RSA, Public-Key Cryptography, and Authentication Protocols

by

Moriah E. Wright

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in the

Mathematics

Program

YOUNGSTOWN STATE UNIVERSITY

May, 2012

RSA, Public-Key Cryptography, and Authentication Protocols

Moriah E. Wright

ABSTRACT

In the modern electronic age, secure communication is vital to the efficient workings in all areas of life. From governmental proceedings and military devices to personal banking and email, securely communicating between parties has become a necessity that has furthered research in the field of cryptography. This thesis is an introduction to a particular type of cryptography, public-key cryptography, and examines the basics of RSA and ElGamal cryptosystems. Worked-out examples for these types of encryption, decryption and authentication protocols are explored, as well.

Contents

1 Background

History is saturated with the need and solution to implementing secure and private communications between different parties. From the ancient Roman emperor's infamous encryption of military strategy using the "Caesar cipher" to the encrypting of transaction and communications in the modern electronic age, there is a need to protect information from those with malicious intent.

Even with the best precautions, it is difficult to maintain secrecy and security when there are many parties involved in the ring of communication. Thus, it is the goal of modern encryption to reduce the number of components of the cryptosystem that must be kept confidential and rely more heavily on the difficulty - even infeasibility - of decryption given modern computing capabilities. As advances in mathematical techniques and hardware are made in the coming years, encryption methods need to be adapted to ensure continued security.

The branch of modern cryptography known as "public-key cryptography" is explored in this paper with the intent of providing an deeper look at the mathematical processes implemented in the encryption and authentication of messages. Sophisticated software and significantly larger values are used in practice with these encryption schemes, but for illustration in this paper, we will use unrealistically small examples.

Before proceeding any further, I would like to introduce to you Alice and Bob, our friends who wish to send messages to each other in a secure manner. Eve is the malicious eavesdropper who is persistently trying to intercept and decipher Alice and Bob's messages. These three will be very familiar characters in the "stories" of the following pages.

1.1 Theorems and Definitions

The theory behind encryption is heavily rooted in the field of number theory and abstract algebra. Many of the concepts and computations stem from the definitions and theorems below.

Definition 1 (Divisibility). *An integer $a \neq 0$ divides b if $\frac{b}{a} \in \mathbb{Z}$, that is, $b = ka$ for some $k \in \mathbb{Z}$. We denote it by $a \mid b$; otherwise, $a \nmid b$.*

Theorem 1 (The Division Algorithm). *Let $a, b \in \mathbb{Z}$, $a > 0$ and $b \geq a$. Then, there exist unique integers q and r with $0 \leq r < a$ such that $b = aq + r$.*

Theorem 2 (The Euclidean Algorithm). *Let $a, b \in \mathbb{Z}$, $(a \geq b > 0)$, and set $a = r_{-1}, b = r_0$. By repeatedly applying the Division Algorithm, we get $r_{j-1} = r_j q_{j+1} + r_{j-1}$*

with $0 < r_{j+1} < r_j$ *for all* $0 \leq j < n$*, where* n *is the least nonnegative integer such that* $r_{n+1} = 0$*, in which case* $\gcd(a, b) = r_n$*.*

Theorem 3 (The Chinese Remainder Theorem)**.** *Let* $n_i \in \mathbb{N}$ *for natural numbers* $i \leq k \in \mathbb{N}$ *be pairwise relatively prime, set* $n = \prod_{j=1}^{k} n_j$ *and let* $r_i \in \mathbb{Z}$ *for* $i \leq k$*. Then the system of* k *simultaneous linear congruences given by:*

$$\begin{aligned} x &\equiv r_1 \bmod n_1, \\ x &\equiv r_2 \bmod n_2, \\ &\vdots \\ x &\equiv r_k \bmod n_k, \end{aligned}$$

has a unique solution modulo n*.*

Definition 2 (Euler's ϕ-Function)**.** *For any* $n \in \mathbb{N}$ *the Euler* ϕ*-function, also known as Euler's Totient,* $\phi(n)$*, is defined to be the number of* $m \in \mathbb{N}$ *such that* $m < n$ *and* $\gcd(m, n) = 1$ *For a prime* p*,* $\phi(p) = p - 1$*.*

Definition 3 (Order)**.** *Let* m *be a positive integer and suppose that* $\gcd(a, m) = 1$*. The order of* a *modulo* m*, denoted by* $\operatorname{ord} a$*, is the smallest positive integer* h *such that* $a^h \equiv 1 \bmod m$*.*

Theorem 4. *Let* m *be a positive integer and suppose that* $\gcd(a, m) = 1$*.*

1. $a^s \equiv 1 \bmod m$ *if and only if* $(\operatorname{ord} a) | s$*. In particular,* $(\operatorname{ord} a) | \phi(m)$*.*
2. $a^s \equiv a^t \bmod m$ *if and only if* $s \equiv t \pmod{\operatorname{ord} a}$*.*

Definition 4 (Primitive Root)**.** *Let* m *be a positive integer, and suppose that* $\gcd(a, m) = 1$*. If the order of* $a \bmod m$ *is* $\phi(m)$*, then* a *is called a primitive root of* m*.*

Theorem 5. *If* $\gcd(g, m) = 1$*, then* g *is a primitive root of* m *if and only if*

$$g^{\phi(m)/q} \neq 1 \bmod m$$

for every prime divisor q *of* $\phi(m)$*.*

Theorem 6 (Euler Theorem)**.** *Let* $a, m \in \mathbb{N}$*. If* $\gcd(a, m) = 1$*, then*

$$a^{\phi(m)} \equiv 1 \bmod m.$$

Theorem 7 (Fermat's Little Theorem)**.** *Let* p *be prime and suppose* $p \nmid a$*. Then* $a^{p-1} \equiv 1 \bmod p$*. Equivalently, if* a *is any integer, then* $a^p \equiv a \bmod p$*.*

Definition 5 (Modular Order of an Integer)**.** *Let $m \in \mathbb{Z}$, $n \in \mathbb{N}$ and* $\gcd(m,n) = 1$*. The order of m modulo n is the smallest $e \in \mathbb{N}$ such that $m^e \equiv 1 \bmod n$, denoted by $e = \operatorname{ord}_n(m)$, and we say that m belongs to the exponent $e \bmod n$.*

Theorem 8. *If $k = ord_n a$, then the integers $a, a^2, a^3, ..., a^k = 1$ are all incongruent modulo n.*

Definition 6. *If $\gcd(a,n) = 1$, and a has order $\phi(n)$, then a is called a primitive root $\bmod n$. It is a generator of $\mathbb{Z}_n^*$.*

Theorem 9. *If integer n has a primitive root, then it has exactly $\phi(\phi(n))$ of them.*

Theorem 10. *The only integers that have primitive roots are: $p^\alpha, 2p^\alpha, 2, 4$ where p is an odd prime.*

Theorem 11 (Corollary)**.** *There are $\phi(p-1)$ incongruent primitive roots modulo p.*

Theorem 12. *If $\gcd(g,m) = 1$, then g is a primitive root of m if and only if*

$$g^{\frac{\phi(m)}{q}} \not\equiv 1 \bmod m$$

for every prime divisor q of $\phi(m)$.

1.2 Useful Algorithms

Changing Between Different Bases

Much of the processes involved in cryptosystems involve changing a block of plaintext from base 26 (since there are 26 letter in the English alphabet) to base 10 (the base of our standard number system) and vice versa. The process below describes changing a number, n, from base 10 into base b.

Using the Division Algorithm, we find the quotients and remainders when dividing by b until we have a quotient of 0. For the sake of this example, let's assume that four iterations had to be performed until a quotient of 0 was achieved.

$$\begin{aligned} n &= q_1 \cdot b + a_1 \\ q_1 &= q_2 \cdot b + a_2 \\ q_2 &= q_3 \cdot b + a_3 \\ q_3 &= 0 \cdot b + a_4 \end{aligned}$$

Now, we substitute back in each quotient:

$$\begin{aligned} n &= q_1 b + a_1 \\ &= (q_2 b + a_2)b + a_1 \\ &= ((q_3 b + a_3)b + a_2)b + a_1 \\ &= (((0 \cdot b + a_4)b + a_3)b + a_2)b + a_1 \end{aligned}$$

Simplifying, we get

$$\begin{aligned} n &= (((0 \cdot b + a_4)b + a_3)b + a_2)b + a_1 \\ &= ((0b^2 + a_4 b + a_3)b + a_2)b + a_1 \\ &= (0b^3 + a_4 b^2 + a_3 b + a_2)b + a_1 \\ &= 0b^4 + a_4 b^3 + a_3 b^2 + a_2 b + a_1 \\ &= a_4 b^3 + a_3 b^2 + a_2 b + a_1 \end{aligned}$$

Thus, a number n in base 10 can be written as $(a_4\ a_3\ a_2\ a_1)_b$ in base b.

The Repeated Squaring Method for Modular Exponentiation

In theory, we can perform any calculations we wish. However, we are often limited by the computational power at our disposal. Exponentiation is one of the most common calculations involved in encryption processes. Even with most powerful computers of today, the task of exponentiation modulo n would overflow the memory of most supercomputers. Thus, methods for efficiently computing modular exponentiation must be implemented. One of the most useful methods, the Repeated Squaring Method, is discussed below. We can efficiently square and reduce modulo n using the algorithm described below to compute the least positive residue modulo n.

Repeated Squaring Method for Modular Exponentiation

We compute $b^r \bmod n$ for $b, r, n \in \mathbb{N}$ as follows. First, write r in its binary (base 2) representation:

$$r = \sum_{j=0}^{k} a_j 2^j.$$

We wish to calculate $c \equiv b^r \pmod{n}$ in a stepwise fashion as follows.

Initial step: Set $b_0 = b$ and

$$c = \begin{cases} 1, & \text{if } a_0 = 0, \\ 0, & \text{if } a_0 = 1. \end{cases}$$

Perform the following step for each $j = 1, 2, \ldots, k$:

j-th Step: Calculate the least nonnegative residue, b_j, of $(b_{j-1})^2 \bmod n$. If $a_j = 1$, then replace c by $c \cdot b_j$, and reduce modulo n. If $a_j = 0$ leave c unchanged. What is achieved at the j^{th} step is the computation of

$$c_j \equiv b^{r_j} \pmod{n},$$

where c_j is the least nonnegative residue of b^{r_j} modulo n, and

$$r_j = \sum_{i=0}^{j} a_i 2^i.$$

Hence, at the k^{th} step, we have calculated $c \equiv b^r \pmod{n}$.

Example using repeated squaring method

We wish to compute $3^{61} \bmod 101$. For this relatively small exponentiation, 3^{61} is already a thirty-digit number. To compress the size of computations such as this, we use the repeated squaring method. First, we write 61 in its binary representation:

$$\begin{aligned} 61 &= 2 \cdot 30 + 1 \\ 30 &= 2 \cdot 15 + 0 \\ 15 &= 2 \cdot 7 + 1 \\ 7 &= 2 \cdot 3 + 1 \\ 3 &= 2 \cdot 1 + 1 \\ 1 &= 2 \cdot 0 + 1 \end{aligned}$$

Thus, the binary representation of 3^{61} is:

$$\begin{aligned} 61 &= (1\,1\,1\,1\,0\,1)_2 \\ &= 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2^5 + 2^4 + 2^3 + 2^2 + 1 \end{aligned}$$

Thus, $3^{61} \bmod 101 = 3^{1+2^2+2^3+2^4+2^5} \bmod 101$.

$$\begin{aligned}
3^1 &\equiv 3 \bmod 101\\
3^{2^2} &\equiv (3^2)^2 \equiv 9^2 \equiv 81 \bmod 101\\
3^{2^3} &\equiv ((3^2)^2)^2 \equiv (81)^2 \equiv (-20)^2 \equiv -4 \bmod 101\\
3^{2^4} &\equiv (3^{2^3})^2 \equiv (-4)^2 \equiv 16 \bmod 101\\
3^{2^5} &\equiv (3^{2^4})^2 \equiv (16)^2 \equiv 54 \bmod 101
\end{aligned}$$

Therefore,

$$\begin{aligned}
3^{61} &\equiv 3^1 \cdot 3^{2^2} \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^5}\\
&\equiv (3)(81)(-4)(16)(54) \bmod 101\\
&\equiv -94 \bmod 101\\
&\equiv 7 \bmod 101.
\end{aligned}$$

2 The Discrete Log Problem

The security of many of the algorithms involved in encryption rely on the predicted difficulty in solving a problem known as the discrete log problem (DLP).

Generalized Discrete Logarithm Problem: Given a finite cyclic group, G, of order $n \in \mathbb{N}$, a generator α of G, and an element $\beta \in G$, find that unique nonnegative integer $e \leq n-1$ such that $\alpha^e = \beta$.

The Discrete Logarithm Problem: Finding a unique nonnegative integer $e \leq p-2$ such that $c \equiv m^e \pmod{p}$ given integers m, c, and prime p,

$$e \equiv \log_m(c) \bmod p. \tag{1}$$

If the modulus p is chosen properly (a rather ambiguous criterion), the discrete log is very difficult to solve. As computer software advances, the size of the modulus p has to be increased to maintain the difficulty of solving the problem. With current methods and computing capabilities, if p has more than 308 digits (1024 bits) and $p-1$ has at least one large factor (another seemingly arbitrary criterion), then p has been properly chosen. We choose p in this way since the following algorithm can efficiently calculate discrete logs when $p-1$ has only small prime factors.

2.1 Silver-Pohlig-Hellman Algorithm

Silver-Pohlig-Hellman Algorithm for Computing Discrete Logs

Let α be a generator of $\mathbb{F}_p^*$ and let $\beta \in \mathbb{F}_p^*$ and assume that we have a factorization:

$$p - 1 = \prod_{j=1}^{r} p_j^{\alpha_j},$$

$\alpha_j \in \mathbb{N}$, where p_j are distinct primes. We compute $e = \log_\alpha \beta$ by computing e mod $p_j^{\alpha_j}$ for $j = 1, 2, \ldots, r$, and apply the Chinese Remainder Theorem.

Pohlig-Hellman Symmetric-Key Exponentiation Cipher

- A secret prime p and a secret enciphering key, $e \in \mathbb{N}$ with $e \leq p - 2$ are chosen.
- A secret deciphering key d is computed via $ed \equiv 1 \pmod{p-1}$.
- Encryption of of plaintext message units m is accomplished via $c \equiv m^e \pmod{p}$.
- Decryption is achieved via $m \equiv c^d \pmod{p}$.

3 Public-Key Cryptosystems

3.1 Symmetric vs. Asymmetric Systems

As mentioned in the introduction, message encryption is not a new art. In the messages pertaining to military strategy sent from Julius Caesar to his generals, known as the "Caesar cipher", an algorithm known as a "shift cipher" (a type of substitution cipher) was used. This is one of the simplest encryption techniques. In this method, each letter of the plaintext is replaced with a letter a fixed number of positions further down the alphabet. The decryption of the message is achieved by simply shifting the letters a fixed number of positions in the opposite direction. This is an example of a *symmetric-key algorithm* in which the key for enciphering the plaintext and deciphering the ciphertext are trivially related (or even identical). In modern practice, the Caesar cipher offers essentially no communication security.

Before sending messages to each other using a symmetric cipher, Alice and Bob must first decide and agree on their secret key, k. This is feasible if they are able to meet face-to-face or use another secure channel rather than communicating their secret key over an insecure channel. This may not always be feasible, however, and the ever-vigilant Eve will snatch the opportunity to intercept any insecure communication. Thus, in this electronic age, a system which does not rely on insecure communication of private keys - an asymmetric encryption system - must be developed.

An non-mathematical example of public-key cryptography is to visualize a safe in the middle of a public park. Alice is the only one who owns a key to this safe, which locks as soon as the door is closed. She leaves the safe open so that anyone who wishes may place a message into this safe, then close the door. Once someone, such as Bob, puts his message into the safe, only Alice can retrieve the message using her private key. The encryption process is the process of putting the message in the open safe and shutting the door - the public key (open door) is available for anyone. The decryption algorithm is the process of opening the safe with the key, owned only by Alice.

So now we examine the formulation of an asymmetric cipher. We have the spaces of keys $\mathcal{K}$, plaintext $\mathcal{M}$, and ciphertexts $\mathcal{C}$. Each key is comprised of two parts, $k = (k_{priv}, k_{pub})$, known as the private key and public key, respectively. For each public key, there is an encryption function

$$e_{k_{pub}} : \mathcal{M} \longrightarrow \mathcal{C},$$

and for each private key, there is a decryption function

$$d_{k_{priv}} : \mathcal{C} \longrightarrow \mathcal{M}.$$

We must have the property that if $k \in \mathcal{K}$, then

$$d_{k_{priv}}(e_{k_{pub}}(m)) = m \quad \forall m \in \mathcal{M}.$$

To ensure security, it must be difficult for Eve to be able to compute $d_{k_{priv}}(c)$ even with knowledge of and access to k_{pub}. This "trapdoor function" scheme allows the public key to be communicated between Alice and Bob over an insecure channel without the integrity of the message being compromised.

The effective implementation of a public-key cryptographic system relies on the effectiveness of a one-way function.

Definition 7 (One-way Functions)**.** *A one-way function f is a function from a set $\mathcal{M}$ of plaintext to a set $\mathcal{C}$ of ciphertext where computing $f(m) = c$ for any $m \in \mathcal{M}$ is mathematically "easy" but finding $f^{-1}(c) = m$ for a $c \in \mathcal{C}$ is computationally infeasible.*

$$\begin{tikzcd} \boxed{m \in \mathcal{M}} \arrow[r, bend left, "\text{easy}"] & \boxed{f(m) \in \mathcal{C}} \arrow[l, bend left, "\text{hard}"] \end{tikzcd}$$

A cryptosystem consisting of a set of enciphering transformations $\{E_e\}$ and a set of deciphering transformations $\{D_d\}$ is called a Public-key Cryptosystem if, for each pair (e, d), the enciphering key e, called the public key, is made publicly available, while the deciphering key d, called the private key, is kept secret. The cryptosystem must satisfy the property that it is computationally infeasible to compute d from e.

3.2 Diffie-Hellman Key-Exchange

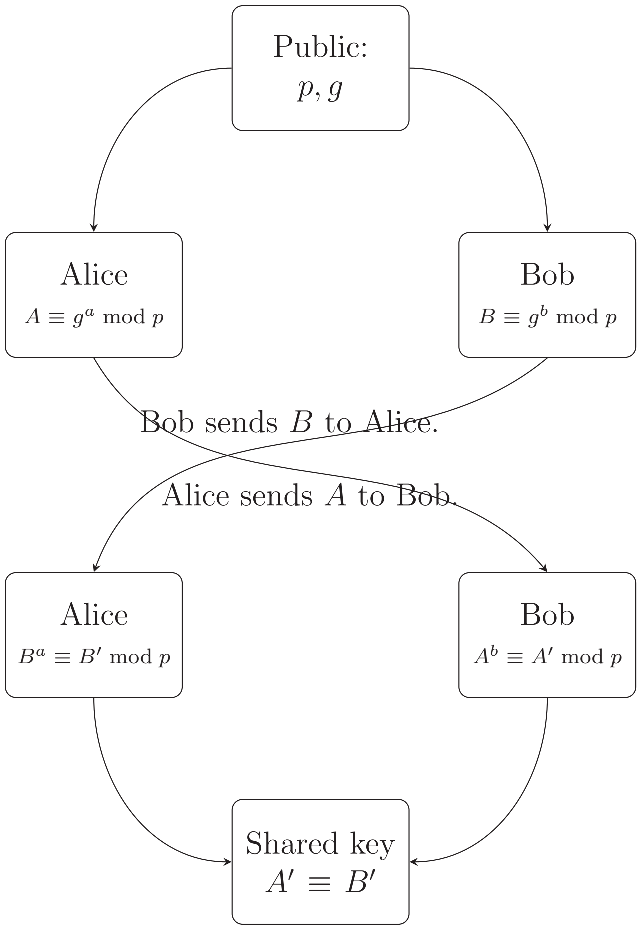
In their 1976 paper, "New Directions in Cryptography", Whitfield Diffie and Martin Hellman outlined an effective implementation of an asymmetric cryptosystem. Relying on sheer difficulty of solving the discrete logarithm problem, the Diffie-Hellman key-exchange solves the problem of sending secure information over an insecure channel.

Alice and Bob want to share messages over an insecured channel. They agree on a large prime p and an integer $g \mod p$ of $\mathbb{F}_p^*$ $(2 \leq g \leq p-2)$. Alice chooses a secret integer a and privately computes $A \equiv g^a \bmod p$ while Bob chooses a secret integer b and privately computes $B \equiv g^b \bmod p$. Alice sends A to Bob and Bob sends B to Alice.

Alice then privately computes $B^{'} \equiv B^a \bmod p$ and Bob privately computes $A^{'} \equiv A^b \bmod p$. Note that A' and B' are actually equal since

$$A' \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B' \mod p.$$

The value $A' \equiv B'$ is the shared secret key. Below is a description of the exchange.


Public: p, g
Alice $A \equiv g^a \bmod p$
Bob $B \equiv g^b \bmod p$
Bob sends B to Alice.
Alice sends A to Bob.
Alice $B^a \equiv B' \bmod p$
Bob $A^b \equiv A' \bmod p$
Shared key $A' \equiv B'$

Suppose that Eve observes the exchange of A and B between Alice and Bob over an insecured channel. The only way that Eve could decipher the secret key $A' \equiv B'$ is to compute

$$g^a \equiv A \bmod p \quad \text{or} \quad g^b \equiv B \bmod p$$

without knowledge of a or b. Eve could feasibly compute all possible powers of g modulo p to find $A' \equiv B'$ if g and p are small enough. Thus, it is recommended to choose the prime p as a number of approximately 1000 bits ($p \approx 2^{1000}$). If Eve can solve the DLP, she could easily solve this problem. However, Eve is faced with an easier problem of solving what is known as the Diffie-Hellman Problem.

Theorem 13 (Diffie-Hellman Problem)**.** *Let p be a prime and $g \in \mathbb{Z}$. The Diffie-Hellman Problem (DHP) is the problem of computing the value $g^{ab} \bmod p$ from the known values of $g^a \bmod p$ and $g^b \bmod p$.*

This groundbreaking work provided an outline for implementing a cryptosystem that relied more heavily on the difficulty of solving a problem equivalent to solving the DLP.

3.3 RSA Cryptography

Even with the revolutionary concept of the Diffie-Hellman key exchange algorithm allowing two parties to publicly share information resulting in a private key, a fully-functioning public-key encryption scheme was not fully developed. The Diffie-Hellman key exchange allowed for randomly selected numbers to be securely transmitted over an insecure channel, but what about meaningful messages? It was not until the work of Rivest, Shamir, and Adelman was published in 1978 that the first public-key cryptosystem was developed. This cryptosystem, known by the initials of its creators, is currently still one of the most widely used forms of public-key encryption.

Part I: Creating RSA Key

Step 1: Bob generates two large primes (∼100 digits each) and computes both $n = pq$ and $\phi(n) = (p-1)(q-1)$ where n is the RSA modulus (which is ∼200 digits long).

Step 2: Bob selects a random integer $e \in \mathbb{Z}^+$, $1 < e < \phi(n)$ where $gcd(e, \phi(n)) = 1$; e is known as the RSA enciphering exponent. Then, he uses the Euclidean Algorithm to compute d, $1 < d < \phi(n)$ such that $de \equiv 1 \pmod{\phi(n)}$; d is known as the RSA deciphering element.

Step 3: Bob now publishes n and e while keeping $d, p, q,$ and $\phi(n)$ private.

Part II: Enciphering and Deciphering Algorithms

Let $m \in \mathcal{M}$ be the plaintext. Assume that m has a numerical value less than n and $gcd(m, n) = 1$.

Step 1: Alice obtains (n, e) from public key database.

Step 2: She encrypts m by computing $m^e \equiv c \pmod{n}$ and sends c to Bob.

Step 3: Bob receives c and computes $c^d \equiv (m^e)^d \equiv m^{ed} \equiv m \pmod{n}$.

If the plaintext message, m, has a numerical value $m \geq n$, there is the possibility that data could be lost between the encoding and decoding processes. Thus, a process known as *message blocking* in which the numerical equivalent of the plaintext is divided into blocks of equal length must be implemented. Message blocking may be achieved by choosing that unique integer ℓ such that $N^\ell < n < N^{\ell+1}$, then writing the message as blocks of ℓ-digit, base N integers (with zeros packed to the right of hte last block if necessary), and encipher each separately. In this way, since each block of plaintext corresponds to an element of $\mathbb{Z}/n\mathbb{Z}$ given that $N^\ell < n$; and since $n < N^{\ell+1}$, then each ciphertext message unit can be uniquely written as an $(\ell + 1)$-digit, base N integer in $\mathcal{C} = \mathbb{Z}/n\mathbb{Z} = \mathcal{M}$. For practical purposes, since the English alphabet has 26 letters, we use $N = 26$.

3.3.1 RSA Encryption Example

Alice wishes to send the message m: "Santa Claus uses code" to Bob. Bob chose $n = 14,785,217$, $\phi(n) = 14,777,520$ and $e = 17$ in creating his RSA private key. He then published $(n, e) = (14,785,217; 17)$ in a public key database. Alice obtains Bob's RSA public key from the database and proceeds to encode her message.

Alice breaks the plaintext into blocks:

$$11,881,376 = 26^5 \leq 14,785,217 \leq 26^6 = 308,915,776$$

So, break message into blocks of $l = 5$ digits or less. Since plaintext is 18 letters/digits, break into 4 blocks of 5 letters 0's packed on the end of the last block (the letter "A" has the numerical value of 0). Thus, the blocks to encode are: SANTA, CLAUS, USESC, ODEAA.

Alice then converts each block to number form, then to base 26.

$$\begin{aligned}
\text{SANTA} &= (18\ 0\ 13\ 19\ 0) \\
&= 18 \cdot 26^4 + 0 \cdot 26^3 + 13 \cdot 26^2 + 19 \cdot 26 + 0 \\
&= 8234850 \pmod{n} \\
\text{CLAUS} &= (2\ 11\ 0\ 20\ 18) \\
&= 2 \cdot 26^4 + 11 \cdot 26^3 + 0 \cdot 26^2 + 20 \cdot 26 + 18 \\
&= 1107826 \pmod{n} \\
\text{USESC} &= (20\ 18\ 4\ 18\ 2) \\
&= 20 \cdot 26^4 + 18 \cdot 26^3 + 4 \cdot 26^2 + 18 \cdot 26 + 2 \\
&= 1233494 \pmod{n} \\
\text{ODEAA} &= (14\ 3\ 4\ 0\ 0) \\
&= 14 \cdot 26^4 + 3 \cdot 26^3 + 4 \cdot 26^2 + 0 \cdot 26 + 0 \\
&= 6453096 \pmod{n}
\end{aligned}$$

She next encodes each block using $m^e \equiv c \pmod{n}$:

$$\begin{aligned}
8234850^{17} &\equiv 8485514 \pmod{14785217} \\
1107826^{17} &\equiv 10537779 \pmod{14785217} \\
12333494^{17} &\equiv 655742 \pmod{14785217} \\
6453096^{17} &\equiv 2958865 \pmod{14785217}
\end{aligned}$$

Convert each block into letter form in base 26.

$$\begin{aligned}
8485514 &= (18\ 14\ 20\ 23\ 21)_{26} = \text{SOUNY} \\
10537779 &= (23\ 1\ 14\ 11\ 5)_{26} = \text{XBOLF} \\
655742 &= (1\ 11\ 8\ 0\ 22)_{26} = \text{BLIAW} \\
2958865 &= (6\ 12\ 9\ 0\ 13)_{26} = \text{GMJAN}
\end{aligned}$$

Ciphertext Alice sent to Bob:

SOUNY   XBOLF   BLIAW   GMJAN

Bob will decode using $ed + x\phi(n) \equiv 1$ and his private key $d = -6954127 \equiv 7823393 \pmod{n}$. Bob takes the ciphertext sent to him by Alice, SOUNY XBOLF BLIAW GMJAN, and converts each block to numerical form, then to base 26.

$$\begin{aligned}
\text{SOUNY} &= (18\ 14\ 20\ 23\ 21)_{26} \\
&= 8485514 \pmod{n} \\
\text{XBOLF} &= (23\ 1\ 14\ 11\ 5)_{26} \\
&= 10537779 \pmod{n} \\
\text{BLIAW} &= (1\ 11\ 8\ 0\ 22)_{26} \\
&= 655742 \\
\text{GMJAN} &= (6\ 12\ 9\ 0\ 13)_{26} \\
&= 2958865
\end{aligned}$$

Next, Bob decodes each block using $d = 7823393$.

$$\begin{aligned}
8485514^{7823393} &= 8234850 \pmod{n} \\
10537779^{7823393} &= 1107826 \pmod{n} \\
655742^{7823393} &= 12333494 \pmod{n} \\
2958865^{7823393} &= 6453096 \pmod{n}
\end{aligned}$$

Bob then converts each block to base 26. With the first block, 8234850, converting to base 26 can be accomplished using the following algorithm:

$$\begin{aligned}
8234850 &= 316725 \cdot 26 + 0 \\
316725 &= 12181 \cdot 26 + 19 \\
12181 &= 468 \cdot 26 + 13 \\
468 &= 18 \cdot 26 + 0 \\
18 &= 0 \cdot 26 + 18
\end{aligned}$$

Thus, $8234850 = (18\ 0\ 13\ 19\ 0)_{26}$ Using the same process, we convert the remaining blocks to base 26:

$$\begin{aligned}
8234850 &= (18\ 0\ 13\ 19\ 0)_{26} \\
&= \text{SANTA} \\
1107826 &= (2\ 11\ 0\ 20\ 18)_{26} \\
&= \text{CLAUS} \\
1233494 &= (20\ 18\ 4\ 18\ 2)_{26} \\
&= \text{USESC} \\
6453096 &= (14\ 3\ 4\ 0\ 0)_{26} \\
&= \text{ODEAA}
\end{aligned}$$

and thus Bob recovers "Santa Claus uses codeaa". At this point, the additional "a's" that were added to create the equal block sizes are omitted and the message reads "Santa Claus uses code".

3.3.2 Attacks on RSA

Even with security of RSA ensured by difficulty of factorization, attacks at breaking the cryptosystem have not only been attempted, but have succeeded. In many cases, it is not an inherent weakness of the cryptosystem, but rather poor implementation in practice.

Common Modulus Protocol Failure

Alice and Bob have the same RSA modulus, n, but different enciphering exponents, say e_A and e_B. Assume that $gcd(e_A, e_B) = 1$. A third party, Sam, wishes to send the same message, m, to both Alice and Bob. Sam obtains the enciphering exponents and RSA modulus from the public key database, encodes the plaintext, and sends the ciphertext Alice and Bob, respectively. Eve intercepts both ciphertexts, $m^{e_A} \bmod n$ and $m^{e_B} \bmod n$.

Eve can obtain Alice and Bob's enciphering exponents from the public key database. Since e_A and e_B are relatively prime, $e_A x + e_B y = 1$ can be solved for x and y using the Euclidean Algorithm. Eve can then calculate

$$\begin{aligned}
((m^{e_A})^x (m^{e_B})^y) \bmod n &= (m^{e_A x})(m^{e_B y}) \bmod n \\
&= m^{e_A x + e_B y} \bmod n \\
&= m^1 \bmod n
\end{aligned}$$

and recover the plaintext, m, without knowledge of the deciphering exponent or having to factor the RSA modulus. A protocol failure is not an inherent weakness of the cryptosystem but rather poor implementation of the cryptosystem.

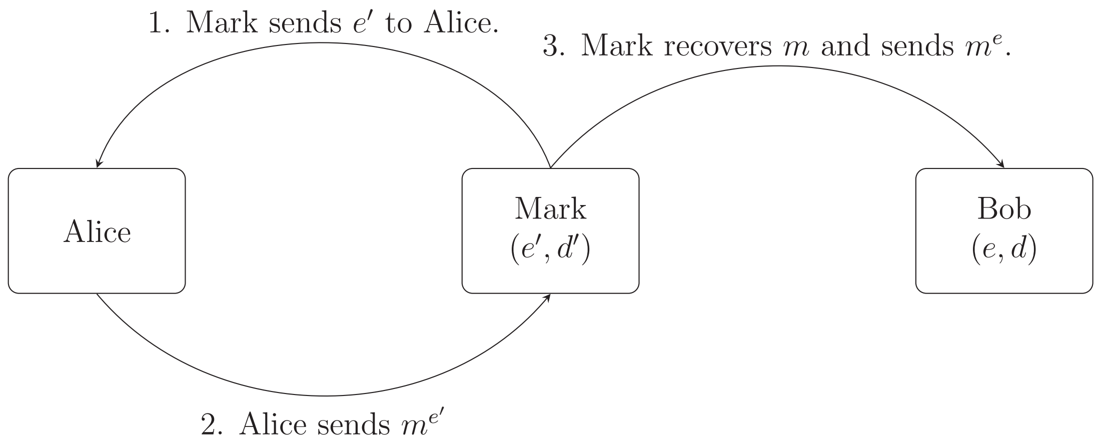
Man-in-the-Middle Attack

In this type of attack, Alice and Bob's messages are intercepted by an active, third-party attacker, Mark. Mark can substitute his own messages from Alice to Bob (after intercepting the message originally) in such a way that Alice and Bob are not aware that they are being attacked. A way to circumvent this problem is to require message authentication and digital signatures, which we describe next.

3.3.3 RSA Authentication Protocol

Thus far, we have discussed the steps that Bob and Alice may take to ensure that the messages can be sent securely so that it is computationally infeasible for a malicious third party to decrypt the message. However, before two parties can interact in sending messages, it is essential that the legitimacy of the entities involved in the protocol be verified, as well. This process is known as *authentication*.

Suppose that Alice wishes to send a message m to Bob. She requests Bob's public enciphering key, e, but Mark (a malicious third party impersonating Bob) sends to Alice his enciphering key, e', instead. Alice encrypts her message using e' and sends the message $m^{e'}$ to Mark. Mark uses his private deciphering key, d', to compute $(m^{e'})^{d'} = m$. He now relays the message m to Bob using Bob's public enciphering key, e. Neither Alice nor Bob are aware that the message has been intercepted by Mark since there was no authentication protocol set forth to ensure that communication occurs only between the intended, legitimate parties. This attack, known as an *impersonation attack*, is described by the diagram below.


1. Mark sends e' to Alice.
3. Mark recovers m and sends m^e.
Alice
Mark (e', d')
Bob (e, d)
2. Alice sends $m^{e'}$

The following is a description of the implementation of a digital signature scheme for RSA as a form of authentication protocol.

RSA Digital Signature Scheme

Bob publishes his public key $E_B = (n, e)$ and keeps his deciphering exponent, d, private. Alice sends Bob a message m and so encrypts using Bob's public key. Bob receives $c \equiv m^e \bmod n$. Bob recovers m by computing $c^d \equiv m \bmod n$.

Alice's public RSA key, $E_A = (n', e')$ is made available in the public key database while her deciphering exponent d' is kept secret.

Signature Stage Alice wants to send a signature m. She computes $(m^{d'})^e \bmod n$ and sends it to Bob.

Verification Stage Bob recovers $m^{d'}$ by computing $((m^{d'})^e)^d \equiv m^{d'} \bmod n$. Bob recovers m by computing $(m^{d'})^{e'} \bmod n'$. Since $e'd' \equiv 1 \bmod n$, Bob recovers Alice's signature, m.

3.4 ElGamal Public Key Cryptographic System

While RSA is undoubtedly one of the best-known public-key encryption methods, it is not the only public-key cipher. The security of RSA is based on the difficulty of integer factorization. The following system was developed by Egyptian mathematician Taher ElGamal and described in his 1985 paper. The ElGamal Cryptosystem bases its security on the intractability of the discrete log problem.

The ElGamal Cryptosystem

We assume that Alice wishes to send a message, m, to Bob where $m \in \{0, 1, \ldots, p-1\}$.

Key Generation

1. Bob chooses a large prime p and a primitive root α of $\mathbb{F}_p^*$.
2. Bob chooses an integer a where $2 \leq a \leq q-2$ and computes $\alpha^a \pmod{p}$.
3. Then, Bob's public key is $(p, \alpha, \alpha^a \pmod{p})$ and his private key is a.

Enciphering Stage

1. Alice obtains Bob's public key $(p, \alpha, \alpha^a \pmod{p})$ from database.
2. She chooses a random integer $2 \leq b \leq p-2$.
3. She computes $\alpha^b \pmod{p}$ and $m_i(\alpha^a)^b \equiv m_i\alpha^{ab} \pmod{p}$ where m_i is the plaintext broken into blocks of appropriate length.
4. She sends the ciphertext $c \equiv (\alpha^b, m\alpha^{ab}) \pmod{p}$ to Bob.

Deciphering Stage

1. Bob uses his private key a to compute $(\alpha^b)^{-a} \equiv (\alpha^b)^{p-1-a} \pmod{p}$
2. Then, he deciphers c by computing

$$(\alpha^b)^{-a} \cdot m \cdot \alpha^{ab} \equiv m \cdot \alpha^{ab-ab} \equiv m \pmod{p}.$$

3.4.1 ElGamal Example

Alice wishes to send the message "Thanks for the memories" to Bob. Bob had chosen prime $p = 178,301,561$ with primitive root $\alpha = 6$. Bob's private key is $a = 13$ and Alice chooses $b = 71$. Alice accesses Bob's public key from a public key database:

$$(p, \alpha, \alpha^a \bmod p) = (178301561, 6, 44680063).$$

To begin the enciphering process, Alice breaks the plaintext, m, into blocks:

$$11,881,376 = 26^5 \leq 178,301,561 \leq 26^6 = 308,915,776$$

and thus breaks her plaintext into blocks of length $\ell = 5$ letters or less. Alice then translates each of the blocks to numerical form, the from base 26 mod$p$.

$$\begin{aligned}
\text{THANK} &= (19\ 7\ 0\ 13\ 10)\\
&= 19 \cdot 26^4 + 7 \cdot 26^3 + 0 \cdot 26^2 + 13 \cdot 26 + 10\\
&= 8805924 \bmod p\\
\text{SFORT} &= (18\ 5\ 14\ 17\ 19)\\
&= 18 \cdot 26^4 + 5 \cdot 26^3 + 14 \cdot 26^2 + 17 \cdot 26 + 19\\
&= 8323373 \bmod p\\
\text{HEMEM} &= (7\ 4\ 12\ 4\ 12)\\
&= 7 \cdot 26^4 + 4 \cdot 26^3 + 12 \cdot 26^2 + 4 \cdot 26 + 12\\
&= 3277364 \bmod p\\
\text{ORIES} &= (14\ 17\ 8\ 4\ 18)\\
&= 14 \cdot 26^4 + 17 \cdot 26^3 + 8 \cdot 26^2 + 4 \cdot 26 + 18\\
&= 6701986 \bmod p
\end{aligned}$$

Alice then calculates $(\alpha^a)^b \bmod p = (44680063)^{71} \bmod 178301561 = 97835431$. She encodes each plaintext block, m_i by calculating $m_i(\alpha^a)^b \equiv \alpha^{a \cdot b} \bmod p$.

$$\begin{aligned}
8805924 \cdot (\alpha^a)^b \bmod p &= 158233247\\
&= (13\ 8\ 6\ 20\ 22\ 3)_{26}\\
&= \text{NIGUWD}\\
8323373 \cdot (\alpha^a)^b \bmod p &= 82188785\\
&= (6\ 23\ 22\ 5\ 1\ 3)_{26}\\
&= \text{GXWFBD}\\
3277364 \cdot (\alpha^a)^b \bmod p &= 126115730\\
&= (10\ 15\ 25\ 11\ 19\ 0)_{26}\\
&= \text{KPZLTA}\\
6701986 \cdot (\alpha^a)^b \bmod p &= 1096175\\
&= (2\ 10\ 9\ 14\ 15)_{26}\\
&= \text{CKJOP}
\end{aligned}$$

Alice then sends to Bob the ciphertext:

$$c = (\alpha^b, m \cdot \alpha^{a \cdot b}) = (150151472, \text{NIGUWD GXWFBD KPZLTA CKJOP})$$

Bob obtains Alice's ciphertext and proceeds to decode by first calculating

$$(\alpha^b)^{p-1-a} \bmod p = (150151472)^{178301547} \bmod p = (150151472)^{3^3 \cdot 6603761} \bmod p$$

$$(\alpha^b)^{-a} \bmod p = (150151472)^{-13} \bmod 178301561 = 104796284$$

Next, Bob deciphers each block by computing

$$(\alpha^b)^{-a} \cdot m \cdot \alpha^{ab} \equiv m \cdot \alpha^{ab-ab} \equiv m \bmod p$$

For NIGUWD:

$$\begin{aligned}
158233247 \cdot 104796284 \bmod p &= 8805924\\
&= (19\ 7\ 0\ 13\ 10)_{26}\\
&= \text{THANK}
\end{aligned}$$

For GXWFBD:

$$\begin{aligned} 82188785 \cdot 104796284 \bmod p &= 8323373 \\ &= (18\ 5\ 14\ 17\ 19)_{26} \\ &= \text{SFORT} \end{aligned}$$

For KPZLTA:

$$\begin{aligned} 126115730 \cdot 104796284 \bmod p &= 3277364 \\ &= (7\ 4\ 12\ 4\ 12)_{26} \\ &= \text{HEMEM} \end{aligned}$$

For CKJOP:

$$\begin{aligned} 1096175 \cdot 104796284 \bmod p &= 6701986 \\ &= (14\ 17\ 8\ 4\ 18)_{26} \\ &= \text{ORIES} \end{aligned}$$

Thus, Bob has recovered Alice's plaintext "Thanks for the memories".

3.4.2 ElGamal Authentication Protocol

As is the concern with any communication that takes place over the expanse of electronic communication, one wishes to ensure that they are indeed communicating directly with the intended recipient rather than an intermediate or impostor. The following is a description of implementing authentication while utilizing the ElGamal cipher.

El Gamal Digital Signature Scheme

Setup: Alice is sending the message $m \in \mathbb{F}_p^*$ to Bob. Her public key generated using the ElGamal key generation algorithm is (p, α, y), her private key is a, and $y \equiv \alpha^a \pmod{p}$.

Signing: Alice performs each of the following.

- Select random $r \in (\mathbb{Z}/n\mathbb{Z})^*$.
- Compute $\beta \equiv \alpha^r \pmod{p}$
- Compute $\gamma \equiv (m - \alpha\beta)r^{-1} \pmod{p-1}$
- For $k = (p, \alpha, a, y)$, the signed message $sig_k(m, r) = (\beta, \gamma)$ is sent to Bob.

Verification: Bob performs each of the following.

- Using Alice's public key, (p, α, y), verify that $\beta \in \mathbb{F}_p^*$ and reject if not.

- Compute $\delta \equiv y^{\beta}\beta^{\gamma} \pmod{p}$
- Compute $\sigma \equiv \alpha^{m} \pmod{p}$
- $ver_k(m,(\beta,\gamma)) = 1$ if and only if $\sigma \equiv \delta \pmod{p}$. Otherwise, reject.

3.5 Massey-Omura Cryptosystem

A related encryption scheme that is not truly public-key (or private-key!) is known as a *three-pass protocol* uses neither private nor public keys. In this encryption scheme, let p be a prime, $n \in \mathbb{N}$, and m be an element of the multiplicative group $\mathbb{F}_{p^n}$.

1. Alice and Bob independently select random $e_A, e_B \in \mathbb{Z}$ where $2 \leq e_A, e_B < p^n - 1$ and $\gcd(e_a, p^n - 1) = 1 = \gcd(e_b, p^n - 1)$.

2. Alice and Bob compute $d_A \equiv e_A^{-1} \bmod p^n - 1$ and $d_B \equiv e_B^{-1} \bmod p^n - 1$, respectively, using the extended (reverse) Euclidean Algorithm.

3. Alice sends m^{e_A} to Bob.

4. Bob sends $m^{e_A e_B}$ back to Alice.

5. Alice sends $m^{e_A e_B d_A} = m^{e_B}$ to Bob, since $\gcd(e_A, d_A) = 1$.

6. Bob computes $m^{e_B d_B} = m$ since $\gcd(e_B, d_B) = 1$.

A disadvantage to this system is the requirement that three separate transmissions between Alice and Bob must take place prior to the message being successfully sent. Without further protection, this scheme is also susceptible to the man-in-the-middle attack. However, a great advantage to the scheme is that neither shared public nor shared private keys are necessary to utilize the scheme. New private keys can be used each time. Yet, the system is secure since prior to Bob's receiving $m^{e_A e_B d_A}$, he is faced with the DLP and could not decipher Alice's message. At each stage, Eve cannot decipher m since she does not have any of e_A, e_B, d_A, d_B and would be faced with the DLP.

4 Difficulty of Factorization

As noted throughout, the beauty of a public-key cryptosystem lies not in the secrecy in which the encryption is developed, but rather in the ability to make available the method and public keys while still maintaining security based on the sheer difficulty of large integer factorization. One method of factorization, Pollard's $p-1$ method, was discussed earlier in the paper. The following is a description of some "sieve methods" for factoring large integers.

4.1 Quadratic Sieve

Definition 8 (Quadratic Residue)**.** *Let p be an odd prime. We say that an integer, n, $p \nmid n$ is a quadratic residue modulo p if there is an $x \in \mathbb{Z}$ such that $n \equiv x^2 \bmod p$.*

Definition 9 (Smooth numbers)**.** *An integer n is called B-smooth if all of its prime factors are less than or equal to B.*

Definition 10 (Factor Base)**.** *The set of primes less than B is called the factor base.*

Credit for the development of the Quadratic Sieve goes to Carl Pommerance, 1981. Goal: Factor an integer, n. Let

$$k \approx \sqrt{e^{\sqrt{\ln(n)\ln(\ln n)}}} = \left\lfloor \sqrt{e^{\sqrt{\ln(n)\ln(\ln n)}}} \right\rfloor$$

1. Choose a factor base, $\mathscr{B} = \{p_1, p_2, \ldots, p_k\}$ such that $\frac{n}{p} = 1$.

2. For each $j \geq 0$, let $t = \pm j$, compute $y_t = (\lfloor \sqrt{n} \rfloor + t)^2 - n$ until $k+2$ such values are found to be p_k-smooth. For each such t, factor
$$y_t = \pm \prod_{i=1}^{k} p_i^{a_{it}}$$
and form a binary $(k+1)$ tuple, $v_t = (v_{0t}, v_{1t}, \ldots, v_{kt})$ for each $0 \leq i \leq k$, $v_{it} = a_{it} \bmod 2$ and $v_{0t} = 0$ if $y_t > 0$ and $v_{0t} = 1$ if $y_t < 0$.

3. Obtain a set, S, of the values of t found in step 2 such that for each $i = 0, 1, \ldots, k$,
$$\sum_{t \in S} v_{it} \equiv 0 \bmod 2.$$

In such case,

$$x^2 = \prod_{t \in S} x_t^2 = \prod_{t \in S} y_t = y^2 \bmod n.$$

The quadratic sieve method is still the preferred algorithm for factoring numbers with 50-100 digits. For larger numbers, however, the General Number Field Sieve is the most efficient method known to date. As an incentive to progress the techniques of large number factorization, the RSA Factoring Challenge was proposed in 1991 with a list of RSA moduli (denoted by RSA-XXXX, based on the number of bits in the number). Due to the significant advances in factoring made over the following decade, the challenge was cancelled and prize offers retracted in 2007.

The largest number factored to date is the RSA-768 (a 232 decimal-digit number) and was announced in early 2010. Even though the joint effort of 6 research institutions and 2000 CPU years ($\approx$ 3 calendar years), the factorization has confirmed the concerns of security analysts that the standard of using RSA keys with 1024 bits, and it is time to increase security to using a standard of keys with 2048 (2^{11}) bits. It is assumed that keys of this length will remain unfactored for the next several decades.

4.2 Running Time of Computing Discrete Logs

As described earlier, the security of RSA lies in the difficulty of large integer factorization. The security of the ElGamal cryptosystem, on the other hand, lies in the sheer difficulty of solving the DLP. Depending on the method used for solving a DLP, one may be faced with an algorithm that runs in exponential time (rather than polynomial time). To demonstrate the running time for an algorithm, in this case, the *index calculus method*, which can solve the DLP in

$$L(N) = e^{\sqrt{\ln N \ln \ln N}}$$

operations (addition/subtraction, multiplication/division, and exponentiation). For this example, assume that our computer performs one billion operations per second.

How long would it take to solve the DLP if we must perform $N = 2^{100}$ operations?

$$L(2^{100}) = e^{\sqrt{\ln 2^{100} \ln \ln 2^{100}}} \approx 2.7802 \times 10^7$$

and thus

$$\frac{2.7802 \times 10^7 \text{ operations}}{10^9 \text{ operations per second}} \approx 0.0278 \text{ sec} < 1 \text{ sec}.$$

Now, consider a computation requiring $N = 2^{1000}$ operations.

$$L(2^{1000}) = e^{\sqrt{\ln 2^{1000} \ln \ln 2^{1000}}} \approx 1.75217 \times 10^{29} \text{ operations}$$

and thus

$$\frac{1.75217 \times 10^{29} \text{ operations}}{10^9 \text{ operations per second}} \approx 1.751 \times 10^{20} \text{ sec} \approx 5.5 \text{ trillion years.}$$

Clearly, this computation will not be performed in our lifetime. Unless significant advances are made in the foreseeable future, we expect our current public-key encryption schemes to remain secure and usable for quite some time. However, we should never fall into a state of complacency and remain ever-vigilant to possible attacks while developing new techniques of encryption.

5 References

References

[1] Diffie, W., and Hellman M.E., *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644 – 654.

[2] Elgamal, T., *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE Transactions on Information Theory, Volume: 31, Issue: 4, Jul 1985, pp 469 – 472.

[3] Hoffstein J., Pipher J., and Silverman J.H., *An Introduction to Mathematical Cryptography*, Springer, New York, NY, 2008.

[4] Mollin R.A., *RSA and Public-Key Cryptography*, Chapman & Hall/CRC, Boca Raton, FL, 2003.

[5] Pomerance, C., *Analysis and Comparison of Some Integer Factoring Algorithms.* Computational Methods in Number Theory, Part I, H.W. Lenstra, Jr. and R. Tijdeman, eds., Math. Centre Tract 154, Amsterdam, 1982, 89 – 139.

[6] Rivest, R., Shamir, A., Adleman, L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM 21 (2), 1978, pp 120 – 126.