A Survey of Cryptographic Applications of Number Theory

Submitted by,
Brian Greenberg

Mathematics 124
December 13, 2002

Contents

1 Introduction: Number Theory and Cryptography

In this paper, we explore two forever married disciplines: *Number Theory*, the study of the natural numbers, $\mathbb{N}$, and *Cryptography*, the study of communication with encrypted messages. Why do we care about these topics? It is somewhat easier to answer this question with regard to cryptography. How many times per day do you check your email? If you're anything like me, probably many. Each time you check your email, you must be *authenticated* with the mail server before it delivers your mail. Imagine what could happen if this authentication did not exist. Anyone could read your private email. Anyone could access your credit card number. Anyone could access your social security number. It is clear that private information such as this needs to be protected somehow. This protection is one particular application of cryptography. Why do we care about number theory? As discussed above, many of the fundamental ideas of encryption and decryption rely on the beautiful properties of the natural numbers, especially the primes. In this paper, we will explore the intersection between these two fields. We will first introduce some convenient terminology and notation that we will use throughout our explanation. We will then introduce two well-known public-key encryption protocols and discuss some variants of these protocols. Finally, we will show how to use public-key encryption for other applications, such as digital signatures and zero-knowledge proofs. We will continually refer to concepts learned in an introductory number theory course, as each protocol relies heavily on algebraic number theory.

Throughout our discussion, we will use the ongoing saga of our old friends Michael and Nikita to motivate each of the different protocols. Recall the situation in which we left Michael and Nikita. They had orchestrated a plan in which Nikita feigned capture as a hostage by the enemy, but in reality, Michael and Nikita were secretly communicating. When Section One's master hacker, Birkoff, discovered encrypted messages being sent back and forth between Michael and Nikita, Madeline suspects foul play. Now, Michael and Nikita needed a secure way to communicate messages, knowing that their enemies were listening in on their communication channel. They found that they were able to use the Diffie-Hellman key exchange to successfully exchange a public-key, which they would henceforth use for secure communications. This brings us to the present: Nikita is hiding out in an underground bunker, with only a computer, food, and water at her disposal. Michael has seemed to disappear off the face of the earth according to Section One, but he has actually changed his appearance through a series of plastic surgeries with Los Angeles' finest doctors, and now roams the streets with his laptop computer and an ample supply of money. He and Nikita communicate by text messaging systems over a secure channel and are in the process of planning their future attack on Section One. Now that they have the Diffie-Hellman key exchange at their disposal, they set out to encrypting their messages...

2 Rabin Public-Key Encryption

Michael and Nikita consider using the popular RSA cryptosystem, but after hearing that even though most people believe breaking RSA to be as hard as factoring, no one has proven it yet [5]. They thus decide that a good place to start is the first *provably* secure public-key cryptosystem, developed by Rabin in 1979 [6].

Most public-key cryptosystems are based on the presumed difficulty of certain number theoretic problems. That is, a crytographer develops a protocol and proves that breaking his protocol is equivalent to solving some "hard" problem, where the hardness of a problem is governed by its computational complexity. For example, most readers should be aware of the RSA cryptosystem, the security of which is based on the computational intractability of factoring large integers into their prime factorizations. For reference in this paper, we state the problem formally:

Problem 1 (`FACTORIZATION`) *Given a positive integer n, compute its prime factorization. That is, find $p_1, \ldots, p_k$ and $e_1, \ldots, e_k$ such that $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, where the p_i are distinct and each e_i is a positive integer.*

The reader should be somewhat familiar with the difficulty of Problem 1; any known algorithm requires exponential time to compute the factorization of n. The Rabin protocol is based on the problem of computing square roots of elements in $\mathbb{Z}_n$ for a composite n. We state the problem formally as follows:

Problem 2 (`SQROOT`) *Given a composite integer n and a quadratic residue y modulo n, find a square root of y modulo n, that is, find an $x \in \mathbb{Z}_n$ such that $x^2 \equiv y \bmod n$.*

We state without proof the following fact:

Fact 3 `SQROOT` *and* `FACTORIZATION` *are computationally equivalent.*

Thus, since `FACTORIZATION` is widely believed to be a hard problem, `SQROOT` is also considered a hard computational problem (in fact, equally as hard as `FACTORIZATION`).

2.1 Creation of the Public Key

The first step in applying the Rabin encryption protocol is choosing a public key for each party. Michael and Nikita want to send messages back and forth, so they will each create a public key. How is this public key chosen? Let us examine in detail how Nikita will choose her public key, and then Michael can apply the same procedure. Nikita will choose two large primes p and q and compute their product $n = p \cdot q$. Michael will do the same, and their public keys will be n_N and n_M, respectively. It will behoove Michael and Nikita to choose primes p and q such that $p \equiv q \equiv 3 \bmod 4$, and we will see why in a few moments. Their *private keys*, which they each must keep secret, will be (p_N, q_M) and (p_M, q_M), respectively. Formally, the algorithm for key generation runs as follows [6]:

```
Procedure RABIN-KEY:
   Choose two large primes p and q
   Compute n = pq
   Public key is n
   Private key is (p, q)
End procedure
```

2.2 Encryption and Decryption

Now that Michael and Nikita have computed their public keys, all that lies ahead is to encrypt their message. Suppose Nikita wants to notify Michael of her whereabouts, so she decides to send Michael her location with respect to longitude and latitude. Let us suppose her message, X, is an element of the group $\mathbb{Z}_{n_M}^*$. This assumption is valid for two reasons. First, it is easy to translate any text message into a number X, and second, even if $X \notin \mathbb{Z}_{n_M}^*$, we can simply truncate it into individual messages $X_1, \ldots, X_k$, where each $X_i \in \mathbb{Z}_{n_M}^*$ [5]. Now all Nikita has to do in order to encrypt her message is to look up Michael's public key n_M and compute $Y = X^2 \bmod n_M$. The result Y will be the ciphertext, which Nikita may now send to Michael. Formally, the algorithm runs as follows [6]:

```
Procedure ENCRYPT-RABIN(X):
  Find appropriate public key n
  Compute Y = X^2 mod n
  Return Y
End procedure
```

Now how can Michael, upon receiving $Y = X^2 \bmod n_M$, recover Nikita's message X? We'll need the following result:

Claim 4 *Suppose X, Y, and $n_M = pq$ are defined as above. Then*

$$\begin{aligned} Y &= X^2 \bmod n_M & (1)\\ Y &= X^2 \bmod p \text{ and } Y = X^2 \bmod q & (2) \end{aligned}$$

are equivalent.

Proof: Note that by definition of modulo arithmetic, (1) implies that $n_M \mid (X^2 - Y)$ and (2) implies that $p \mid (X^2 - Y)$ and $q \mid (X^2 - Y)$. It is clear that if $n_M \mid (X^2 - Y)$ then so does each of n_M's divisors, including p and q. By unique factorization, $p, q \mid (X^2 - Y)$ implies that $n_M \mid (X^2 - Y)$, since $p, q \mid (X^2 - Y)$ means that $X^2 - Y = pk = ql$ for $k, l \in \mathbb{Z}$, and $q \not| p$, so $q \mid k$. Thus $pq \mid pk$ and $n_M \mid (X^2 - Y)$. Thus the claim is proven.

■

Let p and q be Michael's private keys. Upon receiving Y, Michael must compute the square root of Y modulo p and q. Since Michael and Nikita had the foresight to choose their primes p and q to be 3 mod 4, Michael can do this easily:

$$U \equiv \sqrt{Y} \equiv Y^{\frac{(p+1)}{4}} \bmod p$$
$$V \equiv \sqrt{Y} \equiv Y^{\frac{(q-1)}{4}} \bmod q$$

By Claim 4, Michael, upon receiving Y, has a system of two congruences:

$$\begin{aligned} X &\equiv U \bmod p & (3)\\ X &\equiv V \bmod q & (4) \end{aligned}$$

Michael will need the Chinese Remainder Theorem, which we state without proof.

Theorem 5 (Chinese Remainder Theorem) *Let $a, b \in \mathbb{Z}$ and $n, m \in \mathbb{N}$ with n and m relatively prime. Then there is some $x \in \mathbb{Z}$ such that*

$$\begin{aligned} x &\equiv a \bmod m\\ x &\equiv b \bmod n \end{aligned}$$

and x is unique modulo mn.

Armed with Theorem 5 and congruences (3) and (4) above, Michael can now solve for X, the original message. We will see in a moment, however, that Michael can run into trouble here.

2.3 The Blum Variant

Now that Michael has computed the square root of Y, he is justifiably confused. When he fed Y into his number-to-text transformation machine, the text was jibberish. After scratching his head for a while, Michael finally realized the major flaw in Rabin's protocol:

Theorem 6 *Suppose $x, n \in \mathbb{Z}$ with $(x, n) = 1$, and $n = pq$ for primes p and q, and $y = x^2 \bmod n$. Then there are exactly four quadratic residues x_1, x_2, x_3, x_4 of y modulo n.*

Proof: By Claim 4, we have $y = x^2 \bmod p$ and $x^2 \bmod q$. Now let a be an integer with $1 \leq a < p$ and $a^2 \equiv y \bmod p$. Now let $a' = p - a$. Then $a'^2 \equiv y \bmod p$. Now let b be another integer with $1 \leq b < q$ and $b^2 \equiv y \bmod q$. Again, let $b' = q - b$, then we also have $b'^2 \equiv y \bmod q$. Then we have four options:

- x_1 where $x_1 \equiv a \bmod p$ and $x_1 \equiv b \bmod q$
- x_2 where $x_2 \equiv a \bmod p$ and $x_2 \equiv b' \bmod q$
- x_3 where $x_3 \equiv a' \bmod p$ and $x_3 \equiv b \bmod q$
- x_4 where $x_4 \equiv a' \bmod p$ and $x_4 \equiv b' \bmod q$

The fact that each of these x_i exists follows from Theorem 5 above. ■

What is Michael to do? It turns out that he has recovered just *one* of the four possible residues of Y. Scratching his head a bit more, Michael finally remembers the Blum disambiguation of Rabin's protocol [7]. Blum answers the question of how to distinguish which of the four possible residues X_1, X_2, X_3, X_4 is the intended message, X. It turns out that a simple decision on the part of the sender will result in disambiguation. Blum tweaks the protocol in two ways:

1. The public keys must be so called *Blum integers*, that is, a public key $n = pq$ is a Blum integer if $p, q \equiv 3 \bmod 4$.
2. The message, X, must be such that $X < \frac{n}{2}$ and $\left(\frac{X}{n}\right)_J = 1$. Recall that $\left(\frac{X}{n}\right)_J$ and $\left(\frac{X}{n}\right)_L$ denote the Jacobi and Legendre symbols of X with respect to n, respectively.[1]

What do these parameters do for Michael and Nikita? It turns out that exactly one of the four possible quadratic residues of Y will have property (2) above.

Theorem 7 *Of the X_1, X_2, X_3, X_4 with $X_i^2 \equiv Y \bmod n_M$, exactly one X_i is such that $X_i < \frac{n_M}{2}$ and $\left(\frac{X}{n_M}\right)_J = 1$.*

Proof: By Theorem 6, we have

$$\begin{aligned} a^2 &\equiv Y \bmod p \\ a'^2 &\equiv Y \bmod p \\ b^2 &\equiv Y \bmod q \\ b'^2 &\equiv Y \bmod q \end{aligned}$$

[1] This may seem like an unreasonable specification, but it turns out to be relatively easy to express any message this way.

We also know that exactly one of $\{a, a'\}$ is a quadratic residue modulo p and exactly one of $\{b, b'\}$ is a quadratic residue modulo q. So without loss of generality, let

$$\begin{aligned} \left(\frac{a}{p}\right)_J &= 1 \\ \left(\frac{a'}{p}\right)_J &= -1 \\ \left(\frac{b}{q}\right)_J &= 1 \\ \left(\frac{b'}{q}\right)_J &= -1 \end{aligned}$$

Now, as in Theorem 6, if we consider X_1, we have $x_1 \equiv a \bmod p$ and $X_1 \equiv b \bmod q$.

$$\left(\frac{X_1}{n}\right)_J = \left(\frac{X_1}{p}\right)_L \left(\frac{X_1}{q}\right)_L = \left(\frac{a}{p}\right)_L \left(\frac{b}{q}\right)_L = 1 \cdot 1 = 1$$

Now if we consider X_4, we have $X_4 \equiv a' \bmod p$ and $X_4 \equiv b' \bmod q$, we have

$$\left(\frac{X_4}{n}\right)_J = \left(\frac{X_4}{p}\right)_L \left(\frac{X_4}{q}\right)_L = \left(\frac{a'}{p}\right)_L \left(\frac{b'}{q}\right)_L = (-1) \cdot (-1) = 1$$

Thus we have

$$\begin{aligned} \left(\frac{X_1}{n}\right)_J &= 1 \\ \left(\frac{X_4}{n}\right)_J &= 1 \\ \left(\frac{X_2}{n}\right)_J &= -1 \\ \left(\frac{X_3}{n}\right)_J &= -1 \end{aligned}$$

And since $X_1 = n - X_4$, only one of $\{X_1, X_4\}$ can be less than $\frac{n}{2}$. ■

Armed with this knowledge, Michael and Nikita begin using the Rabin public-key encryption over their secure channel of communication. Fearing their adversaries from Section One, they decide to remain in their respective hiding places until they decide on a definite plan...

2.4 Breaking Rabin PKE

After some time, Madeline is able to track down Michael's whereabouts in L.A. Instead of capturing him, Madeline and her cohorts at Section One decide that it would be more advantageous if they could instead listen in on conversations between Michael and Nikita. One day, while Michael is out eating lunch, Madeline breaks into his apartment and sits down at his desk. Seeing notes and papers on Rabin public-key encryption strewn about, she surmises that Michael and Nikita are using this protocol. Using his computer's encryption and decryption software, Madeline is able to steal Michael's private key (p, q). She then leaves the apartment.

How was Madeline able to recover Michael's private key using just his encryption and decryption software? She adopted the following strategy [5]. She chose messages $X \in \mathbb{Z}_p^*$ at random until she found an X such that $X < \frac{n_M}{2}$ and $\left(\frac{X}{n_M}\right)_J = -1$. In general, she would succeed in two tries, since each X has a $1/2$ probability of being a quadratic residue. She then used Michael's encryption software to encrypt X and obtain $Y = X^2 \bmod n_M$. She then fed Y to Michael's decryption software. But his decryption program returned not X', but X, which was another square root of Y modulo n_M, and the *legal* Blum message (i.e. the X_i with $\left(\frac{X_i}{n_M}\right)_J = 1$). Thus, Madeline had two distinct square roots of Y modulo n_M. Since Michael and Nikita were using the Blum variant of Rabin PKE, Madeline knew that n_M was a Blum integer. Finaly, she considered the following fact:

Fact 8 *Let $x, y, z \in \mathbb{Z}$. If $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$, then $gcd(x - y, n)$ is a nontrivial factor of n.*

Proof: Suppose x, y, z are defined as in the claim above, that is, $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$. Then $n \mid (x^2 - y^2)$ by the definition of mod, and thus $n \mid (x - y)(x + y)$. But since $x \not\equiv \pm y \bmod n$, we know that $n \not\mid (x - y)$ and $n \not\mid (x + y)$. Thus $gcd(x - y, n)$ must be either p or q, a nontrivial factor of n. ■

Now since Madeline had both X' and X, and $X \not\equiv \pm X' \bmod n_M$, and $X'^2 \equiv X^2 \equiv Y \bmod n_M$, she used Fact 8 to recover p. Now holding n_M and p, she was able to quickly find q by simple division. She then left the apartment with Michael's private key.

When Michael returns to his apartment, he sees that his desk chair has been moved and that Madeline has left footprints behind on his carpet. Sensing a security breach, He text-messages Madeline to abort the Rabin protocol. Their communications have been foiled, and they need a new protocol...

3 ElGamal Public-Key Encryption

Michael and Nikita each purchase a secure cell phone, with which they can communicate for the time being. Fearing that their cell phone communications may become insecure, they use the phones only to discuss a new encryption protocol for their text-based messaging. Remembering that Madeline's attack on their Rabin protocol involved properties of quadratic residues, Michael suggests that they base their protocol on a different computational problem. Nikita suggests that they consider the discrete logarithm problem, reminding Michael that it is formally stated as follows:

Problem 9 (`DISCRETE-LOG`) *Given a prime p, a generator $g \in \mathbb{Z}_p^*$, and an element $a \in \mathbb{Z}_p^*$, find the integer x such that $g^x \equiv a \bmod p$. x is referred to as the* discrete logarithm *of a to the base g, and is denoted $\log_g a$.*

Another problem related to both `DISCRETE-LOG` and ElGamal is the `DIFFIE-HELLMAN` problem:

Problem 10 (`DIFFIE-HELLMAN`) *Given a prime p, a generator $g \in \mathbb{Z}_p^*$, and elements $g^a \bmod p$ and $g^b \bmod p$, compute $g^{ab} \bmod p$.*

We note that Problem 10 is known to be at least as hard as Problem 9.[???]

The definition of Problem 9 jogs Michael's memory enough that he remembers reading about the ElGamal public-key encryption protocol, which is based on the difficulty of Problem 9. He consults his bookshelf to recall the protocol...

3.1 Creation of the Public Key

The creation of the key in ElGamal is somewhat more complicated than in Rabin encryption. We will discuss Michael's key generation protocol, and note that Nikita will repeat the same process on her end. First, Michael chooses a large prime p and a generator g of the group $\mathbb{Z}_p^*$. He then selects a random integer a with $1 \leq a \leq p-2$ and computes $g^a \bmod p$. His public key will be the triple (p, g, g^a). His private key will be the exponent a. Formally, the key generation algorithm runs as follows [3]:

> `Procedure ELGAMAL-KEY:`
>   `Choose a large prime` p
>   `Choose a generator` g `of` $\mathbb{Z}_p^*$
>   `Choose a random element` a `of` $\mathbb{Z}_p^*$ `with` $1 \leq a \leq p-2$
>   `Compute` $g^a \bmod p$
>   `Public key is` (p, g, g^a)
>   `Private key is` a
> `End procedure`

Michael and Nikita choose their keys, $(p_M, g_M, g_M^{a_M})$ and $(p_N, g_N, g_N^{a_N})$, respectively, and continue with the encryption.

3.2 Encryption and Decryption

Let's suppose again that Nikita wants to send Michael her whereabouts as an encrypted message X. As with the Rabin protocol, she must first represent her message as an integer in $\mathbb{Z}_{p_M}^*$. As with Rabin PKE, we note that this is not a limiting factor of the protocol. She then chooses a random integer b, with $1 \leq b \leq p-2$ and computes g_M^b and $X \cdot (g_M^{a_M})^b$ using Michael's public key. The ciphertext is the pair $(g_M^b, X \cdot g_M^{a_M b})$, which she then sends to Michael. Formally, the encryption algorithm runs as follows [3]:

> `Procedure ENCRYPT-ELGAMAL(`X`):`
>   `Select a random integer` $b \in \mathbb{Z}_p^*$
>   `Compute` g^b `and` g^{ab} `using the public key`
>   `Return` $(g^b, X \cdot g^{ab})$
> `End procedure`

For the sake of notation, let Michael's public key be (p, g, g^b). Then Michael receives the pair $(g^b, X \cdot g^{ab})$ and wishes to recover Nikita's whereabouts, X. How can he do this? He first computes $(g^b)^{p-1-a} \bmod p$. Using some tricky arithmetic, note that

$$(g^b)^{p-1-a} \equiv (g^b)^{-a} \bmod p$$

Now Michael can recover X by computing

$$
\begin{aligned}
((g^b)^{-a}) \cdot (X \cdot g^{a_M b}) \bmod p &= g^{-ab} X g^{ab} \\
&= g^{-a_M b} g^{ab} X \\
&= g^{(-ab+ab)} X \\
&= 1X
\end{aligned}
$$

Thus Michael can recover the message X using just a few modular exponentiations and multiplications. Formally, the algorithm runs as follows:

Procedure DECRYPT-ELGAMAL((g^b, Xg^{ab})):
  Compute $g^{-ab} = (g^b)^{p-1-a} \bmod p$
  Compute $X = g^{-ab} X g^{ab}$
  Return X
End procedure

Well-versed in ElGamal PKE, Michael and Nikita generate keys and use the Diffie-Hellman key exchange to initialize their communications. Meanwhile, Madeline has been foiled, as the private key that she stole from Michael's apartment is no longer relevant. Angry that her reconnaissance work was in vain, she assembles her hit-men from Section One and orders them to infiltrate Michael's apartment and take him hostage, in hopes of torturing him into revealing Nikita's whereabouts. The hit-squad armed and ready, they set out on their mission. Michael is oblivious to his imminent danger...

3.3 Security of ElGamal

Before we continue, we will make a note on the security of ElGamal. We state the following without proof and direct the reader to [3]:

Theorem 11 *Breaking ElGamal is computationally equivalent to solving* DIFFIE-HELLMAN.

We also note that since breaking ElGamal is at least as hard as solving DIFFIE-HELLMAN, we have that breaking ElGamal is at least as hard as DISCRETE-LOG by transitivity.

3.4 Key Generation Considerations

We digress momentarily from our heroes to discuss some important considerations when generating the key for ElGamal encryption. We will first need a few facts about cyclic groups and generators. We state the following facts without proof, for proofs we direct the reader to any abstract algebra text, such as [2].

Fact 12 *Let G be a cyclic group. Then G has $\varphi(|G|-1)$ generators, where φ is the Euler-phi function.*

Fact 13 *Let G be a group and $a \in G$ an element of finite order k. Then H, the subgroup generated by a has order k.*

Fact 14 *If G is a cyclic group of order n, then for each positive divisor d of n, G has exactly one subgroup of order d.*

Fact 15 *If G is a cyclic group of order n and $d|n$, then G has $\varphi(d)$ elements of order d.*

How do these simple group-theoretic facts relate to ElGamal encryption? Let us examine the key generation algorithm, `ELGAMAL-KEY`. Note that the second step involves selecting a generator $g \in \mathbb{Z}_p^*$. The problem is that we need to pick one of the $\varphi(p-1)$ generators in $\mathbb{Z}_p^*$, which presents the difficulty of actually finding these elements. One strategy might be to select elements at random until we find a generator. But how do we test whether our element is a generator? One way is certainly to consider all the prime factors of $p-1$ and make sure that our random element doesn't generate a "small" subgroup of $\mathbb{Z}_p^*$ by using Fact 13 and Fact 14 above.[2] It is worth noting why we care that we do in fact have a generator of the entire group $\mathbb{Z}_p^*$. If we used a generator that generated some small subgroup of $\mathbb{Z}_p^*$, then an adversary could break our encryptions in practice. The reason is that `DISCRETE-LOG` is very hard for large groups, but for relatively small groups, an adversary can simply use trial and error with a powerful computer to find the discrete logarithm quickly. Thus the importance of finding a proper generator is clear. We could employ the following algorithm, given in [5]. Given a group G of order $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$:

```
Procedure FIND-GEN(n):
   Choose a random element a ∈ G
   For i = 1 to k do:
      if a^(n/p_i) eq 1 then choose another element
   End for
   Return a
End procedure
```

By now the reader should see a rather blatant problem with this algorithm, if we want to apply it to our key generation. We need the factorization of $p-1$, which, in general, would require us to efficiently solve the `FACTORIZATION` problem! If we can do that, then key generation is the least of our worries, since almost every public-key protocol would then be easily breakable.[3] Instead of immediately nixing the `FIND-GEN` algorithm, though, we can instead think of a way to choose our prime p intelligently so that the algorithm becomes more useful.

Definition 16 (Sophie-Germain prime) *A* Sophie-Germain prime *is a prime of the form $2q+1$, where q is also prime.*

Let's see what happens if we strictly use Sophie-Germain primes for our public keys in ElGamal. When we seek a generator of $\mathbb{Z}_p^*$, where p is a Sophie-Germain prime, our `FIND-GEN` algorithm above becomes amazingly efficient. Notice that we now know the factorization of $p-1$, since $p-1 = 2q$. Thus the only prime factors of p are 2 and q, so the `For` loop will only run for 2 iterations each time. But what if we keep getting unlucky; that is, we keep choosing a's that don't work? Well, if we're using a Sophie-Germain prime p, then our probability of choosing a generator on each trial rises significantly. By Fact 12, we know that there are $\varphi(p-1)$ generators of $\mathbb{Z}_p^*$. But

$$\varphi(p-1) = \varphi(2q) = \varphi(2)\varphi(q) = q-1$$

Note that we have used the fact that $\varphi(n)$ is a multiplicative function. So now our probability of choosing a generator on each trial is given by

[2] A "small" subgroup is any subgroup whose order is less than the order of $\mathbb{Z}_p^*$.

[3] This statement derives from the fact that all $\mathcal{NP}$-complete problems are within a polynomial factor of each other in difficulty, and most PKE schemes are based on the difficulty of such a problem.

$$\Pr(a \text{ is a generator}) = \frac{q-1}{\varphi(p-1)} = \frac{q-1}{2q} = \frac{1}{2}$$

This means that we can expect in general to require only two trials until we find a, and since each trial only requires 2 iterations of the `For` loop, we can see that `FIND-GEN` is very efficient.

3.5 ElGamal With a Random Prime: `NO-FACTORS`

Returning to Michael and Nikita, recall that the hit-squad from Section One was mobilizing toward Michael's apartment when we last left them. Now, Michael, forever vigilant after Madeline's intrusion, has installed a high-tech security system in his apartment. The system will notify him of any Section One employees within a six-block radius of his apartment. His security system suddenly starts ringing, and Michael quickly sits down at his laptop to notify Nikita of this development before he flees the apartment. Realizing the time pressure, Michael doesn't have time to generate a new key for their ElGamal based protocol; that is, he doesn't have enough time to find a Sophie-Germain prime to use. Knowing that it might take too long to run `FIND-GEN` on a random prime p without the factorization of $p-1$, Michael panics; Section One is just around the corner...
How can Michael avert disaster here? His problem is that he doesn't have time to go through and test primes for the Sophie-Germain property. His only option is to choose a random prime p and then choose a random element of $\mathbb{Z}_p^*$ and pray that it's a generator. We will now present a method which will allow Michael to use p as his prime and be confident that he can choose a generator of $\mathbb{Z}_p^*$ without the factorization of $p-1$. We will first present the method, a variation of the ElGamal protocol.

In key generation, instead of choosing a large prime p and just one generator, we choose p and a j-vector of pseudo-generators $\vec{g} = (g_1, \ldots, g_j)$. We then select a random $a \in \mathbb{Z}_p^*$ and compute $\alpha = (g_1 g_2 \cdots g_j)^a$. The public key will be $(p, \vec{g}, \alpha)$ and the private key will be a. Formally, key generation runs as follows:

```
Procedure NO-FACTORS-KEY(p):
   Select a j-vector g⃗ of random elements (g_1,...,g_j) in Z_p^*
   Select a random a ∈ Z_p^*
   Compute α = (g_1 g_2 ··· g_j)^a mod p
   Public key is (p, g⃗, α)
   Private key is a
End procedure
```

Now note that key generation in our modified protocol retains the structure of the key generation in ElGamal, except that we work with a vector of random elements instead of a single generator. We will soon see how this helps us ensure security. Now encryption and decryption follow the same structure as in ElGamal, so we present them here without comment (assume, as usual, that X is a message represented as an element in $\mathbb{Z}_p^*$):

```
Procedure NO-FACTORS-ENCRYPT(X):
   Choose a random element b ∈ Z_p^*
   Compute β = (g_1 g_2 ··· g_j)^b mod p
   Compute γ = X · ((g_1 g_2 ··· g_j)^a)^b mod p
   Return (β, γ)
End procedure
```

```
Procedure NO-FACTORS-DECRYPT(X):
   Compute (g1 g2 ··· gj)^{-ab} = β^{p-1-a}
   Compute X = β^{p-1-a} X γ mod p = (g1 g2 ··· gj)^{-ab} (g1 g2 ··· gj)^{ab} X mod p = X
   Return X
End procedure
```

It is clear from the two algorithms that this protocol "works"; that is, the `NO-FACTORS-DECRYPT` will always recover the original message X. We will now show that with high probability, this protocol is *at least* as secure as ElGamal with the same parameters.

Claim 17 *Given that at least one of the g_i generates $\mathbb{Z}_p^*$, the* `NO-FACTORS` *protocol is at least as secure as ElGamal with the same parameter p.*

Proof: Consider an arbitrary message X and its encryption $E(X)$ under the `NO-FACTORS` protocol. Suppose there is some entity A who can "break" `NO-FACTORS` (that is, he can recover X from $E(X)$). We will show that A can also (independently) recover X from $E'(X)$, where E' represents the encryption algorithm of ElGamal. For the sake of notation, we now refer to the product $(g_1 g_2 \cdots g_j)$ in the `NO-FACTORS` protocol as Π_g. Now for A to recover X from $E(X)$, he must have the ability to compute X given only $(\beta, \gamma) = ((\Pi_g)^b, X(\Pi_g)^a b)$. Recall that A knows the following:

- Each of the g_i (since they're in the public key)
- $(\Pi_g)^a$ (it's also in the public key)
- $(\Pi_g)^b$ (it's simply β in the encryption)

The fact that A can compute X from $E(X)$ implies that A can compute $(\Pi_g)^{ab} = (g_1 g_2 \cdots g_j)^{ab}$ (since that's precisely what it means to decrypt $E(M)$ in `NO-FACTORS`). But this means that A can compute g_i^{ab} for each $1 \le i \le j$. But by assumption, at least one of the g_i, say g_1, is a generator of $\mathbb{Z}_p^*$. Thus A can solve `DIFFIE-HELLMAN` problem, but by Theorem 11, this is computationally equivalent to breaking ElGamal, and thus A can recover X' from $E'(X')$. ■

Now we have shown that if one of the g_i is a generator of $\mathbb{Z}_p^*$, then `NO-FACTORS` is as secure as ElGamal. But how likely is it that one of the g_i is a generator? Obviously, this depends on the value of j, that is, the size of our j-vector. We will make this precise in a moment, but we will first need the following fact:

Fact 18 *If $\varphi(n)$ is the Euler-phi function of n, then for all $n \ge 5$ we have*

$$\varphi(n) \ge \frac{n}{6 \log \log n}$$

where $\log n$ denotes the natural logarithm of n.

Now, we know that the number of generators in $\mathbb{Z}_p^*$ is given by $\varphi(p-1)$ by Fact 12. Now given that we choose a random element from $\mathbb{Z}_p^*$, let the Bernoulli random variable Y be 1 if that element is a generator, and 0 otherwise. Then we have:

$$\begin{aligned} \Pr(Y = 1) &= \frac{\varphi(p-1)}{p-1} \\ &\ge \frac{\frac{p-1}{6 \log \log p - 1}}{p-1} \quad \text{By Fact 18} \\ &= \frac{1}{6 \log \log p - 1} \end{aligned}$$

Thus if we choose j elements uniformly at random with replacement and let Z denote the number of those elements that are generators of $\mathbb{Z}_p^*$, then the expected number of generators is given by

$$E[Z] \geq \frac{j}{6 \log \log p - 1}$$

So if we set $j = \log \log p - 1$, then we can expect (in the probabilistic sense) that at least one of the g_i will generate $\mathbb{Z}_p^*$. In practice, ElGamal uses a modulus p of approximately 2048 bits [??]. Thus if we set $j = 12$, then $E[Z] \geq 1$. Obviously, increasing the value of j will increase our probability.

Now, Michael, satisfied with this probability of error, quickly chooses a random prime p and a j-vector of pseudo-generators, and uses `NO-FACTORS` to encrypt his message to Nikita, thus notifying her that he will be leaving his apartment with his laptop and will contact her when he settles again. After packing up his laptop and some cash, Michael takes the fire escape and eludes his would-be captors in the nick of time. He jumps in an airport-bound taxi and takes the next flight to New York, hoping to escape detection by Section One for the time being...

4 Digital Signatures with ElGamal

4.1 What is a Digital Signature?

After settling in a run-down Queens apartment, Michael finally contacts Nikita. In order to establish a key for their communications protocol, they use the Diffie-Hellman key exchange, as they have always done. But the research division of Section One has devised a way to intercept all communications between Michael and Nikita, regardless of the fact that Section One has no knowledge of their whereabouts. Their communication channel is thus no longer secure. Section One can now mount a so-called "Man in the Middle Attack". When Michael sends his message announcing $g^n \bmod p$ (per the Diffie-Hellman protocol), Section One intercepts it and sends its own $g^t \bmod p$ to Nikita. Nikita then uses the secret key $g^{tn} \bmod p$ to send messages to Michael. Again, Section One intercepts these messages and decrypts them. Now Section One has the power to change any messages sent back and forth between Michael and Nikita. Thankfully, before they exchange any vital information, Michael uses his state-of-the-art listener-detection device to detect a listener on the line. Clearly something needs to be done about this problem if Michael and Nikita are to communicate securely. Nikita needs a way to verify that a given message actually comes from Michael, and not from some other entity, especially Section One.

We digress for a moment to see how this is accomplished outside the digital world. If Bob wants to send Alice a letter, he writes his message in the letter and signs the sheet of paper with his *official signature*. Alice is then able to examine the signature and verify that the letter in fact came from Bob. Michael and Nikita need a similar protocol. This discussion motivates the use of *digital signatures*:

Definition 19 (Digital Signature) *A* digital signature *is a string of data which associates a message with some originating entity (the sender).*

It turns out that we can associate arbitrary messages with their senders in a secure manner using some clever number theoretic tricks. In fact, most digital signature schemes inolve tweaking one or more encryption schemes (such as RSA or ElGamal) to suit the purpose. Here we present a digital signature protocol based on the ElGamal encryption presented above. The structure of a signature scheme consists of two algorithms: a *signature* algorithm (performed by the sender) and a *verification* algorithm (performed by the receiver). We now discuss how Michael and Nikita might subvert Section One's Man in the Middle Attack.

4.2 Key Generation

The key generation algorithm for ElGamal signatures runs in the same manner as the key generation of ElGamal encryption. Assuming Michael is the sender, he chooses a large prime p and a generator g of $\mathbb{Z}_p^*$. He then selects a random integer a, $1 \leq a \leq p-2$ and computes $g^a \bmod p$. The public key is then (p, g, g^a), and the private key is a. We do not write the algorithm out formally because it is identical to the `ELGAMAL-KEY` algorithm above [4].

4.3 The Signature

Now how does Michael "sign" his document?[4] Let's say that Michael has some document X that he wants to sign and send to Nikita. We must introduce the notion of a *hash function*. We will be only semi-rigorous about our definition, but it will suit our purposes here. We direct the unsatisfied reader to [5]. A *hash function* h is a map

$$h : \{0,1\}^* \to \mathbb{Z}_p$$

That is, it takes a number in binary as input and maps it onto an integer modulo p. This map has the property that it "looks random", that is, it's computationally intractable to find x and y such that $h(x) = h(y)$. Michael will use h to map his document onto an integer modulo p. The signature algorithm runs very similarly to the ElGamal encryption algorithm, and we present it formally as follows [4]:

```
Procedure SIGN-ELGAMAL(X):
   Choose a random integer b such that 1 ≤ b ≤ p − 2 and (b, p − 1) = 1
   Compute α = g^b mod p
   Compute b^{-1} mod (p − 1)
   Compute β = b^{-1}(h(X) − aα) mod (p − 1)
   Signature is (α, β)
End procedure
```

Upon receiving the signature pair (α, β), Nikita will run the verification algorithm to verify that Michael is the sender of X, and not some entity at Section One. The verification algorithm runs as follows:

```
Procedure VERIFY-ELGAMAL((α, β)):
   Look up public key (p, g, g^a)
   Compute u = (g^a)^α α^β mod p
   Compute h(X) and v = α^{h(X)} mod p
   If u eq v then accept
   Else reject
End procedure
```

[4] We now refer to a "message" as a "document", since the latter term meshes with the idea of a "signature".

How does the signature scheme work? That is, how does it use number theory to its advantage? Notice that if the signature is valid, then we have

$$\beta \equiv b^{-1}(h(X) - a\alpha) \bmod (p-1)$$

Multiplying both sides of the congruence by b, we have

$$b\beta \equiv h(X) - a\alpha \bmod (p-1)$$

And thus (rearranging terms),

$$h(X) \equiv a\alpha - b\beta \bmod (p-1)$$

And then we have

$$\begin{aligned} g^{h(X)} &\equiv g^{a\alpha+b\beta} \\ &\equiv (g^a)^\alpha \alpha^\beta \end{aligned}$$

Thus $u = v$ and the signature will be accepted by the algorithm.

Now the observant reader will notice that `VERIFY-ELGAMAL` requires the receiver to actually use X. But how can the receiver know what X is? Isn't that supposed to be the signed document? This confusion is resolved simply by considering X to be an encrypted version of the document. Then the receiver can first verify the signature with X, and if the verification algorithm accepts X, the receiver can then decrypt X.

4.4 Security of the ElGamal Signature

We now make a note on the security of this protocol. Security of digital signatures is discussed in slightly different terms. What would it mean to "break" the protocol? It would mean that someone from Section One is able to "forge" a signature of either Michael or Nikita; that is, to send a document to either Michael or Nikita with a signature such that `VERIFY-ELGAMAL` accepts the signature even though it didn't come from the intended sender. We will informally present the security of ElGamal.

In order for someone from Section One to forge Michael signature, he would first choose a random b per `SIGN-ELGAMAL`. But he then must compute $\alpha = b^{-1}(h(X) - a\alpha) \bmod (p-1)$. Assuming that `DISCRETE-LOG` is computationally intractable as discussed above, this computation of α is infeasible to anyone at Section One, since they would have no knowledge of the secret key a. Thus we assert that forging a signature under this scheme is at least as hard as solving Problem 9 [4].

Again consulting his bookshelf, Michael comes across his old cryptography notes and refreshes his memories of the ElGamal signature scheme. When his listener-detector reveals that the Section One adversaries are at lunch, he seizes the opportunity to use the temporarily secure line to tell Nikita about the protocol, and they exchange keys. By the time the Section One listeners return from lunch, Michael and Nikita are communicating with digital signatures, thus thwarting the Man in the Middle Attack. Any malicious messages sent to Nikita or Michael are summarily rejected by `VERIFY-ELGAMAL`. Michael and Nikita's communications are finally safe, for the time being. They agree that the situation as it stands is too dangerous for them to meet, so Michael continues his life in Queens. As an added precaution, he undergoes several more plastic surgeries to disguise his appearance. One day, however, while jogging through Central Park, Michael runs across William, an infamous member of the Section One hit-squad. Despite the plastic surgeries, William recognizes the elliptic curve shaped birthmark on Michael's arm. Although Michael is able to escape danger, he fears that his location is compromised. Even though William knows nothing of Michael's apartment, Michael decides that the time for another move is imminent...

5 Zero-Knowledge Proofs with Rabin Public-Key Encryption

5.1 What is a Zero-Knowledge Proof?

We haven't yet discussed the problem of how Michael and Nikita authenticate themselves to each other before beginning a communication session. It turns out that they have been using a simple password scheme. That is, Michael has some password X_M and Nikita has some password X_N. When Michael wants to authenticate himself to Nikita, he encrypts X_M and sends it to Nikita, who can decrypt and verify that the password is in fact X_M. She can then agree to communicate with Michael, since she knows it's not someone else on the other end of the line. We will see in a moment that this protocol can be disastrous.

After his encounter with William, Michael decides to be extra careful. He rarely leaves his apartment and keeps a constant watch of his surroundings. Unfortunately, Nikita is not so conscious. After a company-wide search effort, Madeline is able to locate Nikita. Deciding that, at this point, it would be better to bring the two cohorts down together than to capture Nikita individually, Madeline decides to take a more passive approach. One morning, while Nikita showers, Madeline infiltrates Nikita's bunker using the latest in morph-through-physical-barriers technology. She sits down at Nikita's computer and initiates a session with Michael, using Nikita's authentication software to impersonate her. When Michael sends his encrypted password to authenticate himself, Madeline uses Nikita's decryption software to decrypt and recover Michael's password. She then leaves Nikita's apartment seemingly untouched. With Michael's password in her memory, she is able to impersonate Michael in subsequent communications with Nikita. This situation is clearly unfavorable, to say the least. How can Michael and Nikita overcome this shortcoming of their authentication protocol?

This question motivates our discussion of Zero-Knowledge Proofs (hereafter referred to as "ZKP's"). We will again be semi-rigorous in our discussion, since a rigorous discussion of ZKP's would be beyond the scope of this paper. However, we will introduce the necessary definitions and present a ZKP protocol. The interested reader can refer to [5] for more information on ZKP's.

The basic concept of a ZKP is that Michael wants to authenticate himself to Nikita. He has some secret S that no one else knows. Using the ZKP protocol, he will prove to Nikita that he knows the secret, thus letting her be sure that it is in fact Michael on the other end of the line. But the key to the ZKP is that Michael will prove his knowledge of S without revealing *any information whatsoever* about S, in a very rigorous sense. We present a semi-rigorous definition of a ZKP here:

Definition 20 (Zero-Knowledge Proof) *A* Zero-Knowledge Proof (ZKP) *is a protocol involving a secret S, a prover P and a verifier V such that*

1. *If P knows S, P will always "pass" the protocol (called the* completeness *property).*[5]

2. *If P doesn't know S, P has a $\frac{1}{2}$ probability of passing the protocol (called the* soundness *property).*

3. *Any number of sequential repetitions of the protocol reveals nothing beyond the fact that P knows S (called the* zero-knowledge *property).*

This doesn't seem like a very good idea at first glance, especially since even a prover who doesn't know the secret can pass the protocol with probability $1/2$. We will see in a moment how we can decrease this probability exponentially. We first present the paradigm of the ZKP. As in Definition 20, there is a prover, P (Michael), who wishes to verify his identity to the verifier, V (Nikita). P can prove his identity by proving to V that he knows some secret S known only to him. Moreover, his proof will reveal *nothing* about S except that P knows S. We can see that, had Michael and Nikita used this type of protocol, Madeline would not have been able to recover Michael's password (secret), because his proof would reveal nothing about it! How can we implement such a protocol? We now discuss how to tweak Rabin PKE into a ZKP.

5.2 The ZKP Protocol

We will see that ZKP protocols are *interactive*, that is, they involve actions on the part of both P *and* V. We will refer to the execution of the protocol as a *conversation* between P and V, and each conversation comprises k *rounds* of communication. What does a round of communication involve?

At the beginning of the conversation, P will choose large primes p and q and compute $n = pq$ as in the Rabin PKE. P then chooses a random integer modulo n, x. Finally, P computes $y \equiv x^2 \bmod n$. The pair (n, y) is public, and x is P's secret.[6] After this initialization procedure, P and V may begin their rounds of communication.

Each round is conducted as follows. P will choose a random u in $\mathbb{Z}_n^*$. He sends $w \equiv u^2 \bmod n$ to V. V then chooses a random element c of $\{0, 1\}$ (perhaps by a fair coin flip). The element c is referred to as the "challenge". V sends c to P. Now if $c = 1$, then P lets $v = u$ and sends v back to P, who verifies that $v^2 \equiv w \bmod n$. If $c = 0$, then P computes $v \equiv ux \bmod n$, sends v to V, who verifies that $v^2 \equiv wy \bmod n$. We formalize the algorithm and then provide discussion [5]:

```
Procedure ZKP-ROUND((n, x)):
   P chooses a random u ∈ Z_n^* and computes w ≡ u^2 mod n
   P sends w to V
   V chooses a random c ∈ {0,1}
   V sends c to P
   If c eq 1 then:
      P sends v = u to V
      V verifies that v^2 ≡ w mod n
   Else if c eq 0 then:
      Using the secret x, P computes v ≡ ux mod n
      P sends v to V
      V verifies that v^2 ≡ wy mod n
```

[5] We say that P "passes" if V verifies his knowledge of S

[6] p and q are also kept secret, but x is the secret about which P will prove knowledge.

```
        End if
    End procedure
```

Now we will prove that this ZKP protocol fulfills the properties in Definition 20. These proofs will also serve to illuminate the protocol.

Claim 21 `ZKP-ROUND` *is* complete*: if P knows the secret x, then P will always pass the* `ZKP-ROUND` *algorithm.*

Proof: When V sends his challenge c, we know that either $c = 0$ or $c = 1$. We first consider the case in which $c = 1$. This is trivial, since P originally chose u, so he simply sends $v = u$, and since $v^2 \equiv u^2 \equiv w \bmod n$ by definition, V will always correctly verify that $v^2 \equiv w \bmod n$. Now in the case where $c = 0$, P knows x by assumption, so he can compute $v \equiv ux \bmod n$. But then $v^2 \equiv (ux)^2 \equiv u^2x^2 \equiv wy \bmod n$, so V will always correctly verify that $v^2 \equiv wy \bmod n$. Thus in either case P will pass. ■

We now see how the ZKP protocol works in a number theoretic sense [7].

Claim 22 `ZKP-ROUND` *is* sound*: If P does not know the secret x, then P will pass the* `ZKP-ROUND` *algorithm with probability $\frac{1}{2}$.*

Proof: First note that if $c = 1$, then P is sending $\sqrt{w} \bmod n$ to V, and if $c = 0$, then P is sending $\sqrt{wy}$ to V, since $ux \equiv \sqrt{wy} \bmod n$. Now assume that we have a malicious verifier P' who does not know x but is attempting to authenticate himself to V. He knows that if $c = 0$, he will need to send $\sqrt{wy} \bmod n$. He only knows u, w, y, and n. How can he do this? At the beginning of `ZKP-ROUND`, instead of choosing a random u and sending $w \equiv u^2 \bmod n$ to V, he will simply choose w so that $k \equiv wy \bmod n$, where k is some value for which P *does* know the square root modulo n. Now, if $c = 0$, P' is equipped to send $ux \bmod n$ to V. But what if $c = 0$? Then P' is expected to provide $u \equiv \sqrt{w} \bmod n$. But since he hasn't followed the protocol correctly, P' doesn't know u unless he can solve `SQROOT`. Thus P' will only pass if $c = 0$; that is, with probability $1/2$. Conversely, P' knows that if $c = 0$, he will have to provide $u \equiv \sqrt{w} \bmod n$. In this case, P' can simply follow the protocol, and he will have u to send. But now what if $c = 0$? Then P' is expected to provide $xu \equiv \sqrt{yw} \bmod n$. But P' doesn't know x by assumption. So in this case, P' only passes if $c = 1$; that is, with probability $1/2$. Thus, unless P' can solve `SQROOT` in practice, he can only pass a round with probability $1/2$. ■

Claim 22 doesn't seem like a terribly optimistic result. Why would we ever use this protocol if a dishonest prover has a probability of $1/2$ of fooling us? Well, we can run `ZKP-ROUND` as many times as we want, right? And each time we run it is independent of any previous time, so if we define the Bernoulli random variable Y_i to be 1 if a dishonest prover succeeds in round i and 0 otherwise, then if we run `ZKP-ROUND` k times the probability that a dishonest prover passes all k rounds is 2^{-k}. Thus if we have a conversation of 20 rounds, a dishonest prover will have probability $\frac{1}{2^20}$, a mind-numbingly small number. Thus we can make the probability of a dishonest prover passing the conversation negligible.

Claim 23 `ZKP-ROUND` *is* zero-knowledge*: any number of sequential rounds reveals nothing beyond the fact that P knows x.*

Proof: We sketch a proof here, as a rigorous proof requires concepts that are beyond the scope of this paper. Proving the zero-knowledge property amounts to proving that the protocol is *simulatable*. This means that there is a polynomial-time algorithm A which can produce, upon input of the secret to be proved, but without interacting with P, transcripts indistinguishable from those resulting with interaction with P. What does this mean? It means that there is some *black-box* with which any outside party can communicate and produce a conversation that looks *exactly* like a real conversation between P and V. ■

The zero-knowledge property implies that Michael, even when interacting with a malicious verifier such as Madeline, does not release any information about his secret that Madeline couldn't have figured out by herself from public knowledge. This is an extremely powerful result. In fact, if we return to the scenario above in Nikita's apartment, Madeline would not have been able to deduce any information whatsoever about Michael's password.

Nikita and Michael finally seem to have a solid grasp of their cryptographic protocols. For several weeks they communicate in complete secrecy. Any attempts by Section One to intercept and decrypt their messages are in vain. Confident in their security, Nikita and Michael finally reunite. Using the knowledge they have garnered along the way, they hack into the Section One mainframe and steal documents testifying to a twenty-year history of sinister, corrupt activities. After the F.B.I. anonymously receives these documents, they launch one of the most successful raids in history, bringing the entire Section One organization down and allowing Michael and Nikita to live happily and securely ever after.

6 Acknowledgements

I would like to thank Professor William Stein for assigning this project; it was a fun way to explore the intersection between two such interesting fields.

A An Instructive Implementation of Concepts

A.1 A High-Level Description

As an afterthought to our exploration of number theoretic themes in cryptography, we present a sample implementation of digital signatures and zero-knowledge proofs through a client-server model. We first note that our implementation is in no way secure; it is meant only as an illustrative tool of the concepts presented in this paper. The encryptions used in this implementation, while correct, do not support large enough integers to be of practical use, since today's computers can crack 32-bit encryption by brute force in a matter of seconds. Nonetheless, we believe it is of use to see what a sample implementation looks like. We first present a high level description.

The implementation consists of two programs: `message_server.c` and `message_client.c`. The motivation for our model lies in the following scenario. Suppose Alice is running some kind of messaging software (e.g. AOL Instant Messanger) on her machine. This software is represented by `message_server.c` (which we will henceforth refer to as the *server*). Now, it may be that Alice is a cautious woman, and decides that when she is away from her workstation she should only receive messages from people she knows. The people who try to contact her are represented by `message_client.c` (which we will henceforth refer to as the *client*). Now, this is a problem that most of us face (perhaps unknowingly) on a daily basis. A reasonable suggestion would be to have some sort of authentication scheme whereby a client identifies itself securely to the server before it begins messaging. That way, Alice's server will verify in advance that it is communicating with a "friendly" client. Now, we presented above reasons why a password scheme will not suit this purpose, since if someone breaks into Alice's office, he can steal any client's password and impersonate that client in the future. We thus use a zero-knowledge proof for this purpose. The motivation for further security runs as follows. Consider Bob, a friend of Alice, who uses his client software to authenticate himself and initiate a communication session with Alice's server. What if Bob, in turn, leaves his office without shutting down his connection? Then someone may break into his office and use the communication medium already established with Alice's server. This scenario motivates the use of digital signatures in our implementation. That is, each message sent from client to server is individually verified upon receipt using the `ELGAMAL` scheme presented above. We do not actually encrypt the message in our implementation; we assume that the actual communication medium is secure against listeners.[7]

A.2 Implementation Details

We now discusst he details of our implementation. The software lies in the two files mentioned above, along with a "public directory file" of public keys, `public_keys.directory`. The algorithm runs as follows. We first start the server (we can imagine that Alice is leaving her office and leaving up some sort of "answering service" for her message software). We then start the client, giving it the server port number, two flags[8], and the client's private keys p and q used for ZKP. The client then initiates ZKP with the server by sending its commitment. The server then checks the directory file for the client's public key y. Note that our implementation only supports one client per server, so the file contains only one number, but the software can easily be tweaked to support

[7] This is clearly not a valid assumption in everyday life, but we view it as a small allowance in light of all that is learned from the use of digital signatures and ZKP.

[8] The first flag turns on the `verbose` option, which means that the program will output all information as it performs the protocols. The second flag turns on the `sleep` option, which makes the program run more slowly, so the user can see what is going on.

multiple clients. After performing the ZKP authentication, the server invites the client to initiate messaging, at which point the client sends its digital signature public keys. It can then begin sending messages to the server. Messages are accepted from `stdin`, and each time a message is sent, the client prompts the user for his signature key. Before messaging began, the signature key was printed to the screen for 10 seconds (so that the client user may commit it to memory). The algorithms can be more formally summarized as follows:

```
Procedure MESSAGE_SERVER(port, verbose, sleep):
  For i = 1 to NUM_ROUNDS do
    Accept commitment from client
    Send challenge
    Accept response
    If verify_response(response) eq true then:
      Authenticate client
    Else:
      Reject client
    End if
  End for
  Accept signature keys from client
  While (true) do:
    If received_message then:
      If authenticate_signature then:
        print_message
      End if
    Else:
      Abort program
    End if
  End while
End procedure
```

```
Procedure MESSAGE_CLIENT(port, verbose, sleep, p, q):
  For i = 1 to NUM_ROUNDS do:
    Compute commitment and send to server
    Accept challenge from server
    Compute response and send to server
    If not authenticated then abort
  End for
  Send signature keys to server
  While (true) do:
    If user input detected
    Sign message
    Send message
  End while
End procedure
```

We can see from the pseudocode that our algorithms are very simple implementations of the ZKP and digital signature protocols presented above.

A.3 Sample Interaction

We now present a sample interaction between a server and client. Assume that the client's private keys are $p = 67$ and $q = 71$. Thus $n = 4757$. The client's public ZKP key is $y = 129$, which appears in the `public_key.directory` file. Thus his secret for ZKP is $x = \sqrt{129} \bmod 4757 = 120$. Note that if the client user wishes to change either p, q, or x, he obviously must change the corresponding y in the directory file. Finally, assume for the sake of page space that the `NUM_ROUNDS` for ZKP parameter has been set to 1 (i.e. we preform a 1 round conversation). As we can see, the client is verified using a ZKP, and he sends the message "Cryptography is fun!" after signing it. However, the user then leaves his room, and an adversary enters and begins using his computer. Fortunately, he does not know the real user's private digital signature key, so his message is rejected by the server. We first present a final note, and the transcript for the interaction appears on the following page.

A.4 A Final Note

For questions about this software or if you have any problems using it, please contact me at bgreenb@fas.harvard.edu. All readers have my full permission and encouragement to modify either source code file in any way they see fit.

```
greensmack% message_server 3000 1 1

Starting authentication...

Beginning round 1 of 1

Received your commitment:  3385

Sending my challenge (1)...

Received your response:  456

Authenticating...You have been authenticated.

Client has passed round 1 of 1

Client verified via zero-knowledge proof.

Received signature keys (p,g,gâ):  (67,7,55)

Received message!

Verifying sender...Sender verified.

Cryptography is fun!

Received message!

Verifying sender...Signature not validated!
```

```
greensmack% message_client 3000 1 1 67 71

Beginning authentication...

Beginning round 1 of 1

Please input your secret:  120

Private Key (p,q):  (67,71)
Secret (x):  120
Public Key (n,y):  (4757,129)

Sending my commitment (3385)...

Received your challenge:  1

Sending my response (456)...

Awaiting authentication...

Received authentication.

You have passed round 1

Sending my public keys (p,g,gâ):  (67,7,55)...

Private signature key:  52

Begin messaging

Cryptography is fun!

Please enter signature key:  52

Begin messaging

I am an adversary

Please enter signature key:  2
```

References

[1] Cormen, Leiserson, Rivest, and Stein. *Introduction to Algorithms.* The MIT Press. Cambridge, Massachusetts, 2001.

[2] Dummit and Foote. *Abstract Algebra.* John Wiley and Sons, Inc., New York, NY, 1999.

[3] T. ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE Transactions on Information THeory, 31 (1985), 469-472.

[4] T. ElGamal. *Cryptography and Logarithms over Finite Fields.* PhD Thesis, Standford University, 1984.

[5] Menezes, van Oorschot, and Vanstone. *Handbook of Applied Cryptography.* CRC Press LLC. Boca Raton, Florida, 1997.

[6] M.O. Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization.* MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[7] M.O. Rabin. *Lecture Notes on Public Key Cryptography.* September 26, 2002.