In effect, the bank has signed the message with the private signature \(a\).

Note that here the scheme is not being used for secrecy. Anyone can intercept the message \(r\) and use the bank’s public key to compute \(m\). In this scenario the scheme is being use to authenticate the message.
10.6 Practical security and cryptanalysis of RSA

- The security of RSA rests on the difficulty of factoring the modulus $n$ (equivalently, the difficulty of computing $\phi(n)$).

- Suppose that we are given $n = pq$ where $p$ and $q$ are primes with about 100 digits each. If we tried to find $p$ by dividing $n$ by all the primes less than $\sqrt{n}$, then we would have to do about

$$\frac{\sqrt{n}}{\ln \sqrt{n}} \approx \frac{10^{100}}{100 \ln(10)} \geq \frac{10^{98}}{3}$$

divisions. $10^{14}$ operations is currently regarded as a limit for computational feasibility. There are better algorithms known for factoring integers, but if $n$ is around 200 digits, then it is infeasible to find the factors of $n$ by any known algorithm.

- The best known theoretical factoring algorithm (Brent-Pollard) takes:

<table>
<thead>
<tr>
<th>Decimal Digits in $N$</th>
<th>Number of Bit Operations to Factor $N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$7.20 \times 10^4$</td>
</tr>
<tr>
<td>40</td>
<td>$3.11 \times 10^6$</td>
</tr>
<tr>
<td>60</td>
<td>$4.63 \times 10^8$</td>
</tr>
<tr>
<td>80</td>
<td>$3.72 \times 10^{10}$</td>
</tr>
<tr>
<td>100</td>
<td>$1.97 \times 10^{12}$</td>
</tr>
<tr>
<td>120</td>
<td>$7.69 \times 10^{13}$</td>
</tr>
<tr>
<td>140</td>
<td>$2.35 \times 10^{15}$</td>
</tr>
<tr>
<td>160</td>
<td>$5.92 \times 10^{16}$</td>
</tr>
<tr>
<td>180</td>
<td>$1.26 \times 10^{18}$</td>
</tr>
<tr>
<td>200</td>
<td>$2.36 \times 10^{19}$</td>
</tr>
</tbody>
</table>

- We have seen slow improvements in factoring over the years. The biggest improvement would come from an improved algorithm (rather than faster computers). Barring a dramatic breakthrough 1024+ bit RSA looks secure for now.

- The RSA inventors proposed a challenge published in Scientific American in 1977. They offered a prize of US$100 to factor a 129 digit number. The prize was claimed in 1994. The following table gives the results of research into factoring numbers having 100 - 200 decimal digits. Note that a MIPS-year is $3 \times 10^{13}$ instructions/year, and the times in CPU Years are on a 2.2 GHz Opteron CPU.

<table>
<thead>
<tr>
<th>Decimal Digits</th>
<th>When</th>
<th>MIPS-Years</th>
<th>Algorithm Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Apr 91</td>
<td>7</td>
<td>Quadratic Sieve</td>
</tr>
<tr>
<td>110</td>
<td>Apr 92</td>
<td>75</td>
<td>Quadratic Sieve</td>
</tr>
<tr>
<td>120</td>
<td>Jun 93</td>
<td>830</td>
<td>Quadratic Sieve</td>
</tr>
<tr>
<td>129</td>
<td>Apr 94</td>
<td>5000</td>
<td>Quadratic Sieve</td>
</tr>
<tr>
<td>130</td>
<td>Apr 96</td>
<td>500</td>
<td>Number Field Sieve</td>
</tr>
<tr>
<td>140</td>
<td>Feb 99</td>
<td>1500</td>
<td>Number Field Sieve</td>
</tr>
<tr>
<td>155</td>
<td>Aug 99</td>
<td>8000</td>
<td>Number Field Sieve</td>
</tr>
<tr>
<td>160</td>
<td>April 2003</td>
<td>?</td>
<td>Lattice Sieve</td>
</tr>
<tr>
<td>174</td>
<td>Dec 2003</td>
<td>?</td>
<td>Lattice Sieve</td>
</tr>
<tr>
<td>193</td>
<td>Nov 2005</td>
<td>30 CPU Years</td>
<td>Lattice Sieve</td>
</tr>
<tr>
<td>200</td>
<td>May 2005</td>
<td>55 CPU Years</td>
<td>Lattice Sieve</td>
</tr>
</tbody>
</table>
There are a number of possible cryptanalysis attacks on RSA. These include:

- **Timing attacks**: try to keep track of *how long* it takes the computer to perform modular exponentiation.
- **Factoring attacks**: try to *factor* $n$, thus giving $p$ and $q$.
- **Totient function attacks**: try to find an efficient algorithm which, when given $n$, computes $\phi(n)$.

### 10.7 Internet application of RSA: *Pretty Good Privacy*

- Early versions of PGP encrypted data using IDEA with a random session key, and used RSA to encrypt the session key.
- No secure channels were needed to exchange keys between users, which made PGP much easier (and safer) to use.
- In December 1994 Philip Zimmermann faced prosecution for “exporting” PGP out of the United States but in January 1996 the US Government dropped the case. A US law prohibits the export of encryption software out of the country. Zimmermann did not do this, but the US government hoped to establish the proposition that posting an encryption program on the Internet constitutes exporting it - in effect, stretching export control into domestic censorship. If the government had won it would have had a severe effect on the free flow of information on the global network.

### 10.8 The ElGamal algorithm

The ElGamal algorithm was proposed in 1985 by Taher ElGamal and is based on the Discrete Logarithm problem in $\mathbb{Z}_p^*$ or $GF(2^k)$. Consider the group $(\mathbb{Z}_p^*, \times_p)$ with generator $\alpha$. Given an element $\beta \in \mathbb{Z}_p^*$ such that $\beta = \alpha^i \pmod{p}$, the discrete logarithm problem asks you to find $i$.

**Algorithm 10.29 ElGamal Cryptosystem**

Let $p$ be a large prime, let $\alpha$ be a generator of $(\mathbb{Z}_p^*, \times_p)$ and let $a \in \{2, 3, \ldots, p-2\}$. Let $\mathcal{P} = \mathbb{Z}_p^*$ and $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) \mid \beta = \alpha^a \pmod{p}\}.$$  

For $K = (p, \alpha, a, \beta) \in \mathcal{K}$, $x \in \mathcal{P}$, $(y_1, y_2) \in \mathcal{C}$ and $d \in \{2, 3, \ldots, p-2\}$ define

$$e_K(x, d) = (y_1, y_2) \text{ where } y_1 = \alpha^d \pmod{p}, \quad y_2 = \beta^d x \pmod{p}$$

and

$$d_K(y_1, y_2) = (y_1^a)^{-1} y_2 \pmod{p}.$$  

The values $p$, $\alpha$ and $\beta$ are made public, and the value of $a$ is kept secret. The value of $d$ is chosen independently for each encryption and is not used in the decryption process.
The following theorem proves that the ElGamal algorithm is a valid cryptosystem.

**Theorem 10.30** If $p$ is a prime, $\alpha$ is a generator of $(\mathbb{Z}_p^*, \times_p)$, $a \in \{2, 3, \ldots, p - 2\}$, $\beta = \alpha^a \mod p$ and $x \in \mathbb{Z}_p^*$ then for any $d \in \{2, 3, \ldots, p - 2\}$

$((\alpha^d)^a)^{-1} \beta^d x = x \mod p$.

**Exercise 10.31** Prove Theorem 10.30.

Thus we have shown that for any message $x$, if we encrypt $x$ and then decrypt the ciphertext according to the rules of ElGamal, then the original message $x$ will be recovered.

### 10.9 Setting up and using ElGamal

**Algorithm 10.32 Setting up ElGamal**

Let $R$ be the recipient of a message, and $S$ be the sender.

1. $R$ generates a large prime $p$ and a generator $\alpha$ for the group $(\mathbb{Z}_p^*, \times_p)$.

2. $R$ chooses a random number $a \in \{2, 3, \ldots, p - 2\}$.

3. $R$ computes $\beta$ such that $\beta = \alpha^a \mod p$.

4. $R$ publishes $p$, $\alpha$ and $\beta$ in a directory as their public key, and keeps $a$ as their (private) decryption key.

□
To pass an encrypted message from the sender to the receiver the following steps are taken.

**Algorithm 10.33 Using ElGamal**

Let \( R \) be the recipient of a message, and \( S \) be the sender.

1. \( S \) reads the public (encryption) key \( K = (p, \alpha, \beta) \) from the public directory.
2. \( S \) chooses a random integer \( d \in \{2, 3, \ldots, p - 2\} \).
3. To encrypt a message \( m \), where \( 0 \leq m < p \), the sender \( S \) computes \( c_1 = \alpha^d \mod p \) and \( c_2 = \beta^d m \mod p \).
4. \( S \) sends ciphertext \((c_1, c_2)\) to \( R \).
5. To decrypt the ciphertext \((c_1, c_2)\), \( R \) uses their private key \( K^{-1} = (a) \), and computes \( m = ((c_1)^a)^{-1}c_2 \mod p \).

\[\square\]

**Example 10.34** Suppose Bob chooses \( p = 727 \), the generator \( \alpha = 80 \) and the secret value \( a = 6 \). He computes \( \beta = \alpha^a \mod p = 80^6 \mod 727 = 514 \).

Bob publishes \( p = 727 \), \( \alpha = 80 \) and \( \beta = 514 \) in a public directory.

Now if Alice wants to send the plaintext 123 to Bob, she chooses \( d \in \{2, 3, \ldots, p - 2\} \), say \( d = 7 \) and computes

\[c_1 = \alpha^d = 80^7 \mod 727 = 408 \quad \text{and} \quad c_2 = \beta^d m = 514^7 \cdot 123 \mod 727 = 390.\]

When Bob receives the ciphertext \((408, 390)\) he uses his secret key \( a = 6 \) to compute

\[((c_1)^a)^{-1}c_2 \mod p = ((408)^6)^{-1}390 \mod 727 = 123.\]

**Remark 10.35** The key setup is done once (rarely) when a user establishes (or replaces) their public key. In the setup we need to choose a large prime \( p \) with the additional requirement that \( \phi(p) = p - 1 \) has a large prime factor (to protect against certain attacks). There are reasonably efficient algorithms for

- choosing a large prime \( p \);
- finding a primitive element in \( \mathbb{Z}_p \);
- carrying out exponentiation and modular multiplications.

The biggest problem in the setting up of ElGamal is making sure the large prime is suitable, which requires factoring \( \phi(p - 1) \).

It is critical that a new value of \( d \) is chosen for each encryption. If the same \( d \) is used repeatedly, then the cryptosystem is vulnerable to a known plaintext attack.

ElGamal encryption causes message expansion, generating the ciphertext pair for each plaintext, so using ElGamal encryption reduces the transmission throughput by a factor of two.
To decrypt the message \((y_1, y_2)\), we must calculate \(((y_1)^a)^{-1}\). Calculating a multiplicative inverse using the Euclidean Algorithm is computationally expensive. However, in this case, the inverse can be quickly computed (in a similar way to Exercise 10.13) using Fermat’s Little Theorem (see Remark 10.12) which says \(a^{p-1} \equiv 1 \pmod{p}\) provided \(\gcd(a, p) = 1\).

\[
((y_1)^a)^{-1} \pmod{p} = \frac{y_1^{p-1}}{y_1} \equiv y_1^{p-1-a} \pmod{p}.
\]

**Example 10.36** In decrypting the ciphertext \((c_1, c_2) = (408, 390)\) from Example 10.34 we note that

\[
(408^6)^{-1} = 408^{727-1-6} = 408^{720} \pmod{727} = 375
\]

and \(375 \cdot 390 \pmod{727} = 123\).

### 10.10 Practical security and cryptanalysis of ElGamal

- The security of ElGamal rests on the difficulty of solving the equation \(\beta = \alpha^i \pmod{p}\) for \(i\).
- An Exhaustive Key Search involves repeatedly choosing a value for \(i \in \{2, 3, \ldots, p-2\}\), calculating \(\alpha^i \pmod{p}\) and then comparing this to the known value \(\beta\). To be secure against an Exhaustive Key Search attack, we need \(p - 1 \geq 2^{128}\).
- More sophisticated attacks against the ElGamal cryptosystem include Shank’s Algorithm, the Pollard’s Rho Method, the Pohlig-Hellman Algorithm, and the Index Calculus Method. To be secure against these attacks, we require \(p - 1 \geq 2^{256}\). To protect against these attacks we also require that \(p - 1\) has a large prime factor.
- Thus, as with RSA, we need \(p\) to be a prime with binary representation of at least 1024 bits. (For ElGamal we also want \(\alpha\) and \(\beta\) to be 1024-bit values.)
- RSA and ElGamal exhibit similar performances when both use 1024-bit sized parameters.

### 10.11 Internet application of ElGamal: Pretty Good Privacy 3

Early versions of PGP used RSA for the public-key cryptosystem, but in addition to the illegal export charges faced by Zimmermann, there were also some difficulties with the patent for RSA. Consequently, in 1996, in the development of PGP 3 (and PGP 5), the decision was made to used the ElGamal cryptosystem instead as it didn’t have any patent issues.