ElGamal-Type Signature Schemes in Modular Arithmetic and Galois Fields

Gerard J. Nealon

Department of Computer Science

Rochester Institute of Technology

Rochester, NY USA

gjn3855@cs.rit.edu

May 23, 2003

Chairman: Stanislaw Radziszowski

Advisor: Edith Hemaspaandra

Contents

Abstract

A digital signature is like a handwritten signature for a file, such that it ensures the identify of the person responsible for the file and prevents any unauthorized changes to the original file. Digital signatures use the same technology as most public key cryptosystems in which there is a public and private key. Most mathematical operations are done over a field $\mathbb{Z}_p$ where p is some large prime. It is possible to do the same operations over other finite fields.

1 Overview

There are many ways of implementing digital signatures, some more efficient than others, some easier than others. In my project, I am going to implement ElGamal-type digital signatures along with two different ways of implementing them; modular arithmetic and Galois fields. I will also discuss the use of hash functions within digital signatures and some of the attacks and weaknesses that were developed when hash functions and digital signatures were brought together. There are two main ways of representing Galois fields, polynomial basis representation and normal basis representation both of which will be implemented in this project. I will go into the theory behind finite fields and focus mainly on the efficiency differences between polynomial basis representation and normal basis representation of Galois fields along with irreducible polynomials that are required within. After understanding the ElGamal signature scheme and Galois fields independently, I will show how I will implement the ElGamal signature scheme over a Galois field.

1.1 Digital Signatures

When people are asked to authenticate themself, such as at a bank or a food store, they are often asked to sign their signature on a piece of paper. Once a person's signature is on paper, anybody can compare the signature to another signature and verify if the two signatures are the same or not. Unfortunatley this method is often used in our society regardless of its highly insecure nature. Handwritten signatures have many problems, forging someone else's signature is easily accomplised and frequently people do not sign their name exactly the same way each time. What if we wanted to attach our signature to a document stored on a computer? There are many cases when people would like to know who is responsible for a particular document, and they would also like to be assured that the requested document was not tampered with by a nonauthoritative entity. Digital signatures are a means of signing a document with all of the properties of a handwritten signature plus additional security. A digital signature is unforgeable. This means that there can be many different signatures for a given plaintext because of the random nature of the algorithm. It is a computational proof that the signer is responsible for the document. Digital signatures convince the receiver of the document that it has been signed by the claimed signer. A digital signature is not reusable. Once a document is signed, it can not be moved to another document. If the document is altered or the signature is moved to another document then the signature is no longer valid. [**?**]

1.2 Hashing and Attacks on Digital Signatures

Hash algorithms produce a specialized size data block from an arbitrary size file. Many computer scientists feel that 160 bits provide enough security for average scenarios. Hashes are often used because it is more efficient and easier to sign a 160 bit data block rather than a arbitrary length file. It is important to keep in mind that digital signatures sign the hash of a file and not the actual file. Because of this, you must be careful not to sacrifice the security of the system. When hash algorithms are used in digital signature schemes, the user generating the signature must hash the data first in order to produce the signature. On the other hand, the user who is verifying a signature must also hash the plaintext before computing the verification process. Hash algorithms tend to run fast so the double hashing required in a scheme is not a problem. It is convenient to have a hash algorithm as a standard, such has SHA-1, so that both users know what hash algorithm was used in the signature scheme.

Certain attacks have been produced that take advantage of some of the properties of hash algorithms. If one is not careful in choosing a hash algorithm, they can seriously degrade the security of the overall signature. We can assume that Oscar has a valid signed message, (x, y), and y is the signature of the hash of x, denoted as $h(x)$. Oscar can compute $h(x)$ and try to find $x' \neq x$ such that $h(x') = h(x)$. If Oscar succeeds at this attack, he then came up with another message, x' with a valid signature y.[**?**] This attack is very much brute force, and Oscar is limited to what x''s he can generate. It is also possible for a person to imitate Alice and pretend that he or she

has Alice's private key. If this is possible, Oscar, pretending to be Alice, can generate forged signtaures. It is often the practice of including some identify to a documents to prevent this from happening. Many people also add a timestamp from a third party trusted source, to signify the time a message was signed. It is important for a third party source to generate the time stamp because computer clocks can be easily changed.

1.3 The ElGamal Signature Scheme

The ElGamal cryptosystem can not be used as a signature scheme, although it can be modified to become one. The ElGamal cryptosystem is a non-deterministic, probabilistic signature scheme meaning there are many valid signatures for the same plaintext. In order for the scheme to work, the verification algorithm must be able to validate any of the valid signatures. According to [**?**] the ElGamal signature scheme is defined as follows:

Let p be a prime such that the discrete logarithm problem in $\mathbb{Z}_p$ is intractable, and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*$, $\mathcal{A} = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a\}$$

The values p, α, and β are the public key, and a is the private key. For $K = (p, \alpha, a, \beta)$, and for a secret random number $k \in \mathbb{Z}_{p-1}^*$, define

$$sig_K(x, k) = (\gamma, \delta)$$

where

$$\gamma = \alpha^k \pmod{p}$$

and

$$\delta = (x - a\gamma)k^{-1} \pmod{p-1}$$

For $x, \gamma \in \mathbb{Z}_p^*$ and $\delta \in \mathbb{Z}_{p-1}$, define

$$ver_K(x, (\gamma, \delta)) = true \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}$$

In the verification part of the scheme, it is easy to prove that this actually works. Take the formula, $\beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}$ and substitute γ and β from the defined formula's in the ElGamal scheme. We end up with the formula $\alpha^x \equiv \alpha^{a\gamma + k\delta} \pmod{p}$. We can see that the exponents have to be congruent, modulo $p-1$ in order for the entire equation to be congruent. We are left with the formula, $a\gamma + k\delta = x \pmod{p-1}$, from the exponents. Solving this formula for δ we get $\delta = (x - a\gamma)k^{-1} \pmod{p-1}$, which is the original formula defined in the ElGamal scheme.

1.4 Finite Fields

A finite field, denoted as $\mathbb{F}_q$, consists of a set of elements $\mathcal{F}$ and two binary operations on $\mathcal{F}$ called addition and multiplication that satisfy some arithmetic properties. In the scheme's thus far, I have been showing operations over some $\mathbb{Z}_q$. If q is prime, then $\mathbb{Z}_q$ is the same as $\mathbb{F}_q$. A finite field also exists when $q = p^m$, where p is prime. In this case, the field consists of q elements if and only if p is prime. For efficiency and security reasons, most cryptosystems and digital signature schemes operate over $\mathbb{F}_p$ where p is prime or $\mathbb{F}_{2^m}$. These fields, discovered by Galois, have order, number of elements,

equal to p and 2^m respectively. $\mathbb{F}_p$, $\mathbb{F}_{2^m}$, and $\mathbb{F}_{p^m}$ where p is prime are called Galois fields, or also known as $GF(p)$, $GF(2^m)$, and $GF(p^m)$, respectively. In the next couple of sections, I will talk about the two most commonly used finite fields.

1.4.1 Properties of Finite Fields

Finite fields have the following properties:

- Addition in this *prime field* is described as, given $a, b \in \mathbb{F}_p$, then $a+b = z$, where z is $a + b \pmod{p}$
- Multiplication in this *prime field* is described as, given $a, b \in \mathbb{F}_p$, then $a \times b = z$, where z is $a \times b \pmod{p}$
- Inversion in a *prime field* is described as, given $a \in \mathbb{F}_p : a > 0$, the inverse of a, denoted as a^{-1}, is c such that $a \times c \pmod{p} = 1$

1.4.2 $GF(p)$

Galois field p is very trivial to implement. In my project I will be using a large number package so implementing algorithms in $GF(p)$ will be very easy. Galois field p, or doing operations modular p, has field elements $\{0, 1, ..., p-1\}$. Any operations in this field will result in one of the field elements.

1.4.3 $GF(2^m)$

The most commonly used nonmodular finite field in cryptographic applications is the Galois field 2^m. There exist many algorithms for representing elements and computing operations in this field very efficiently. Galois field

2^m is called a binary finite field because it can be represented in its multiplication table as $\{0,1\}^m$ elements. Different algorithms make use of this binary format to manipulate numbers the fastest. For example, addition in this field is nothing more that XORing the array representation of field elements. Another way of representing $GF(2^m)$ is through a polynomial or normal basis. In the subsequent sections I will go into more detail of the representation of such binary fields.

1.4.4 Polynomial Basis Representations of Galois Fields

Given a irreducible polynomial in $\mathbb{Z}_p$ of the form:

$$f(x) = x^m + a_{m-1}x^{m-1} + ... + a_2x^2 + a_1x + a, \text{ where } a_i \in \mathbb{Z}_p$$

If we can choose a polynomial of this form such that it is irreducible, we can say that it defines a finite field. It defines a finite field because a irreducible polynomial is analogous to a prime number p which also defines a finite field $\mathbb{Z}_p$. All operations in this field will be reduced modulo $f(x)$, therefore we consider $f(x)$ to be the *reduction polynomial*. Finite fields, such as $\mathbb{F}_{p^m}$ with a polynomial basis are often denoted as, $\mathbb{Z}_p\ [x]/(f(x))$, where all coefficients are reduced modulo p and the polynomials are reduced modulo $f(x)$. In the case of a binary finite field 2^m, all coefficients are reduced modulo 2 therefore we can represent any field element as a binary string of length m.

1.4.5 Normal Basis Representations of Galois Fields

Another way of representing Galois fields is through the so-called normal basis. A normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is of the form $\{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{m-1}}\}$

where $\beta \in \mathbb{F}_{2^m}$. Given any element $a \in \mathbb{F}_{2^m}$, it can be represented in the form $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0, 1\}$.[?] One of the main advantages of representing elements of a finite field in this form is the efficiency of squaring. Without getting into the complicated mathematics, squaring ends up to be a simple rotation of the vector representation. Although multiplication can be very difficult in this representation, many people use an extension of normal basis called Gaussian normal basis which helps out a great deal in multiplication. The Gaussian normal basis records the complexity of the multiplication operation with respect the the basis. Gaussian normal basis contain a type, T, that is defined by the following formula: $p = Tm + 1$ if p is prime and the $gcd(Tm/k, m) = 1$.[?] The type is used in selecting a Gaussian normal basis for finite field 2^m.

1.5 Reduction Polynomials

Reduction polynomials need to be chosen carefully and also must be irreducible. A irreducible polynomial is a polynomial $f(x)$ such that it can't be factored into two polynomials $f_1(x)$ and $f_2(x)$ of smaller degree. A trinomial over $\mathbb{F}_{p^m}$ is a polynomial of the form $a_1 x^m + a_0 x^k + 1$, where $1 \leq k \leq m - 1$ and $a_i \leq p - 1$. A pentanomial polynomial over $\mathbb{F}_{p^m}$ is a polynomial of the form $a_3 x^m + a_2 x^{k_3} + a_1 x^{k_2} + a_0 x^{k_1} + 1$, where $1 \leq k_1 \leq k_2 \leq k_3 \leq m - 1$ and $a_i \leq p - 1$. According to [?] the following are the criteria for choosing a reduction polynomial over $\mathbb{F}_2$:

- If there exists an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial $f(x)$ must be an irreducible trinomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62

recommends that the trinomial used should be $x^m + x^k + 1$ for the smallest possible k such that $0 < k < m$ [**?**].

- If there does not exist an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial $f(x)$ must be an irreducible pentanomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62 recommends that the pentanomial used should be $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ chosen according to the following criteria such that $0 < k^1 < k^2 < k^3 < m$ [**?**]:

  1. k_3 is as small as possible
  2. for this particular value of k_3, k_2 is as small as possible
  3. for these particular values of k_3 and k_2, k_1 should be as small as possible

1.6 Testing for Irreducibility

A polynomial can't be used to define a finite field $\mathbb{F}_{p^k}$ unless it is irreducible over $\mathbb{Z}_p$. Therefore it is important to have a algorithm that can test to see if a polynomial is irreducible. Although I will not demonstrate how this works, it can be shown that there is at least 1 irreducible polynomial for a finite field $\mathbb{F}_{p^m}$. According to the *Applied Handbook of Cryptography* [**?**] a good algorithm for testing irreducibility of a monic polynomial is as follows:

INPUT: a prime p and a monic polynomial $f(x)$ of degree m in $\mathbb{Z}_p[x]$
OUTPUT: an answer to the question: "Is $f(x)$ irreducible over $\mathbb{Z}_p$?

1. Set $u(x) \leftarrow x$

2. For i from 1 to $\lfloor \frac{m}{2} \rfloor$ do the following:

   (a) Compute $u(x) \leftarrow u(x)^p \pmod{f(x)}$

   (b) Compute $d(x) = gcd(f(x), u(x) - x)$

   (c) If $d(x) \neq 1$ then return "reducible"

3. Return "irreducible"

This algorithm can be iterated many times while changing the value of $f(x)$ to find a irreducible polynomial.

1.7 ElGamal over Galois Fields

In the previous section 1.3, about the ElGamal signature scheme, all operations were done modulo some prime, p. It is possible to do the same operations in a finite Galois field and get the same or better security as if we did it modulo p. There are a few steps we must do in setting up the ElGamal signature scheme for computation in a Galois field.

- ~~Choose p, m~~
- ~~We need to choose a function $f(x)$ such that $f(x)$ is irreducible. We can use the algorithm presented in the previous section to test for irreducibility.~~
- ~~We must find a primitive element $\alpha \in \mathbb{Z}_p[x]/(f(x))$~~
- ~~Pick a private key, a~~
- Compute $\beta = \alpha^a$ using a modified version of the square and multiply algorithm

- ~~Choose a random value $k \in \mathbb{Z}_{p-1}$~~

The rest of the process is the same as described in the previous section. All operations are done modulo $f(x)$, all exponentiations are done using a modified version of the square and multiply algorithm, and all multiplicative inverses are performed using a modified version of the extended Euclidean algorithm.

2 Functional Specification

For my masters project, I will implement the ElGamal signature scheme in full power. I decided not to implement odd prime p Galois Fields, therefore I will restrict myself to binary Galois Fields. When I am finished with my project, it will be possible to compare efficiency between $GF(2^m)$ represented with polynomial basis, $GF(2^m)$ represented with normal basis, and $\mathbb{Z}_q$ where q is a prime such that $q \cong 2^m$. I will vary the size of m, to see if certain size fields compute faster then others.

3 Timing

There are many ways of implementing polynomial basis and normal basis representation of Galois fields. Some experts claim that polynomial basis representation algorithms runs faster then normal basis representation algorithms, some do not. In my project I will test the running time of these algorithms over equivalent fields and report my findings. I hope to reveal which basis runs the best and what algorithm is the most efficient for each

basis. Also in my project, I will show the complexity of all the algorithms that I present and I plan on testing my timing results with the estimated complexity to verify its accuracy.

4 Field Sizes

The size of a field is very important in cryptographic algorithms. Depending on the level of security, the recommended field size of discrete logarithm based cryptosystems is 1024 bits. In my project, the modulus I plan on testing, for finite field $\mathbb{F}_p$, will be a prime number between 512 bits and 2048 bits. My reasoning for this wide range is for testing purposes only. Likewise, I will test the same algorithms implemented over a binary field, $\mathbb{F}_{2^m}$ where m will range from 512 to 2048.

5 Deliverables

The following list is what I plan on implementing in my Master's Project:

- I will not implement my own large number package for manipulation of arbitrary size numbers, but I will use a standard package which can handle large numbers. I would like to focus my time and effort in the constuction of Galois fields.
- I will implement the ElGamal Signature Scheme in modular arithmetic
- I will not implement SHA-1 which is the hash function used for signature schemes, but I will use an already made package that implements SHA-1.

- I will implement a package for constructing binary Galois fields. This package will use polynomial basis and normal basis representations of the binary Galois field.
- I will implement an algorithm for testing of irreducibility polynomials over $\mathbb{Z}_p$.
- I will implement ElGamal over Galois fields $\mathbb{F}_{2^m}$.

6 Time Table

Acquiring a Committee	By the end of week 5
Project Proposal	By the end of week 9
Implementation of ElGamal Signature Scheme using Modular Arithmetic	Mid June
Implementation of algorithms for irreducibility	End of June
Implementation of binary Galois Fields using polynomial basis representation	Mid July
Implementation of binary Galois Fields using normal basis representation	Beginning of August
Testing and analysis of Algorithms over different fields	Mid August
Completion of writeup	End of September
Defense	End of September

References

[1] Alfred J. Menezes, Paul C. van Oorschot, S. A. V. *Handbook of Applied Cryptography*. CRC Press, 1996.

[2] Don Johnson, Alfred Menezes, S. V. The elliptic curve digital signature algorithm (ecdsa). *IJIS* (2001).

[3] Gao, S. Digital signatures. http://www.math.clemson.edu /faculty/Gao/cryptomod /node5.html, October 1999.

[4] Publication, F. I. P. S. Digital signature standard, May 1994.

[5] R. Lidl, H. N. *Finite Fields*. Cambridge University Press, 1996.

[6] Stinson, D. R. *Cryprography Theory and Practice*. Chapman and Hall/CRC, 2002.

[7] Tozer, C. *Digital Signature Guidelines*. American Bar Association, 1996.

[8] X9.62, A. Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ecdsa). 1999.