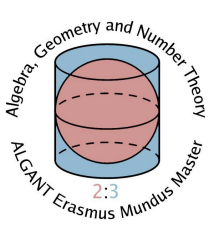

ERASMUS MUNDUS MASTER ALGANT
UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE MM. FF. NN.
CORSO DI LAUREA IN MATEMATICA

ELABORATO FINALE

STUDY OF DIGITAL SIGNATURE SCHEMES FOR GNSS SIGNAL AUTHENTICATION AND INTEGRITY PROTECTION

RELATORE: PROF. N. LAURENTI

DIPARTIMENTO DI MATEMATICA PURA E APPLICATA

LAUREANDO: VARUNI MEHROTRA

ANNO ACCADEMICO 2014/2015

Contents

Introduction

Galileo is the global navigation satellite system (GNSS) that is currently being created jointly by European Commission and the European Space Agency (ESA). One of the aims of Galileo is to provide high-precision positioning system upon which European nations can rely which is independent from the Russian GLONASS and US GPS systems, which are run by military authorities. Galileo system will consist of 30-satellites (27 operational and 3 active spares) by its full completion which is expected by 2020 [1]. As of March, 2015 the system consists of 8 operational satellites [2]. The main difference between Galileo and other existing GNSS is that the services provided by the former will always remain under civilian control [3]. The Galileo system will provide four main services [1],[2] and [3]:

1. **Open Service (OS)**: The OS provides positioning and timing information worldwide through ranging signals and data broadcast by the Galileo satellite constellation. This service is available without charge to all users with a Galileo-compatible receiver.

2. **Commercial Service (CS)**: With an encrypted higher-bandwidth improved-precision the Commercial Service is available at a cost. It aims to provide 'added value' data with respect to the Open Service. These 'added value' services are related to high accuracy which is understood as a position accuracy and authentication. One of the main features the CS will bring with respect to other GNSS is the capability to broadcast globally external data in real time. Note that the CS is based on adding two signals to the OS signals.

3. **Public Regulated Service (PRS)**: The PRS provides positioning and timing information worldwide through ranging signals and PRS data broadcast by the Galileo satellite constellation. EU Member State government agencies including emergency services and police will be the main users of this service. This navigation service will be available at all times even if other services are disabled in time of crisis thus ensuring continuity of service to authorised users. It is an encrypted navigation service. The PRS will only be accessible through receivers with a PRS security module loaded with a valid PRS decryption key.

4. **Search-And-Rescue (SAR)**: This service as the name suggests is intended to support search-and-rescue operations wherein Galileo satellites will be able to pick up signals from emergency beacons carried on ships, planes or persons and send these signals back to national rescue centres. Following which, a rescue centre will know the precise location of the accident.

We are interested in the Galileo Commercial Service (CS) and the added value services of the CS namely authentication i.e., the ability of the system to guarantee the users that they are utilizing signals coming from the Galileo satellites and not from any other source. Commercial GNSS are vulnerable to various attacks. We are concerned with the authentication of the navigation information, which would contribute to improving the worldwide GNSS security and make Galileo more attractive and resistant to the below attacks [11], [17].

1. **Jamming**: Due to the low received power of GNSS signals a jamming attack is the most trivial and common on positioning systems. It is a denial-of-service attack against positioning systems. In such an attack, the adversary tries to interfere with the signals/messages that are being transmitted by the navigation system so that the receiver can't determine the correct position thus targeting the unavailability of the system. This is a great threat to GNSS such as Galileo.

2. **Relaying**: This attack deals with relaying the navigation signals received at the wanted spoofing position say $\bar{p}$ to the receiver at the actual position p, which will in turn make the receiver believe to be at the false position $\bar{p}$, even though the true position of the receiver is p.

3. **Signal-synthesis/ Spoofing**: In this attack, an adversary generates and transmits false navigation signals so that the GNSS receiver computes an incorrect position or timing. In other words, the objective of a spoofing attacking is to fool the receiver into thinking it is in a different (false) position. If structure of the message is known to the public, the adversary can easily create valid navigation messages. This attack attempts to misrepresent the users true location while at the same time avoiding detection of the attack itself. For example,a spoofer could be used to broadcast a false location to a GNSS receiver in a truck carrying a load of dangerous weapons. The spoofer would make the receiver report that it was still on track, when in fact the truck could have been stopped and stolen.

4. **Replay or Meaconing**: In this type of attack, spoofed signals can be generated based on previously received GNSS signals: the adversary records navigation messages and retransmits them. Thus, the attack is based on receiving the GNSS signal and replaying it with some delay.

5. **Selective-delay**: In such an attack the adversary takes advantage of the fact that the arrival time of the navigation signals are used to determine the position so the adversay can delay each navigation signal in such a way that the receiver calculates a false position in turn spoofing the receiver for the entire coverage area of the signals. In other words, the attacker delays each satelltie signal for a different amount of time so that a false position is calculated The signals can only be delayed but can't be sent prior to their reception. Note that the adversary doesn't need to know the data content of the signal for the success of this attack.

6. **Counterfeit correction message**: Positioning systems can be effected by different error sources. To minimize the effects of these error sources, reference stations are used to collect data and to generate correction messages that will help the receiver to calculate a more accurate position. In this attack the adversary forges the correction messages.

A receiver using these forged correction messages will compute a false position. Besides directly forging the correction message, the attacker can also try to attack the reference stations. If the attacker is able to successfully spoof a reference station, this station will generate false correction messages, causing the receivers to compute false positions. Note that this attack is not an attack on the GNSS signal but on the signal which generates the correction messages and is out of the scope of our study.

7. **Tamper the receiver**: If the adversary has access to the receiver he/she can try to tamper the receiver.

Authentication is an essential problem in the field of communication. GNSS authentication is different from information authentication, as it's objective is not only to authenticate the information encoded in the signal but also to authenticate the signal time of arrival, at least against certain threats and with a certain level of confidence. The following authentication issue arises: confirmation that a navigation signal actually originates from the indicated satellite. Navigation system security is ever more important for two reasons. The first reason is to ensure that the position, navigation, and time (PNT) information upon which we increasingly rely are indeed trustworthy. The second is that secure PNT can serve as a building block for protection of critical data and assets in the global fight against information technology attack. We refer to these features as security for navigation and security from navigation. GNSS security threats exist now and will increase in the future [9].

Authentication and integrity protection of navigation messages have emerged as defense mechanisms to the attacks described above. Authentication would ensure that navigation messages generated only by GNSS entities are accepted at and used by receivers. Authenticable signals thus allow a user to recognize forged signals. Integrity would ensure that modification or utilization of part of navigation messages towards spoofing or meaconing is not possible either. We focus on Message Authentication, the goal of which is to prevent an attacker from generating his/her own navigation messages thus making signal-synthesis and counterfeit-correction message attacks nearly impossible. Message authentication can be achieved through message authentication codes (MAC), encryption with integrity check or digital signatures. In our work we will focus on Navigation Message Authentication (NMA) which refers to the cryptographic authentication of the messages wherein the low-rate navigation message is digitally signed, allowing a receiver to verify that the Galileo Control Segment generated the data. Jamming, Relaying, Selective-delay, Non-selective delay, Tamper the receiver are all attacks at the physical layer and cannot be protected by digital signature mechanisms.

The implementation of GNSS will result in an improvement of the performance parameters for accuracy, integrity, availability and continuity. To maximize the use of GNSS authentication, cryptographic key management should be simplified as much as possible. The different approaches for cryptographic protection include [16]:

1. **Symmetric key cryptography** is not viable for our purpose, with one secret key shared by the GNSS and each receiver. Navigation messages would need to be authenticated for each of the receivers individually. In this case a malicious receiver device knowing the symmetric key could forge signals himself and fool other receivers. Another disadvantage

of symmetric key cryptography in GNSS is the requirement of tamper-resistant receivers to protect the secret keys from unauthorized discovery. The cost of manufacturing a tamper-resistant receiver is high. Symmetric key cryptography is more applicable in the Public Regulated Service (PRS) where there is trust among users.

2. **Asymmetric or public key cryptography** on the other hand appears as a viable choice in the given scenario. The GNSS obtains pairs of private and public keys, one pair per satellite with each public key bound to a satellite via a certificate. Each receiver obtains all certified public keys for all GNSS satellites. Each satellite digitally signs with the private key its navigation message. Public key cryptography is more viable in the OS/CS.

Thus, asymmetric schemes, whereby the user receivers need only possess a public key, are preferred to symmetric schemes, whereby the user needs to store a secret key in a security module within the receiver. Asymmetric schemes can be achieved mainly through Digital Signatures, such as RSA, DSA or its Elliptic Curve variants whereby the satellites transmit a digital signature of their navigation data. The main advantage of authentication through digital signatures is that there are known methods and functions in the cryptographic standards that make them realiable whereas the main disadvantages of authentication through digital signatures are the bandwidth required to transmit the authentication information, computational effort required per authentication and the fact that some elliptic curves may be subject to patent rights.
Our goal is to evaluate the various digital signature protocols. In other words we discuss the cryptographic signal authentication through the NMA technique whereby a public key digital signature is embedded in the navigation message.

Acknowledgment: The past two years have been a mix of happiness, stress, adventure and learning. I would first like to thank my advisor Professor Nicola Laurenti for his support, guidance and patience in enabling me to explore the area of Cryptography and it's applications in Satellite Navigation Systems. He has been very kind and patient. I would also like to thank all the professors, both at the Mathematical Institute Leiden University, The Netherlands and Padova University, Italy for their undue support. My time in Leiden was filled with memories I will treasure forever. It was great fun solving exercises in the infamous ALGANT Room, going to Dutch Lessons, having random picnics by the canal. A special mention to Kathelijne Smits who has been so very kind to always find a solution to the numerous problems I came to her with. The ALGANT programme gave me the unique opportunity to meet an amazing group of people: Pia - my flatmmate in Italy, Andrei - for helping me with my exams and providing me notes as well as printouts, you have been such a huge help this past year. These two years wouldn't have been possible without the support of my family, who motivated me and helped me overcome major obstacles both academically and in life, when I was at my lowest point and wanted to give up. I love you for always making me smile and believing in me even when I didn't believe in myself. I would also like to thank my friends namely Sunali, Savitha, Shwetha, Bianca, Kaylah (thank you for being here in the last days before my thesis), Irina, Aashik, Abhipsha and Xinran for being such a great support system even when we are not together and spread throughtout the world on different continents, I know that I can always count on you! Lastly, I would like to thank technology in the form of FaceTime and Skype without which it wouldn't have been possible to keep in touch with friends and family.

Chapter 1

Preliminaries

In this chapter we begin by introducing the basic cryptographic concepts and terminology used to develop and implement authentication methods in the field of navigation systems.The general definition of cryptography is the design and analysis of mathematical techniques that enable secure communications in the existence of adversaries. Information security is of the greatest importance in a world in which communication over open networks and storage of data in digital form play a key role in daily life. The science of cryptography provides efficient tools to secure information.

1.1 Cryptographic Terms and Concepts

Definition 1.1.1 *Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.*

Cryptographic Goals: In the context of GNSS we are interested in the following information security objectives which we will discuss in more detail in Chapter 2.

1. *Data Integrity* is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

2. *Authentication* is a service related to identification.This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

3. *Non-repudiation* is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken,

a means to resolve the situation is necessary. A procedure involving a trusted third party is needed to resolve the dispute.

A number of basic cryptographic primitives are used to provide information security. Examples of primitives include encryption schemes, hash functions and digital signature schemes. These primitives should be evaluated with respect to various criteria such as:

1. *Level of Security* - This is usually difficult to quantify. Often it is given in terms of the number of operations required to defeat the intended objective. Typically the level of security is defined by an upper bound on the amount of work necessary to defeat the objective.

2. *Functionality* - Primitives will need to be combined to meet various information security objectives. Which primitives are most effective for a given objective will be determined by the basic properties of the primitives.

3. *Methods of Operation* - Primitives when applied in various ways and with various inputs, will typically exhibit different characteristics. So one primitive could provide very different functionality depending on its mode of operation or usage.

4. *Performance* - This refers to the efficiency of a primitive in a particular mode of operation.

5. *Ease of Implementation* - This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment.

The relative importance of various criteria is very much dependent on the application and resources available. For example, in an environment where computing power is limited one may have to trade off a very high level of security for better performance of the system as a whole.

We now discuss some basic definitions:

Definition 1.1.2 ***(One-way function)****:A function f from a set X to a set Y is called a on-way function if $f(x)$ is 'easy' to compute for all $x \in X$ but for 'essentially all' elements $y \in Im(f)$ it is 'computationally infeasible' to find any $x \in X$ such that $f(x) = y$.*

Definition 1.1.3 ***(Trap-door function)****: A trapdoor one-way function is a one-way function $f : X \to Y$ with the additional property that given some extra information (referred to as trapdoor information) it becomes feasible to find for any given $y \in Im(f)$, an $x \in X$ such that $f(x) = y$.*

1.2 Public Key Cryptography

In modern day communication systems, symmetric cryptosystems turned out to have two essential disadvantages namely:

- The problem of key management and distribution: A communication system with n users, who all use a symmetric cryptosystem to communicate with each other, implies the need of $\binom{n}{2}$ keys and $\binom{n}{2}$ secure channels. Whenever, a user wants to change his keys or a new user wants to participate in the system $n-1$ (resp. n) new keys have to generated and distributed over as many secure channels.

- The non-repudiation problem: In communication systems the electronic equivalent of a signature is needed. Symmetric cryptosystems don't provide this feature in a natural way, especially when there is a conflict between the sender and receiver, it's impossible to decide who is right. Any message by one of them could also have been made by the other.

These disadvantages gave rise to public-key cryptosystems.
Public Key Cryptography (PKC) is an asymmetric scheme, each participant possesses a pair of keys - a public key and a private key. In public key encryption, private key is used for decryption and public key is used for encryption and in public key authentication (digital signatures), private key is used for signing and public key for verification. In PKC the authenticity of public keys is guarenteed via certificates. Efficient Implementation of cryptographic algorithms has been the focus point of major research. Different metrics such as execution time (speed), implementation space and power usage/energy consumption are used to quantitatively measure the performance of a design/implementation. Efficiency can be defined in terms of these metrics depending on the application requirements. While execution time is important for applications where latency is of utmost importance, implementation space is crucial for constrained platforms. Since there is almost always a trade-off between execution time and implementation space, faster designs generally require more area. Examples of Public Key Cryptosystems include Elgamal (in which the security is based on the discrete logarithm problem) and RSA (in which the security is based on the difficulty of factoring).

What you need for a public key cryptographic system to work is a set of algorithms that is easy to process in one direction, but difficult to undo. Algorithms that have this characteristic easy in one direction, hard the other are known as Trapdoor Functions. Finding a good Trapdoor Function is critical to making a public key cryptographic system computationally secure.
Note that a public-key cryptosystem can never provide unconditional security ([28], Chapter 5, pp. 162)

Public Key Cryptography versus Private Key Cryptography:
Problems arise when using symmetric algorithms for communications between two parties say A and B. The main complication is key exchange; in order for B to read messages sent by A, they must first agree on a session key to encrypt all communication between them for a certain period of time. This exchange of keys is very vulnerable to an eavesdropper, because the key must pass over potentially lengthy channels in either unencrypted form, or encrypted using a publicly known key. The latter is essentially equivalent to no encryption at all for all but the most casual of eavesdroppers. Public-key cryptography, to some extent circumvents the key exchange problem, though there are still issues. For example, the Man-in-the-Middle Attack.

In a similar vein, there are also problems with the sheer number of keys needed for symmetric algorithms when communicating in a large network. Assuming each pair of users in the network want to be able to communicate privately, a separate key is needed for each pair. Thus, a network of n users will require $n(n1)/2$ keys. It is easy to see that as the network grows, the number of keys grows at a fairly quick rate. The same network using public-key cryptography would need only $2n$ keys (one public and one private key for each user) for all pairs of users to communicate in private. Probably one of the strongest arguments against public-key algorithms is their bulkiness, both in storage space and computational power required. Overall, it is difficult to say which is better, because they are designed to solve different sort of problems. Symmetric algorithms are best at encrypting data, and many a times faster than their public-key counterparts. Public-key algorithms, though slow, are best for key management and digital signatures. As amazing as public-key cryptography seems, it would not be a wise choice to forgo symmetric cryptography altogether. For our purposes, i.e. digital signatures, public-key cryptography is undoubtedly the right choice.

Let's discuss advantages and disadvantages of each in more detail:

1. Advantages of symmetric-key cryptography
   (a) Keys for symmetric-key ciphers are relatively short.
   (b) Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions and computationally efficient digital signature sch-emes.
   (c) Symmetric-key encryption is perceived to have an extensive history
2. Disadvantages of symmetric-key cryptography
   (a) In a two-party communication, the key must remain secret at both ends.
   (b) In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted third party.
   (c) In a two-party communication between entities A and B, sound cryptographic practice dictates that the key be changed frequently, and perhaps for each communication session.
   (d) Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a trusted third party.
3. Advantages of public-key cryptography
   (a) Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
   (b) Depending on the mode of usage, a private key/ public key pair may remain unchanged for considerable periods of time, e.g., many sessions

(c) Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.

(d) In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

Symmetric-key and public-key cryptography have a number of complementary advantages. Current cryptographic systems exploit the strengths of each. Public-key cryptography may be used to establish a key for a symmetric-key system being used by communicating entities A and B. In this scenario A and B can take advantage of the long term nature of the public/ private keys of the public-key scheme and the performance efficiencies of the symmetric-key scheme.

4. Disadvantages of public-key encryption

(a) Key sizes are typically much larger than those required for symmetric-keyencryption and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

(b) No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key schemes found to date have their security based on the intractability of a small set of number-theoretic problems.

(c) Public-key cryptography does not have as extensive a history as symmetric-key encryption, being discovered only in the mid 1970s.

Thus, in the context of GNSS we prefer public-key protocols over private-key since the latter although are generally more computationally efficient and offer shorter signatures, but they also require shared and secure private keys which makes them inpractical for Navigation Message Authentication (NMA) because storing a private key requires tamper-proof receiver. Furthermore, key management for symmetric protocols would be complicated: if any one of the private keys were disclosed, then every receiver would need to securely update the private key. On the other hand, public key protocols are practical because the public key can be stored unsecured in receiver memory. Despite the fact that the cryptographic key may be widely known, public key protocols offer as much security as private key methods and are believed to be secure against the strongest cryptographic attacks.

1.2.1 Significane of Key Length

Key length is directly proportional to security. In modern cryptosystems, key length is measured in bits and each bit of a key increases the difficulty of a brute-force attack exponentially. It is important to note that in addition to adding more security, each bit slows down the cryptosystem as well. Because of this, key length is a tradeoff between practicality and security. Furthermore, different types of cryptosystems require vastly different key lengths to maintain security. For instance, modulo-based public key systems such as Diffie-Hellman and RSA require rather long keys (generally around 1,024 bits), whereas symmetric systems, both block and stream, are able to use shorter keys (generally around 256 bits). Furthermore, elliptic

curve public key systems are capable of maintaining security at key lengths similar to those of symmetric systems; see table below. While most block ciphers will only use one key length, most public key systems can use any number of key lengths. Key length is equal to the number of bits in an encryption algorithms key. A short key length means poor security. However, a long key length does not necessarily mean good security. The key length determines the maximum number of combinations required to break an encryption algorithm. If a key is n bits long, then there are two to the n^{th} power (2^n) possible keys. For example, if the key is one bit long, and that one bit can either be a zero or a one, there are only two possible keys, 0 or 1. However, if the key length is 40 bits long, then there are 2^{40} possible keys. This term is also known as key size.

1.3 Elliptic Curve Cryptography

1.3.1 Introduction

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. In elliptic curve cryptography, the group used is the group of rational points on a given elliptic curve. The main benefit provided by ECC is a smaller key size, reducing storage and transmission requirements. In the context of GNSS we are worried about ensuring the highest level of security while maintaining performance, thus ECC makes sense to adopt. The security of a public key system using elliptic curves is based on the difficulty of computing discrete logarithms in the group of points on an elliptic curve defined over a finite field. Using elliptic curves as the underlying group for public key cryptography was first suggested in 1985 by N. Koblitz and V. Miller because the discrete logarithm problem was believed to be harder for elliptic curves than for finite fields. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves.

Comparision of Elliptic Curve Cryptography
One way judgments are made about the correct key size for a public key system is to look at the strength of the conventional (symmetric) encryption algorithms that the public key algorithm will be used to key or authenticate. Examples of these conventional algorithms are the Data Encryption Standard (DES) created in 1975 and the Advanced Encryption Standard (AES) now a new standard. The length of a key, in bits, for a conventional encryption algorithm is a common measure of security. To attack an algorithm with a k-bit key it will generally require roughly $2k - 1$ operations. Hence, to secure a public key system one would generally want to use parameters that require at least $2k - 1$ operations to attack. Table 1.1 gives the key sizes recommended by the National Institute of Standards and Technology to protect keys used in conventional encryption algorithms like the (DES) and (AES) together with the key sizes for RSA, Diffie-Hellman and Elliptic Curves that are needed to provide equivalent security.

For the most commonly used 1024-bit keys for an integer based algorithm; the elliptic curve

Symmetric (in bits)	RSA and DLOG (in bits)	Elliptic Curve (in bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1.1: NIST Recommended Key Sizes
[18]

counterpart requires only 160-bit keys for the equivalent security 2.1. This is 7 times reduction in the space required to store these keys, or a similar reduction in bandwidth required to transmit these keys. This reduction in the size of data allows much faster completion of the algorithms. One can see that as symmetric key sizes increase the required key sizes for RSA and Diffie-Hellman increase at a much faster rate than the required key sizes for elliptic curve cryptosystems. Hence, elliptic curve systems offer more security per bit increase in key size than either RSA or Diffie-Hellman public key systems.

Security is not the only attractive feature of elliptic curve cryptography. Elliptic curve cryptosystems are also more computationally efficient than the first generation public key systems, RSA and Diffie-Hellman. Although elliptic curve arithmetic is slightly more complex per bit than either RSA or DH arithmetic, the added strength per bit more than makes up for any extra compute time.
Closely related to the key size of different public key systems is the channel overhead required to perform key exchanges and digital signatures on a communications link. The key sizes for public key in Table 2.1 is also roughly the number of bits that need to be transmitted each way over a communications channel for a key exchange. In channel-constrained environments, elliptic curves offer a much better solution than first generation public key systems.

Elliptic Curve Paramters: Implementation of elliptic curve cryptography involves the selection of a suitable elliptic curve (determined by the coefficients in the elliptic curve equation), the representation of field elements (e.g. a binary field or a prime field), algorithms for field arithmetic and elliptic-curve arithmetic.

1.4 Elliptic Curves over Finite Fields

We introduce elliptic curves over a finite field and describe how to put a group structure on the set of points on an elliptic curve. Elliptic curves are believed to provide good security with smaller key sizes, something that will prove to be useful in Navigation Message Authentication (NMA) for Global Navigation Satellite Systems (GNSS). The cost, speed and feasibility of Elliptic Curve Cryptosystems depend on the finite field $\mathbb{F}_q$ where $q = p^m$ on which it is implemented. There are usually two finite fields to work on: prime finite field $\mathbb{F}_q$ (i.e. m = 1) when p is a prime number > 3 and binary finite field $\mathbb{F}_{2^m}$.

1.4.1 Introduction

Definition 1.4.1.1 *Let K be a field of characteristic $C \neq 2,3$ i.e. a prime field and let $x^3 + ax + b$ where $a, b \in K$ be a cubic polynomial with no multiple roots. An elliptic curve over K is the set of points (x, y) with $x, y \in K$ which satisfy the following equation*

$$y^2 = x^3 + ax + b \tag{1.1}$$

together with an element O called the "point at infinity" where a and b must be such that the discriminant $-16(4a^3 + 27b^2) \neq 0$ which implies that the above curve has no "singular points".

Remark 1.4.1.2 *We shall define a natural abelian group structure on the set*

$$E(K) = \{(x, y) \in K \times K : y^2 = x^3 + ax + b\} \cup \{O\} \tag{1.2}$$

1.4.2 The Group Law

Let E be an elliptic curve over a K, given by equation (5.1). We begin by defining a binary operation '+' on $E(K)$. We can use geometry to make the points of an elliptic curve into a group.

Theorem 1.4.2.1 *The addition law on an elliptic curve, E, as given in the above algorithm has the following properties:*

1. $P + O = O + P = P \quad \forall P \in E$
2. $P + (-P) = (-P) + P = O \quad \forall P \in E$
3. $P + (Q + R) = (P + Q) + R \quad \forall P, Q, R \in E$
4. $P + Q = Q + P \quad \forall P, Q \in E$

In other words, the addition law '+' makes the points of E into an abelian group.

1.4.3 Elliptic Curve Groups over Prime Field $\mathbb{F}_p$

The addition operation in an elliptic curve is the counterpart to modular multiplication in common public-key cryptosystems and scalar multiplication is the counterpart to modular exponentiation.

Now given a message, m, we must first choose a large integer, k and a suitable elliptic curve, $E(\mathbb{F}_p)$. We must then embed the message m onto a point P on the curve. We describe the method below to achieve this task:
Let $p > 3$ be an odd prime. An elliptic curve E over $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b \tag{1.3}$$

where where $a, b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \neq 0 \pmod{p}$. The set $E(\mathbb{F}_p)$ consists of all points

$(x, y) \in \mathbb{F}_p$, which satisfy the defining equation (1.2), together with a special point O called the point at infinity.

Addition Formula: There is a rule, called the chord-and-tangent rule, for adding two points on an elliptic curve $E(\mathbb{F}_p)$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{F}_p)$ forms a group with O as its identity element. We explain Point Addition and Point Doubling geomtrically and then state the algebraic formulas. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve E. Then the sum of P and Q, denoted by $R = (x_3, y_3)$ is defined geometrically as follows,depicted in Figure 1. First draw the line through P and Q which intersects the elliptic curve in a third point say R'. The R is the reflection of R' in the x-axis. If $P = (x_1, y_1)$, then the double of P, denoted by $R = (x_3, y_3)$ is defined geomtrically as follows, depicted in Figure 2. First draw the tangent line to the elliptic curve at P. This line intersects the elliptic curve in a second point say R' and so R is the reflection of this point in the x-axis. The following algebraic formulae for the sum of two points and the double of a point can now be derived from the geometric description.

1. $P + O = O + P = P \ \ \forall P \in E(\mathbb{F}_p)$

2. If $P = (x, y) \in E(\mathbb{F}_p)$, then $(x, y) + (x, -y) = O$ where $(x, -y)$ denotes $-P$ which is a point on the curve.

3. **(Point Addition)** Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, where $P \neq Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{1.4}$$

and

$$y_3 = \left(\frac{y_2 - y_1}{x_1 + x_2}\right)(x_1 - x_3) - y_1 \tag{1.5}$$

4. **(Point Doubling)** Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ where $P \neq -P$. Then $2P = (x_3, y_3)$ where

$$x_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)^2 - 2x_1 \tag{1.6}$$

and

$$y_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)^2 (x_1 - x_3) - y_1 \tag{1.7}$$

Theorem 1.4.3.1 *Let E be an elliptic curve over the finite field $\mathbb{F}_q$. If $E(\mathbb{F}_q)$ has order N we have*

$$|N - (q + 1)| \leq 2\sqrt{q} \tag{1.8}$$

1.4.4 Elliptic Curve Groups over Binary Field $\mathbb{F}_{2^m}$

An elliptic cuve E over $\mathbb{F}_{2^m}$ is defined by an equation of the form

$$y^2 + xy = x^3 + ax^2 + b \tag{1.9}$$

where $a, b \in \mathbb{F}_{2^m}$, and $b \neq 0$. The set $E(\mathbb{F}_{2^m})$ consists of all points $(x, y) \in \mathbb{F}_{2^m}$, which satisfy the defining equation (1.4), together with a special point O called the point at infinity.

Addition Formula: As with elliptic curves over $\mathbb{F}_p$, there is a chord-and-tangent rule for adding points on an elliptic curve $E(\mathbb{F}_{2^m})$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{F}_{2^m})$ forms a group with O as its identity element.
The algebriac formula for the sume of two points and the double of a point are the following:

1. $P + O = O + P = P \ \ \forall P \in E(\mathbb{F}_{2^m})$

2. If $P = (x, y) \in E(\mathbb{F}_{2^m})$, then $(x, y) + (x, x + y) = O$ where $(x, x + y)$ denotes $-P$ which is a point on the curve.

3. **(Point Addition)** Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ and $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, where $P \neq Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \tag{1.10}$$

   and

$$y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 \tag{1.11}$$

4. **(Point Doubling)** Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ where $P \neq -P$. Then $2P = (x_3, y_3)$ where

$$x_3 = {x_1}^2 + \frac{b}{{x_1}^2} \tag{1.12}$$

   and

$$y_3 = {x_1}^2 + \left(x_1 + \frac{y_1}{x_1}\right) x_3 + x_3 \tag{1.13}$$

1.4.5 Elliptic Curve Discrete Logarithm Problem (ECDLP)

Elliptic curve cryptosystems rely on the difficulty of solving the ECDLP. If an adversary is able to solve the ECDLP then the adversary will be able to break the system. Therefore, it is of great importance to understand the methods of tackling the ECDLP.

The Elliptic Curve Discrete Logarithm Problem can be stated as follows:

Definition 1.4.5.1 *Let $E(\mathbb{F}_q)$ be an elliptic curve defined over $\mathbb{F}_q$, where $q = p$ or $q = p^m$ for some $m \in \mathbb{N}$. Suppose that $P \in E(\mathbb{F}_q)$, and that $Q \in E(\mathbb{F}_q)$ such that $Q \in \langle P \rangle$ i.e. the subgroup generated by P. Determine the unique integer $k \in [0, n-1]$ such that $Q = kP$. The integer k is called the discrete logarithm of Q to the base P, denoted as $k = log_P Q$.*

The elliptic curve parameters for cryptographic schemes must be carefully chosen in order to resist all known attacks on the ECDLP.
General Attacks: These are attacks that work in the general setting regardless of properties of a given elliptic curve. [26]

1. **Exhaustive Search**: One way to attack the ECDLP is to perform an exhaustive search when the points P and Q are given. Since, in practice, P is chosen to have significantly large order, this then makes the exhaustive search infeasible.

2. **Baby-Step, Giant-Step Algorithm**: This attack uses a combination of computational power and storage in an attempt to solve the ECDLP. Let $E(\mathbb{F}_q)$ be an elliptic curve with generator P. Suppose that P has order n, and let $Q \in \langle P \rangle$. Suppose that we want to solve $Q = kP$. Set $m = \lceil \sqrt{n} \rceil$ and compute mP. We now make a list of iP for $0 \leq i < m$ and store this list. We can now compute $Q - j(mP)$, for $0 \leq j \leq m-1$ until we have found a match from the list that we have stored. Once we have a match from the list we then have the following:
$$iP = Q - j(mP)$$
and hence
$$Q = iP + j(mP)$$
Therefore we have solved the ECDLP since $k \equiv i + jm \pmod{n}$ [27]. This attack takes at most $\sqrt{n}$ operations and stores $\sqrt{n}$ values in a list to check for a match. Thus, the expected running time of this algorithm is $O(\sqrt{n})$. We can make this more effecient : When we compute the points iP, we only need to store half of these values i.e. we only need to compute iP for $0 \leq i \leq \frac{m}{2}$, and then we can check if $Q - j(mP) = \pm i$.

3. **Pollard's ρ-method**:

   (a) **Method 1**: Let $E(\mathbb{F}_q)$ be an elliptic curve and $P \in E(\mathbb{F}_q)$. Suppose that P has order n, where n is prime, an let $Q \in \langle P \rangle$. Suppose that we want to solve $Q = kP$. In this attack, we will attempt to find distinct pairs of integers (a, b) and (a', b') modulo n such that $aP + bQ = a'P + b'Q$. Rearranging this we can obtain a solution k, namely $k \equiv (a - a')(b' - b)^{-1} \pmod{n}$. (Note that since we assumed n to be prime $(b' - b)$ can be inverted). One method for finding these pairs of integers is to simply select $a, b \in [0, n-1]$ uniformly at random, compute the point $aP + bQ$ and then store the triple $(a, b, aP + bQ)$. We continue to generate pairs (a, b) uniformly at random and check these against all previously stored triples until we find a pair (a', b') with $a'P + b'Q$ where $(a, b) \neq (a', b')$. When this happens we have solved the ECDLP and we can rearrange $aP + bQ = a'P + b'Q$ as $(a - a')P = (b' - b)Q = (b' - b)(kP)$ and thus, $k \equiv (a - a')(b' - b)^{-1} \pmod{n}$. The birthday problem governs the expected running time of this algorithm. This method gives an expected running time of

$O(\sqrt{\frac{\pi n}{2}})$ but unfortunately requires approximately $O(\sqrt{\frac{\pi n}{2}})$ amount of storage for the triples we have computed.

(b) **Method 2**: We now discuss a second method which has roughly the same running time, but uses less storage as we are no longer required to store ordered triples in order to find a collision. Instead of storing a list of triples, we define a function $f : \langle P\rangle \to \langle P\rangle$ so that for any $X \in \langle P\rangle$ and $a, b \in [0, n-1]$ with $X = ap + bQ$, we can easily compute $f(X) = X'$ and $a', b' \in [0, n-1]$ with $X' = a'P + b'Q$. We can define such a function by partitioning $\langle P\rangle$ into L sets of roughly equal size say $\{S_1, S_2, \ldots, S_L\}$. We define a second function H so that $H(X) = j$ if $X \in S_j$. Then $a_j, b_j \in [0, n-1]$ are chosen uniformly at random for $1 \leq j \leq L$. Now our function $f : \langle P\rangle \to \langle P\rangle$ is defined by

$$f(X) = X + a_j P + b_j Q$$

where $j = H(X)$. So, if $X = aP + bQ$, then $f(X) = X' = a'P + b'Q$ where $a' = a + a_j \pmod n$ and $b' = b + b_j \pmod n$. This then determines a sequence of points $\langle P\rangle$. Since $\langle P\rangle$ is finite we will eventually obtain a collision, thus obtaining our pairs of integers (a, b) and (a', b') and enabling us to solve the ECDLP.

4. **Pohlig-Hellman Method**: Suppose that we are given an elliptic curve $E(\mathbb{F}_q)$, a point $P \in E(\mathbb{F}_q)$ of order n and $Q \in \langle P\rangle$. We want to solve for the unique integer k such that $Q = kP$. Suppose further that we know the factorization of n, say $n = \prod_{i=1}^{r} \ell_i^{e_i}$ where each ℓ_i is prime. We will now solve for k by reducing the problem to solve for values of $k_i \equiv k \pmod{\ell_i^{e_i}}$ for $0 \leq i \leq r$, which gives us a system of congruences modulo each prime ℓ_i namely,

$$\begin{array}{rcll} k & \equiv & k_1 & (\text{mod } \ell_1^{e_1}) \\ k & \equiv & k_2 & (\text{mod } \ell_2^{e_2}) \\ \vdots & & & \\ k & \equiv & k_r & (\text{mod } \ell_r^{e_r}) \end{array} \tag{1.14}$$

The Theorem 5.1.2.12 guarentees the existence of a unique solution namely k. We look at how this works. For the moment we fix a prime say $\ell_1^{e_1}$. We compute k_1 as follows. We write the base ℓ_1 representation of k_1,

$$k_1 \equiv a_0 + a_1\ell_1 + a_2\ell_2^2 + \cdots + a_{e_1-1}\ell_1^{e_1-1} \pmod{\ell_1^{e_1}} \text{ where each } a_i \in [0, l_1 - 1] \tag{1.15}$$

We begin by computing a list of small values for each prime divisor ℓ_i of n.

Set $T_i = \{j(\frac{n}{l_i}) : 0 \leq j \leq \ell_i - 1\}$. We will look for a match with these points and values that we will determine below. When we find a match we have solved for a given coeddicient in the base ℓ_1 expansion of k. We can now compute the following,

$$\begin{aligned} \frac{n}{\ell_1}Q &= \frac{n}{\ell_1}([a_0 + a_1\ell_1 + \cdots + a_{e_1-1}\ell_1^{e_1-1}]P) \\ &= a_0\frac{n}{\ell_1}P + ([a_1 + a_2\ell_1 + \ldots])nP \\ &= a_0\frac{n}{\ell_1}P \end{aligned}$$

Thus we can now look in our list T_1, find the matching point in the list and read off the coefficient a_0.

To solve for a_1, we have to change our starting point. Since we have already solved for a_0 we can make use of it and set $Q_1 = Q - a_o P$ then perform the above calculation using Q_1 instead. If we multiply (1.15) by $\frac{n}{\ell_1^2}$ after a_0 has been removed and this will give us

$$\begin{aligned}\frac{n}{\ell_1^2}Q &= ([a_1 + a_2\ell_1 + \dots])\frac{n}{\ell_1}P \\ &= a_1\frac{n}{\ell_1}P\end{aligned}$$

and again we look in our list T_1 for a matching solution. This gives us a result for a_1. We continue in this manner until we have solved for each coefficient in the base ℓ_1 expansion of k_1. We then continue and solve for each k_i in the same manner. When this is done we solve the system as in (1.14) and recover the original value of k thus solving the ECDLP.

The expected running time of this algorithm is $O(\sqrt{\ell'})$ where ℓ' is the largest prime divisor of n. In practice this attack becomes infeasible when n has a large prime divisor. If this is the case, it then becomes difficult to make and store the list T to find matches, let alone attempting to solve for k in its base ℓ expansion.

1.4.6 Elliptic Curve Operations

Chapter 2

Digital Signature Schemes

2.1 Introduction

In this chapter we study the concept of digital signature schemes, which is one of the most important cryptographic primitives enabled by Public-key Cryptography. These methods allow the authentication of messages through the use of asymmetric encryption systems in which the sender and receiver don't need to possess a common, but secret, key. Digital signatures are cryptographic primitives which are fundamental in providing entity authentication, data origin authentication, data integrity and non-repudiation. The functionality provided by digital signature can be stated as follows:

- **Authentication**: Digital signature provides authentication of the source of the messages as a message is signed by the private key of the sender which is only known to him/her. Authentication is highly desirable in many applications such as GNSS. In the context of GNSS, Navigation Message Authentication (NMA) denotes the authentication of satellite signals by means of digitally signing the modulated navigation data which we discuss in section 2.3.2.

- **Integrity**: Digital signature provides integrity as digital signature uniquely associate with corresponding message. i.e. After signing a message a message cannot be altered if someone does do it will invalidate the signature. There is no efficient method to change message and its signature to produce a new message and valid signature without having private key. So both sender and receiver dont have to worry about in transit alteration. The assurance that data has not been altered by unauthorized or unknown means

- **Non-repudiation**: Non-repudiation is a service that is used to provide assurance of the integrity and origin of data in such a way that the integrity and origin can be verified by a third party. This service prevents an entity from successfully denying involvement in a previous action. Non-repudiation is supported cryptographically by the use of a digital signature that is calculated by a private key known only by the entity that computes the digital signature.

A signature scheme consists of two components: a signing algorithm and a verification algorithm.

Notation	Meaning
$\mathcal{M}$	a set of elements called the message space
$\mathcal{M}_S$	a set of elements called the signing space
$\mathcal{S}$	a set of elements called the signature space
R	a 1-1 mapping from $\mathcal{M}$ to $\mathcal{M}_S$ called the redundancy function
$\mathcal{M}_R$	the image of R (i.e., $\mathcal{M}_R = Im(R)$)
R^{-1}	the inverse of R (i.e., $R^{-1} : \mathcal{M}_R \rightarrow \mathcal{M}$)
$\mathcal{R}$	a set of elements called the *indexing set for signing*
H	a one-way hash function with domain $\mathcal{M}$
$\mathcal{M}_H$	the image of H (i.e., $H : \mathcal{M} \rightarrow \mathcal{M}_H$); $\mathcal{M}_H \subseteq \mathcal{M}_S$ called the hash space value

Table 2.1: Notation for digital signature mechanisms
[30]

Before we provide the definition of a digital signature, we fix notation in the table below:
Note:

1. *(messages)* $\mathcal{M}$ is the set of elements to which a signer can affix a digital signature.
2. *(signing space)* $\mathcal{M}_S$ is the set of elements to which the signature transformations are applied. The signature transformations are not applied directly to the set $\mathcal{M}$
3. *(signature space)* $\mathcal{S}$ is the set of elements associated to messages in $\mathcal{M}$. These elements are used to bind the signer to the message.
4. *(indexing set)* $\mathcal{R}$ is used to identify specific signing transformations.

Digital signature schemes can be classified into the following two categories summarized below:

1. **Signature Schemes with Appendix** - when the whole message has to be stored and/or transmitted along with the signature. It is the most common type of scheme in use wherein the original message is needed for verification.Thus, digital signature schemes with appendix require the original message as input to the verification algorithm.
2. **Signature Schemes with Message Recovery** - when the whole message, or part of it, can be recovered from the signature. This type of scheme is rarely used. In such schemes the verification produces from a signature the original message and some additional data to verify its correctness. Thus, digital signature schemes with message recovery do not require the original message as input to the verification algorithm.

Algorithm 2.1.1 *A digital signature scheme with appendix rely on cryptographic hash functions rather than customized redundancy functions, and are less prone to existential forgery.*

Key generation for digital signature schemes with appendix

1. *Each entity A should select a private key which defines a set $\mathcal{S}_A = \{S_{A,k} : k \in \mathcal{R}\}$ of transformations. Each $S_{A,k}$ is a 1-1 mapping from $\mathcal{M}_H$ to $\mathcal{S}$ and is called a signing transformation.*

2. *$\mathcal{S}_A$ defines a corresponding mapping V_A from $\mathcal{M}_H \times \mathcal{S}$ to* {T,F} such that

$$V_A(\widetilde{m}, s^*) = \begin{cases} \text{T} & \text{if } S_{A,k}(\widetilde{m}) = s^* \\ \text{F} & \text{otherwise} \end{cases}$$

   for all $\widetilde{m} \in \mathcal{M}_H$, $s^* \in \mathcal{S}$; here, $\widetilde{m} = H(m)$ for $m \in \mathcal{M}$. V_A is called a verification transformation and is constructed such that it may be computed without knowledge of the signer's private key.

3. A's public key is V_A and A's private key is the set $\mathcal{S}_A$.

Signature generation and verification for digital signature schemes with appendix

1. *Signature generation: Entity A should do the following:*
   (a) *Select an element $k \in \mathcal{R}$*
   (b) *Compute $\widetilde{m} = H(m)$ and $s^* = S_{A,k}(\widetilde{m})$*
   (c) *A's signature for m is s^*. Both m and s^* are made available to entities which may wish to verify the signature.*

2. *Verification: Entity B should do the following:*
   (a) *Obtains A's authentic public key V_A*
   (b) *Compute $\widetilde{m} = H(m)$ and $u = V_A(\widetilde{m}, s^*)$*
   (c) *Accept the signature if and only if $u =$* T

Algorithm 2.1.2 *A digital signature scheme with message recovery is scheme for which a priori knowledge of the message is not required for the verification algorithm. This feature is of use for short messages.*

Key generation for digital signature schemes with message recovery

1. *Each entity A should select a set $\mathcal{S}_A = \{S_{A,k} : k \in \mathcal{R}\}$ of transformations. Each $S_{A,k}$ is a 1-1 mapping from $\mathcal{M}_S$ to $\mathcal{S}$ and is called a signing transformation.*

2. *$\mathcal{S}_A$ defines a corresponding mapping V_A with that property that $V_A \circ S_{A,k}$ is the identity map on $\mathcal{M}_S$ for all $k \in \mathcal{R}$. V_A is called a verification transformation and is constructed such that it may be computed without knowledge of the signer's private key.*

3. *A's public key is V_A and A's private key is the set $\mathcal{S}_A$.*

Signature generation and verification for digital signature schemes with message recovery*: Entity A produces a signature $s \in \mathcal{S}$ for a message $m \in \mathcal{M}$, which can later be verified by B. The message m is recovered from s.*

1. *Signature generation: Entity A should do the following:*
   (a) *Select an element* $k \in \mathcal{R}$
   (b) *Compute* $\widetilde{m} = R(m)$ *and* $s^* = S_{A,k}(\widetilde{m})$
   (c) *A's signature for* m *is* s^* *which is made available to entities which may wish to verify the signature and recover* m *from it.*
2. *Verification: Entity B should do the following:*
   (a) *Obtains A's authentic public key* V_A
   (b) *Compute* $\widetilde{m} = V_A(s^*)$
   (c) *Verify that* $\widetilde{m} \in \mathcal{M}_R$*; if not reject the signature.*
   (d) *Recover* m *from* $\widetilde{m}$ *by computing* $R^{-1}(\widetilde{m})$

Definition 2.1.3 *A redundancy function* R *and its inverse* R^{-1} *are publicly known. Selecting an appropriate* R *is critical to the security of the system. We illustrate this as follows, suppose that* $\mathcal{M}_R = \mathcal{M}_S$*. Suppose* R *and* $S_{A,k}$ *are bijections from* M *to* $\mathcal{M}_R$ *and* $\mathcal{M}_S$ *to* $\mathcal{S}$ *respectively. This implies that* $\mathcal{M}$ *and* $\mathcal{S}$ *have the same number of elements. Then for any* $s^* \in \mathcal{S}$ *we have* $\mathcal{V}(s^*) \in \mathcal{M}_R$ *and it is trivial to find messages* m *and corresponding signatures* s^* *which will be accepted by the verification algoritm as follows:*

1. *Select random* $k \in \mathcal{R}$ *and random* $s^* \in \mathcal{S}$
2. *Compute* $\widetilde{m} = \mathcal{V}_{\mathcal{A}}(s^*)$
3. *Compute* $m = R^{-1}(\widetilde{m})$

The element s^* *is a valid signature for the message* m *and was created without knowledge of the set of signing trasnformations* S_A

2.2 Role of Cryptographic Hash Functions in Digital Signature Schemes

Cryptographic hash functions play a fundamental role in modern cryptography. Our focus is resctricted to hash functions and their use to data integrity and message authentication. Hash functions create a short fingerprint of the input data, and if the data is altered, in general the fingerprint will no longer be valid. In this way, hash functions can be used to provide assurance of data integrity, and therefore play a vital role in the construction of signature schemes. More precisely, with digital signatures, a long message is usually hashed (using a publicly available hash function) and only the hash value is signed. The party receiving the message then hashes the received message, and verifies that the received signature is correct for this hash value. This saves both time and space compared to signing the message directly, which would typically involve splitting the message into appropriate-sized blocks and signing each block individually. Note here that the inability to find two messages with the same hash value is a security requirement, since otherwise, the signature on one message hash-value would be the same as that

on another, allowing a signer to sign one message and at a later point in time claim to have signed another.

There are two main requirements we have for a hash function used in our digital signature schemes: one-wayness and collision free. The concept of one-wayness with respect to hash functions is similar to the concept of a one-way function in public-key cryptography.
Hash functions are useful in signature schemes for the following reasons:

1. Public-key signing algorithms tend to be very slow, so taking the hash value of a large message and signing the resulting (small) value is much more efficient than simply signing the whole message.

2. If you wanted to add multiple signatures to a message, the resulting signed message would be many times the size of the original. If we did the same thing, only signing the hash value of the message, we would only be adding a small overhead for each signature added.

A cryptographic hash function can provide assurance of data integrity. Let h be a hash function and let x be some data, say a binary string of arbitrary length. The corresponding fingerprint is often referred to as the 'message digest' and defined to be $y = h(x)$. Computing a message digest from a piece of message is much faster than encrypting the message with a public-key algorithm, so message digests can be used to speed up digital signature algorithms.

Definition 2.2.1 *A hash family is a four-tuple ($\mathcal{X}$,$\mathcal{Y}$,$\mathcal{K}$,$\mathcal{H}$) where the following conditions are satisfied:*

1. *$\mathcal{X}$ is a set of possible messages*

2. *$\mathcal{Y}$ is a finite set of possible message digests or authentication tags*

3. *$\mathcal{K}$, the keyspace, is a finite set of possible keys*

4. *For each $k \in \mathcal{K}$, there is a hash function $h_k \in \mathcal{H}$ such that $h_k : \mathcal{X} \to \mathcal{Y}$*

Definition 2.2.2 *A hash function is a function H which has as a minimum the following two properties:*

1. *Compression: H maps an input x of arbitrary finite bitlength, to an output $H(x)$ of fixed bitlength n.*

2. *Ease of Computation: Given H and an input x, $H(x)$ is easy to compute.*

Properties 2.2.3 *For a hash function H with inputs x,x' and outputs y,y' we have the following important properties:*

1. *Preimage resistance: for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $H(x') = y$ when given any y for which a corresponding input is not known.*

2. *Second preimage resistantce: it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a second preimage $x' \neq x$ such that $H(x) = H(x')$.*

3. *Collision resistance: it is computationally infeasible to find any two distinct inputs x,x' which hash to the same output, i.e., such that $H(x) = H(x')$. (There is a free choice of inputs.)*

Remark 2.2.4 *To understand what easy, hard and computationally infeasible means please refer to [30]*

Remark 2.2.5 *SHA-1 was developed by the NSA. SHA-1 processes input data in 512-bit blocks, and generates 116-bit message digests. SHA-1 pads the message by adding a 1 bit to the end, followed by as many 0 bits as needed to make the length a multiple of 512 bits. Then a 64-bit number containing the message length before padding is XOR into the low-order 64 bits. [30]*

Hash Function vs. Digital Signature:
A hash value can be computed by anyone, whereas a signature can only be computed by someone knowing the secret key, so their purpose is very different. A hash can be used to verify, that the message has not been altered only if the hash value comes from, a trusted source. Sending the hash value along with the message does not give you any guarantee. A signature on the other hand proves, that the message has not been altered after somebody knowing that particular secret key has signed it. That is, a signature can be used to achieve integrity and authenticity. Signatures and hash functions are related in the way, that many signature schemes us a hash function as a building block.

2.2.1 Navigation Message Authentication

In most peoples minds, privacy is the goal most strongly associated to cryptography. But message authentication is arguably even more important. Indeed we may or may not care if some particular message we send out stays private, but we almost certainly do want to be sure of the originator of each message that we act on. Message authentication guarantees this. Message authentication allows one party - the sender to send a message to another party - the receiver in such a way that if the message is modified during transmission, then the receiver will almost certainly detect this. Message authentication is also called data-origin authentication, since it authenticates the point-of-origin for each message. Message authentication is said to protect the integrity of messages, ensuring that each message that is received and deemed acceptable is arriving in the same condition that it was sent out with no bits inserted, missing, or modified. In the context of GNSS this is referred to as Navigation Message Authentication (NMA) wherein the transmission of information takes place over an insecure data channel. GNSS communication contain several areas of concern regarding the confirmation of the origin of a message and/or the identity of its sender. In GNSS specifically two different kinds of authentication arise:

1. **User Authentication**: Addresses the proof of the identity of a user against a monitoring entity. The purpose of user authentication is to prevent unauthorized entities from using the service.

2. **Signal Authentication**: Refers to the proof that a received signal indeed originates from the claimed source in our case a navigation satellite.

Cryptographic digital signature protocols enable receivers to verify the origin of the signed messages.

2.3 Digital Signature Protocols

We discuss in detail the protocols and cryptosystems that can be used to send signed messages. A digital signature scheme typically consists of three algorithms namely key generation, signature creation and signature verification. We focus our discussion on ElGamal type signature schemes and its variants with appendix and message recovery.

2.3.1 Basic ElGamal Digital Signature

The ElGamal Signature Scheme is non-deterministic; which means that there are many valid signatures for any given message and the verification algorithm must be able to accept any of these valid signatures as authentic. This signature scheme is based on the difficulty of computing discrete logarithms. Suppose we want to sign a message using ElGamal Digital Signature Scheme, we can generate a signature corresponding to message m as follows:

Key Generation: Each entity creates a public key and corresponding private key. Each entity A should do the following:

1. Generate a large random prime p and a generator g of the multiplicative group ${\mathbb{Z}_p}^*$.
2. Select a random integer x such that $1 < x < p - 1$.
3. Compute $y = g^x \pmod{p}$.
4. A's public key is (p, g, y).
5. A's private key is x.

Signature Generation: Entity A signs a binary message m of arbitrary length.

1. Select randomly an integer t such that $1 < t < p - 1$ with $\gcd(t, p - 1) = 1$
2. Compute $r = g^t \pmod{p}$ (eg., using Algorithm 5.1.4.5)
3. Compute $t^{-1} \pmod{(p - 1)}$ (eg., using Algorithm 5.1.4.4)
4. Compute $s = (H(m) - xr)t^{-1} \pmod{(p - 1)}$, where $H(m)$ is the hash of the message to be signed.
5. If $s = 0$ then start again with a new t.
6. The pair (r, s) is the digital signature of the message m. The signer A repeats these steps for every signature.

Signature Verification: To verify A's signature (r, s) on m, B should do the following:

1. Obtain A's authentic public key (p, g, y).
2. Verify that $1 \leq r \leq p-1$; if not, reject the signature
3. Compute $v_1 = y^r r^s \pmod{p}$
4. Compute $H(m)$ and $v_2 = g^{H(m)} \pmod{p}$
5. The signature is declared valid if and only if $v_1 = v_2$

The ElGamal Signature Generation algorithm is probabilistic meaning that it employs a random number t to digitally sign a message m. Also a cryptographic hash function H is used to turn an arbitrarily long message m into a typically much shorter value $H(m)$ that is then digitally signed.

We discuss the above algorithm in more detail. The algorithm takes as input a private signing key x (together with the domain parameters p and g) and a message m, and it generates as output the digital signature for the message m. The signature, in turn consists of two integers $1 \leq r \leq p-1$ and $1 \leq s \leq p-2$. The algorithm also employs an internal random value t and it's inverse $t^{-1} \pmod{(p-1)}$. It operates in three steps:

1. A number $1 < t < p-1$ with $\gcd(t, p-1) = 1$ is randomly selected. The requirement $\gcd(t, p-1) = 1$ suggests that t has an inverse element $t^{-1} \pmod{(p-1)}$, i.e., $tt^{-1} = 1 \pmod{(p-1)}$. The inverse element is used in step 3 and it can be determined using the Extended Euclid algorithm.
2. t is used to compute $r = g^t \pmod{p}$. Again, t is an element in $\mathbb{Z}_p{}^*$. The resulting value r represents the first component of the ElGamal signature for the message m.
3. The message m is hashed with H, and the result $H(m)$ is used to compute $s = (H(m) - xr)t^{-1} \pmod{(p-1)}$ and if the resulting value is 0 then the algorithm is applied again with another value of t. Hence, the resulting number must be between 1 and $p-2$. The resulting number s represents the second component of the ElGamal signature for message m.
4. Note that the algorithm can be made more efficient using precomputation. In fact, it's possible to select $t \in (1, p-1)$ and precompute $r = g^t \pmod{p}$ and $t^{-1} \pmod{(p-1)}$ since both these values don't depend on a particular message m. If one has precomputed t, r and t^{-1}, then one can digitally sign a message m by hashing it and directly computing $s = (H(m) - xr)t^{-1} \pmod{(p-1)}$ and this computation is highly efficient.

The ElGamal Signature Verification algorithm is deterministic. It takes as input a verification key (p, g, y), a message m and an ElGamal signature (r, s), and it generates as output 1 bit saying whether (r, s) is valid signature for message m with respect to (p, g, y). The algorithm first verifies that $1 \leq r \leq p-1$ and $1 \leq s \leq p-2$ and then it verifies that

$$v_1 = v_2 \tag{2.1}$$

The signature is valid if and only if all verification checks are positive. Otherwise, the signature is rejected and considered to be invalid. The proof that signature verification works is as follows:

$$\begin{aligned} v_1 &= y^r r^s \\ &= g^{xr} g^{tt^{-1}(H(m)-xr)} \pmod{p} \\ &= g^{xr} g^{(H(m)-xr)} \pmod{p} \\ &= g^{xr} g^{-xr} g^{H(m)} \pmod{p} \\ &= g^{H(m)} \pmod{p} \end{aligned} \tag{2.2}$$

It is mandatory to verify that $1 \leq r \leq p-1$ otherwise it's possible to construct a new signature from a known signature. Let, for example (r, s) be the ElGamal signature of a message m and m' be another message for which an adversary wants to generate a valid signature. In this case, the adversary first generates:

$$u = H(m')H(m)^{-1} \pmod{(p-1)}$$

and then computes

$$s' = su \pmod{(p-1)}$$

and r' satisfies the following system of equivalences:

$$r' = ru \pmod{(p-1)}$$

$$r' = r \pmod{p}$$

The Chinese Remainder Theorem and the CRA can be used to compute r'. The pair (r', s') then represents the ElGamal signature for m' (or $H(m')$ respectively). We can show that,

$$\begin{aligned} y^{r'} r'^{s'} &= y^{ru} r^{su} \pmod{p} \\ &= g^{u(xr+ts)} \pmod{p} \\ &= g^{H(m')} \pmod{p} \end{aligned}$$

On the other hand, one can show that $r' \geq p$ and hence $1 \leq r \leq p-1$ is not fulfilled. Hence, verifying $1 \leq r \leq p-1$ is important. It is necessary to use a hash function H and not to directly sign messages. If a message m is signed directly then it is possible to existentially forge a digital signature i.e. if message m is signed directly, then the signature verification equivalence is $g^m = y^r r^s \pmod{p}$ and so its possible to select r, s and m such that the equivalence is satisfied. More specifically we can randomly select two integers u and v such that $\gcd(v, p-1) = 1$ and compute r, s and m as follows:

$$\begin{aligned} r &= g^u y^v \pmod{p} \\ s &= -rv^{-1} \pmod{(p-1)} \\ m &= su \pmod{(p-1)} \end{aligned}$$

With these values, we have

$$\begin{aligned} y^r r^s &= y^r g^{su} g^{sv} \pmod p \\ &= y^r g^{su} y^{-r} \pmod p \\ &= g^m \pmod p \end{aligned}$$

Hence the verification equation (2.1) is satisfied. Note that if we digitally sign $H(m)$ instead od m we can still generate signatures for hash values but due to the one-way property of hash functions its not feasible to compute m from $H(m)$.

2.3.2 Digital Signature Algorithm

The National Institute of Standards and Technology (NIST) has published the Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS) which makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the Digital Signature Algorithm (DSA). Because the DSA authenticates both the identity of the signer and the integrity of the signed information it can be used in a variety of applications. Digital Signature Algorithm (DSA) is a variant of the ElGamal digital signature scheme and is a digital signature scheme with appendix. It's security is based on the intractability of the discrete logarithm problem in prime order subgroups of ${\mathbb{Z}_p}^*$.

Domain Paramters

1. Select a 160-bit prime q (i.e., $2^{159} < q < 2^{160}$) and a 1024-bit prime p (i.e., a prime modulus p which is 512+64z bits long for $z \in \{0, \ldots, 8\}$) with the property that $q|p-1$
2. Select a generator g of the unique cyclic group of order q in ${\mathbb{Z}_p}^*$
3. Select an element $h \in {\mathbb{Z}_p}^*$ and compute $g = h^{(p-1)/q} \pmod p$. Repeat until $g \neq 1$
4. Domain parameters are p, q and g.

Key Generation: Each entity A in the domain with domain parameters (p, q, g) does the following

1. Select a random or pseudorandom integer x such that $1 \leq x \leq q-1$
2. Compute $y = g^x \pmod p$
3. A's public key is (p, q, g, y) and A's private key is x.

Note: (Generation of DSA primes p and q) We must select the prime q first and then try to find a prime p such that q divides $p-1$. The algorithm recommended by the DSS for accomplishing this is Algorithm

Signature Generation: Entity A signs a binary message m of arbitrary length.

1. Select a random or pseudorandom integer t, $1 \leq t \leq q-1$
2. Compute $r = (g^t \pmod p) \pmod q$. (eg., using Algorithm 5.1.4.5) If $r = 0$ go to step 1.

3. Compute $e = H(m)$ where H is the hash function SHA $-$ 1
4. Compute $t^{-1} \pmod q$ (eg., using Algorithm 5.1.4.4)
5. Compute $s = t^{-1}(e + xr) \pmod q$. If $r = 0$ or $s = 0$ then go to step 1.
6. A's signature for the message m is the pair (r, s)

Signature Verification: To verfiy A's signature (r, s) on m, B obtains authentic copies of A's domain parameters (p, q, g) and public key y and does the following

1. Verify that r and s are integers $\in \mathbb{Z}_q$; if not reject the signature.
2. Compute $e = H(m)$
3. Compute $w = s^{-1} \pmod q$
4. Compute $u_1 = ew \pmod q$ and $u_2 = rw \pmod q$
5. Compute $v = (g^{u_1} y^{u_2} \pmod p) \pmod q$
6. Accept the signature if and only if $v = r$

We discuss the above algorithm is more detail.
The DSA Signature Generation Algorithm requires and makes use of a hash function H that generates hash values of about the bit length of q. The DSA Signature Generation Algorithm consists of the 5 steps highlighted above. If $r = 0$ or $s = 0$ then it's recommended to recalculate the signature. At the end, the pair (r, s) with $1 \leq r, s \leq q - 1$ represents the digital signature for m or e respectively. Note that r and s are both 160-bit numbers which is in contrast to the ElGamal signature, in which r and s are both of the size of p and m. Consequently, a DSA signature is about 320 bits long, which is more than 6 times shorter than a corresponding ElGamal signature.
The DSA Signature Verification Algorithm is deterministic and can be used to verify DSA signatures. It takes as input a verification key (p, q, g, y), message m and a DSA signature (r, s) and it generates as output 1 bit saying whether (r, s) is a valid DSA signature for m with respect to (p, q, g, y). The algorithm first verifies that $r, s \in \mathbb{Z}_q$. If r or s is not in $\mathbb{Z}_q$, then the signature is considered to be invalid and rejected. If, the two conditions are satisfied, then the algorithm must compute the following values:

$$\begin{aligned} e &= H(m) \\ w &= s^{-1} \pmod q \\ u_1 &= ew \pmod q \\ u_2 &= rw \pmod q \\ v &= (g^{u_1} y^{u_2} \pmod p) \pmod q \end{aligned}$$

The signature is valid if and only if $v = r$. We provide a proof to see why this signature verification works:
We start with $s = t^{-1}(e + xr) \pmod q$ and hence $e = (ts - xr) \pmod q$. If we multiply both

sides by w and rearrange the congruence we obtain $we + xrw = t \pmod{q}$ which is the same as $u_1 + xu_2 = t \pmod{q}$. Raising g to the on both sides of the congruence yields us $(g^{u_1}y^{u_2} \pmod{p}) \pmod{q} = (g^t \pmod{p}) \pmod{q}$ and hence $v = r$ as desired.

2.3.3 Schnorr Digital Signature Algorithm

Schnorr signatures are like (EC)DSA signatures with two key differences namely:

1. The signing equation is simpler which allows for some optimizations
2. The hash function is applied to the concatenation of the message and the ephemeral key

The Schnorr digital signature scheme is a well known variant of the ElGamal digital signature scheme. As with the DSA, this technique employs subgroups of order q in ${\mathbb{Z}_p}^*$ where p is some large prime number. The method also requires a hash function $h : \{0,1\}^* \to \mathbb{Z}_q$. Key generation for the Schnorr signature scheme is the same as the DSA key generation, except for the fact that there are no constraints on the sizes of p and q but the size of the signatures are smaller. Its simplicity and effeciency (short signature length and the possibility of pre-computing exponentiations for very quick on-line signature generation) has attracted considerable attention.

Signature generation: Entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key. Entity A should do the following

1. Select a random secret integer t, $1 \leq t \leq q-1$
2. Compute $r = g^t \pmod{p}$. This computation is a precomputation step which is independent of the message m to be signed.
3. Compute $e = H(m||r)$ i.e. concatenate the message m with r
4. Compute $s = (xe + t) \pmod{q}$.
5. A's signature for m is the pair (s, e).

The Schnorr Signature Generation algorithm is probabilistic and it takes as input the domain parameters p, q and g, a private signing key x, and a message m and it generates as output a digital signature for m. The signature in turn consists of two integers e and s (with $0 \leq e, s \leq q-1$). Similar to the ElGamal Signature Generation algorithm, the Schnorr Signature generation algorithm begins with the selection of a random element t and the computation of $r = g^t \pmod{p}$. In the ElGamal scheme this value is part of the signature whereas in the Schnorr signature it is further processed. The Schnorr Signature Verification algorithm is deterministic.

Signature Verification: To verfit A's signature (s, e) on m, B should do the following:

1. Obtain A's authentic public key (p, q, g, y).
2. Compute $v = g^s y^{-e} \pmod{p}$ and $e' = H(m||v)$

3. Accept the signature if and only if $e' = e$.

To see that the verification works observe the following:
$v = g^s y^{-e} = g^s g^{-xe} = g^{s-xe} = g^t = x \pmod{p}$
Hence, $H(m||r) = H(m||v)$ and $e = e'$.

Performance Characteristics of the Schnorr Scheme: Signature generation in the Schnorr Digital Signature requires one exponentiation modulo p and one multiplication modulo q. The exponentiation modulo p could be done off-line or pre-computed. Depending on the hash algorithm used, the time to compute $H(mr)$ should be relatively small. Verification requires two exponentiations modulo p. These two exponentiations can be computed by Algorithm 5.1.4.6 at a cost of about 1.17 exponentiations. Using the subgroup of order q does not significantly enhance computational efficiency over the ElGamal Digital Signature scheme, but does provide smaller signatures (for the same level of security) than those generated by the ElGamal method. Most of the computation for signature generation can be completed in a preprocessing stage, independent of the message being signed. Hence, it can be done during idle time and not affect the signature speed. An attack against this preprocessing stage, but I dont think its practical. For the same level of security, the length of signatures is less for Schnorr than for RSA. For example, with a 140-bit q, signatures are only 212-bits long, less than half the length of RSA signatures. Schnorrs signatures are also much shorter than ElGamal signatures.
Elgamal/Schnorr and DSA signatures use a per-message secret key and are based on exponentiation.

2.3.4 Nyberg-Rueppel Digital Signature Algorithm without message recovery

The Nyberg-Rueppel (NR) signature algorithm is another adaption of the ElGamal signature scheme and is based on the intractability of solving the DLP in a group G. Assume G is a finite group of order $n \geq 3$ to avoid triviality. The message m is identified with an element of the group G. Entity A signs message m. Any entity B can verify A's signature and recover the message m from the signature.

Signature Generation: Entity A should do the following

1. Compute $\widetilde{m} = R(m)$
2. Select a random secret integer t, $1 \leq t \leq q-1$, and compute $r = g^{-t} \pmod{p}$.
3. Compute $e = \widetilde{m}r \pmod{p}$
4. Compute $s = ae + k \pmod{q}$
5. A's signature for m is the pair (e, s).

Signature Verification: To verify A's signature (e, s) on m, B should do the following:

1. Obtain A's authentic public key (p, q, g, y)

2. Verify that $0 < e < p$; if not, reject the signature
3. Verify that $0 \leq s < q$; if not, reject the signature
4. Compute $v = g^s y^{-e} \pmod{p}$ and $\widetilde{m} = ve \pmod{p}$.
5. Verify that $\widetilde{m} \in \mathcal{M}_R$; if not, then reject the signature
6. Recover $m = R^{-1}(\widetilde{m})$

In its plain form, NR is vulnerable to existential forgery attacks.
Proof of Signature Verification: If A created the signature then

$$v = g^s y^{-e} = g^{s-ae} = g^t \pmod{p}$$

. Thus,

$$ve = g^t \widetilde{m} g^{-t} = \widetilde{m} \pmod{p}$$

2.3.5 Nyberg-Rueppel Digital Signature Algorithm with message recovery

The Nyberg-Rueppel (NR) signature algorithm is another adaption of the ElGamal signature scheme and is based on the intractability of solving the DLP in a group G. Assume G is a finite group of order $n \geq 3$. Entity A signs message $m \in \mathcal{M}$. Any entity B can verify A's signature and recover the message m from the signature.

Signature Generation: Entity A should do the following

1. Compute $\widetilde{m} = R(m)$
2. Select a random secret integer t, $1 \leq t \leq q-1$, and compute $r = g^{-t} \pmod{p}$.
3. Compute $e = \widetilde{m} r \pmod{p}$
4. Compute $s = ae + k \pmod{q}$
5. A's signature for m is the pair (e, s).

Signature Verification: To verify A's signature (e, s) on m, B should do the following:

1. Obtain A's authentic public key (p, q, g, y)
2. Verify that $0 < e < p$; if not, reject the signature
3. Verify that $0 \leq s < q$; if not, reject the signature
4. Compute $v = g^s y^{-e} \pmod{p}$ and $\widetilde{m} = ve \pmod{p}$.
5. Verify that $\widetilde{m} \in \mathcal{M}_R$; if not, then reject the signature
6. Recover $m = R^{-1}(\widetilde{m})$

Proof of Signature Verification: If A created the signature then

$$v = g^s y^{-e} = g^{s-ae} = g^t \pmod{p}$$

. Thus,

$$ve = g^t \widetilde{m} g^{-t} = \widetilde{m} \pmod{p}$$

The only difference between NR signature generation and Schnorr signature generation is the way how s is computed.

2.4 Elliptic Curve Variants of Digital Signatures

2.4.1 Generalized ElGamal Digital Signature

The ElGamal digital signature scheme, originally described in the setting of the multiplicative group ${\mathbb{Z}_p}^*$, can be generalized in a straightforward manner to work in any finite abelian group G. We consider the case where the finite abelian group G is constructed from the set of points on an elliptic curve over a finite field $\mathbb{F}_q$. The discrete logarithm problem in groups of this type appears to be more difficult than the discrete logarithm problem in the multiplicative group of a finite field $\mathbb{F}_q$. This implies that q can be chosen smaller than for corresponding implementations in groups such as $G = {F_q}^*$. The algorithm described below requires a hash function $H : \{0,1\}^* \to \mathbb{Z}_n$ where n is the number of elements in G. It is assumed that each element $r \in G$ can be represented in binary so that $H(r)$ is defined.
Key Generation: Each entity selects a finite group G; generator of G; public and private keys. Each entity A should do the following:

1. Select an appropriate cyclic group G of order n with generator g
2. Select a random secret integer x, $1 \leq x \leq n-1$. Compute the group element $y = g^x$
3. A's public key is (g, y) together with a description of how to multiply elements in G in the case where G is an elliptic curve and A's private key is x.

Signature Generation: Entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key. Entity A should do the following:

1. Select a random secret integer t, $1 \leq t \leq n-1$ with $\gcd(t, n) = 1$
2. Compute the group element $r = g^t$
3. Compute $t^{-1} \pmod{n}$
4. Compute $H(m)$ and $H(r)$
5. Compute $s = t^{-1}(H(m) - xH(r)) \pmod{n}$.
6. A's signature for the message m is the pair (r, s)

Signature Verification: To verfiy A's signature (r, s) on m, B should do the following:

1. Obtain A's authentic public key (g, y)
2. Compute $H(m)$ and $H(r)$
3. Compute $v_1 = y^{H(r)}r^s$ and $v_2 = g^{H(m)}$
4. Accept the signature if and only if $v_1 = v_2$

2.4.2 Elliptic Curve ElGamal Digital Signature

Key Generation: Each entity A should do the following:

1. Choose an elliptic curve E over a finite field $\mathbb{F}_{\text{ıı}}$ such that the discrete logarithm problem is hard for $E(\mathbb{F}_{\text{ıı}})$
2. Choose a point $P \in E(\mathbb{F}_{\text{ıı}})$ such that the order n of P is a large prime.
3. Select a secret integer d and compute $Q = dP$
4. Select a function $f : E(\mathbb{F}_{\text{ıı}}) \to \mathbb{Z}$ such that its image should be large and only a small number of imputs should produce any given output.
5. A's public key is $(E, \mathbb{F}_{\text{ıı}}, f, P, Q)$ and A's private key is d. Note that n need not be made public , also it's secret doesn't affect the analysis of the security of the system. [27]

Signature Generation: Entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key. Entity A should do the following:

1. Select a random secret integer t with $\gcd(t, n) = 1$
2. Compute the group element $R = tP$
3. Compute $s \equiv t^{-1}(m - df(R) \pmod{n}$
4. A's signature for the message m is the pair (R, s)

Signature Verification: To verfiy A's signature (R, s) on m, B should do the following:

1. Obtain A's authentic public key
2. Compute $v_1 = f(R)Q + sR$ and $v_2 = mP$
3. Accept the signature if and only if $v_1 = v_2$

Proof of correction:

2.4.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.So instead of working in a subgroup of order q in $\mathbb{Z}_p^*$, we work in an elliptic curve group $E(\mathbb{Z}_p)$.

Key Generation

1. Select an elliptic curve E defined over $\mathbb{Z}_p$. The number of points in $E(\mathbb{Z}_p)$ should be divisible by a large prime n.
2. Select a point $P \in E(\mathbb{F}_p)$ of order n.
3. Select a random or pseudorandom integer d, such that $1 \leq d \leq (n-1)$
4. Compute $Q = dP$
5. A's public key is (E, P, n, Q) and A's private key is d

Signature Generation: To sign a message m, an entity A does the following:

1. Select a random or pseudorandom integer t such that $1 \leq t \leq n-1$
2. Compute $tP = (x_1, y_1)$ and convert x_1 to an integer X_1.
3. Compute $r = x_1 \pmod{n}$. If $r = 0$, then go to step 1.
4. Compute $t^{-1} \pmod{n}$
5. Compute $e = H^(m)$ where H is the Secure Hash Algorithm (SHA-1)
6. Compute $s = t^{-1}(e + dr) \pmod{n}$. If $s = 0$ then go to step 1.
7. A's signature for the message m is (r, s).

Signature Verification: To verify A's signature (r, s) on m, B obtains an authentic copy of A's domain parameters and associated public key Q. B then does the following:

1. Verify that r and s are integers in the interval $[1, n-1]$
2. Compute $e = \mathrm{SHA}^{-1}(\mathrm{m})$
3. Compute $w = s^{-1} \pmod{n}$
4. Compute $u_1 = ew \pmod{n}$ and $u_2 = rw \pmod{n}$
5. Compute $X = u_1P + u_2Q$
6. If $X = O$ then reject the signature. Otherwise, convert the x- coordinate x_1 of X to an integer X_1 and compute $v = X_1 \pmod{n}$
7. Accept the signature if and only if $v = r$.

Proof that the signature verification works: If a signature (r, s) on a message m was indeed generated by A, then $s = t^{-1}(e + dr) \pmod{n}$. Rearranging gives,

$$t = s^{-1}(e + dr) = s^{-1} + s^{-1}rd = we + wrd = u_1 + u_2 d \pmod{n}$$

Thus, $u_1 G + u_2 Q = (u_1 + u_2 d)G = tG$ and so $v = r$ as required.

Conceptually the ECDSA is simply obtained from the DSA by replacing the subgroup of order q of $\mathbb{Z}_p^*$ generated by g with the subgroup of points on an elliptic curve that are generated by G. The only significant difference between ECDSA and DSA is in the generation of r. The DSA obtains an integer r in the interval $[1, q-1]$ by taking the random element $X = g^k \pmod{p}$ and reducing it modulo q. The ECDSA generates r in the interval $[1, n-1]$ by taking the x-coordinate of the random point kG and reducing it modulo n.

2.4.4 Elliptic Curve Schnorr Digital Signature Scheme

The Elliptic Curve Schnorr Digital Signature Scheme is a variant of the Schnorr Digital Signature Scheme.

2.5 Accelerated Broadcast Authentication with Signature Amortization

2.5.1 Introduction

Asymmetric Key Cryptography is widely used in broadcasting areas for authentication. The below proposed system is a novel broadcast authentication scheme based on PKC with signature amortization. This scheme uses only one ECDSA signature to authenticate a group of broadcast messages. So, the signature overhead is amortized over that group of broadcast messages. However, this system shows low overhead and high security. Security of this system is as strong as typical broadcast authentication schemes by PKC. No time synchronization is required for this scheme and also it can achieve immediate authentication which would benefit GNSS. But ECDSA shows signature verification delay since its signature verification is much slower than signature generation. For the public-key based authentication technique, each message is transmitted along with the digital signature of the message which produced using the senders private key.

2.5.2 System Description and Notations

Digital Signature Amortization was proposed in the case of Wireless Sensor Networks (WSN) which consists of one or more powerful base stations and hundreds of sensor nodes. We draw a parallel of this scheme to our GNSS for our purposes. Broadcast transmission is one of the fundamental communication primitives. Eavesdropping on the broadcast transmission, an adversary is able to intercept broadcast messages or change the intercepted messages and re-broadcast them. These malicious messages may cause destruction of the entire network and so

a defense mechanism needs to be applied. For the sake of simplicity, we consider the case of one base station within a WSN (i.e. one satellite within GNSS) which can be extended to a WSN having multiple base stations (i.e. multiple satellites within GNSS). In addition to this we make the following assumptions in case of WSN which are stronger than those required in the context of GNSS:

1. The base station can't be compromised by adversaries (i.e. the satellite can't be comprised by adversaries in GNSS)
2. The sensor nodes are vulnerable to adversaries which can launch external as well as internal attacks (i.e. the receivers are vulnerable to adversaries which can launch external attacks; internal attacks may not occur as the receiver is with the user and he/she won't lauch an attack on their own device)
3. Sensor nodes are able to perform a limited number of PKC operations (i.e. receivers will obviously be able to perform a said number of PKC operations)
4. One sender broadcasts messages to many receivers (i.e. one satellite broadcasts signals/messages to multiple receivers)

We propose to apply the scheme in the context of GNSS where each satellite transmits to many potential receivers.
For convenience we make use of the following notations. The base station is denoted by bs. It holds a key pair (PR_{bs}) where PR_{bs} is the private key and PU_{bs} is the public key. PU_{bs} is assumed to be stored by all sensor nodes in the network. The sender is denoted by s. It holds a key pair (PR_s, PU_s) where PU_s is the public key and PR_s is the private key. Corresponding to sender s, receivers are denoted by vector $R_s = [r_i]_{i=1,\ldots,c}$. Broadcast messages from sender s to receivers in R_s are denoted by a vector $M = [m_i]_{i=1,\ldots,n}$. The broadcast messages in M are organized through extended blocks which is regarded as a unit authenticating broadcast messages in M. The extended blocks are denoted by a vector $EB = [EB_i]_{i=0,\ldots,k}^{T}$, in total $k+1$ extended blocks wherein the first extended block EB_0 contains an authenticator. An authenticator is the additional information to authenticate an extended block say a signature. Each of the other k extended blocks contains b broadcast messages in M and a specified authenticator.

Definition 2.5.2.1 *Authentication Relation (AR) consists of ordered pairs $\langle EB_i, EB_j \rangle$ where the authenticator in EB_i is used to authenticate EB_j.*

Definition 1 *Collision resistant hash H is the hash satisfying that it is computationally infeasible to find a pair of inputs (x, y) such that $x \neq y$ and $H(x) = H(y)$.*

2.5.3 Proposed PKC based Broadcast Authentication Scheme for GNSS

The proposed PKC based broadcast authentication scheme using signature amortization exploits one ECDSA signature to authenticate all broadcast messages. Only one signature is used

to authenticate the authenticator in EB_0. The authenticator in EB_0 is used to authenticate EB_1 that contains b broadcast messages in M and one authenticator. The authen- ticator in EB_1, in turn, is used to authenticate EB_2 that contains b broadcast messages in M and one authenticator. We continue the process until EB_k. As a result, all broadcast messages can be authenticated with only one signature while the overhead of the signature is amortized over them.
We first present procedures of signature amortization which is presented in three parts:

1. **Generating Extended Blocks Step**: Based on the definition 2.6.2.1, the basis to construct extended blocks is provided in the theorem below:

   Theorem 2.5.3.1 *All broadcast messages in M will be authenticated if EB_0 is authentic and all ordered pairs $\langle EB_{i-1}, EB_i\rangle$ for $1 \leq i \leq k$, belong to authenticated relation AR on EB.*
   Proof. *We proceed by mathematical induction. The first step is authentication of EB_1 and note that EB_1 is authenticated by the authenticator in EB_0 which is authentic. Thus, b broadcast messages in EB_1 will be authenticated by our notation. Now, let's assume that EB_{j-1} where $2 \leq j \leq k$ is authentic. Then, the authenticator in EB_{j-1} is used to authenticate EB_j since by definition $\langle EB_{i-1}, EB_i\rangle$ belongs to AR on EB. Thus, b messages in EB_j will be authenticated. By the induction hypothesis we can conclude that n broadcast messages in M will be authenticated.*

   The authenticators for the extended blocks can be generated by collision resistant hash as follows.
   All n broadcast messages in M is partitioned into k blocks namely $B_1, B_2, \ldots, B_k$ where every block B_i, $1 \leq i \leq k$ contains b messages such that $|B_i| = b$ These blocks comprise a vector $B = [B_i]^T_{i=1,2,\ldots,k}$ that is expressed in the matrix form as shown below in equation (2.1), note that we assume $n = kb$.

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_k \end{bmatrix} = \begin{bmatrix} m_1 & \cdots & m_b \\ \vdots & \vdots & \vdots \\ m_{(k-1)b+1}, & \cdots & m_{kb} \end{bmatrix} \tag{2.3}$$

   Messages in each row of B, accurately block B_i, are concatenated into a long string which is denoted by $CON(B_i)$ as shown in equation (2.2).

$$CON(B_i) = m_{(i-1)b+1}|| \ldots ||m_{ib}, 1 \leq i \leq k \tag{2.4}$$

   Then, $CON(B_i)$, $1 \leq i \leq k$ is padded with authenticators, denoted by $PAD(CON(B_i))$ as shown in equation (2.3). $CON(B_j)$, $1 \leq j \leq k-1$, is padded with digest d_{j+1} which is the digest of $PAD(CON(B_{j+1}))$, as shown in equation (2.4) where H is a collision resistant hash. Note that $CON(B_k)$ is padded with random characters which is denoted by d_{k+1} for consistency.

$$PAD(CON(B_i)) = CON(B_i)||d_{i+1}, 1 \leq i \leq k \tag{2.5}$$

$$d_{j+1} = H(PAD(CON(B_{j+1})), 1 \leq j \leq k-1 \tag{2.6}$$

Block B_i and digest d_{i+1} comprise an extended block EB_i, i.e., $EB_i = [B_i d_{i+1}], 1 \leq i \leq k$. Let $EB_i, 1 \leq i \leq k$ comprise a vector EB^*, i.e., $EB^* = [EB_i]^T_{i=0,\ldots,k} = [B_i d_{i+1}]^T_{i=0,\ldots,k}$. Because EB^* stands for an extended matrix of B, EB_i is called an extended block.
Note that exceptional EB_0 only contains an authenticator. The authenticator is digest d_1 that is the collision resistant hash value of $PAD(CON(B_1))$, i.e., $d_1 = H(PAD(CON(B_1)))$. The authenticity of d_1 is guarenteed by a signature generated by s with PR_s. Thus, $EB_0 = d_1||E(PR_s, d_1)$.
Thus, all extended blocks have been generated and for simplicity let all extended blocks comprise a vector EB, i.e., $EB = [EB_i]^T_{i=0,\ldots,k}$.
The following algorithm presents the description of generating EB:

Algorithm 1 - Generation of EB

(a) Partition n broadcast messages in M into k blocks $B_1, \ldots, B_k$ as shown in equation (2.1)

(b) Initialize d_{k+1} with a string of random characters

(c) **for** $i = k$; $i \geq 1$; $i = i - 1$ **do**

(d) Concatenate message in B_i to generate $CON(B_i)$ as shown in equation (2.2)

(e) Pad $CON(B_i)$ with digest d_{i+1} to generate $PAD(CON(B_i))$ as shown in equation (2.3)

(f) Compute digest d_i of $PAD(CON(B_i))$ with collision resistant hash as shown in equation (2.4)

(g) Let $EB_i = [B_i d_{i+1}]$

(h) **end for**

(i) Sign digest d_1 with sender s private key PR_s to generate $EB_0 = d_1||E(PR_s, d_1)$

(j) Let $EB = [EB_i]^T_{i=0,\ldots,k}$

So based on Theorem 2.6.2.1 and generating process of EB, authenticating n broadcast messages in M proceeds as follows: The signature in EB_0 is used to authenticate d_1. d_1 is used to authenticate EB_1 that contains B_1 and d_2. Then d_2 is used to authenticate EB_2 that contains B_2 and d_3. We continue this process till we reach EB_k and so finally n broadcast messages in M will be authenticated.

2. **Broadcasting Extended Blocks Step**: To ensure that EB_{i-1} reaches receivers before EB_i we follow sequential and reliable broadcast as described below
The sequential broadcast is that extended blocks are broadcast according to AR on EB. Sender s broadcasts EB_{i-1} before EB_i. For simplicity, messages in each extended block are broadcast according to their indices i.e., the sending sequence for EB_i is $m_{(i-1)b+1}, m_{(i-1)b+2}, \ldots, m_{ib}$. Digest d_{i+1} in EB_i could be sent together with a broadcast message in EB_i since the size of a digest is relatively small compared to the message

itself. On receiving a broadcast message m_j, a receiver in R_s checks whether m_j belongs to current extended block, EB_i, whose digest is, d_i, has been received and authenticated with EB_{i-1}. If m_j belongs to EB_i, the receiver tries authenticating EB_i and broadcasts m_j to its neighbours after a short backoff which means that all receivers exchange messages of current block EB_i with each other. During the exchange, a receiver may receive multiple copies of m_j from different neighbours. Thus, the receiver would obtain m_j with high probability but each transmission is not reliable. To resolve this, the unreliable transmission is counteracted by making full use of broadcast nature. So, if m_j belongs to the block EB_{i+1} which follows EB_i and EB_i has not been authenticated yet, that indicates all messages of EB_i have been broadcast. The receiver would not get missing messages later. Hence, the receiver buffers m_j and broadcasts an acknowledgement to ask for the missing messages belonging to EB_i. Soon after the authentication of EB_i, m_j will be broadcast. We employ the sequential broadcast and reliable broadcast to guarentee the successful authentication of extended blocks with low overhead and they ensure the integrity of M.

3. **Verifying Extended Blocks Step**: According to broadcasting extended blocks step, EB_0 reaches receivers in R first. d_1 is authenticated by the signature i.e.,
if $D(PU_s, E(PR_s, d_1)) = d_1$, then d_1 is authentic. Extended blocks in EB^* are authenticated in an efficient way, just using collision resistant hash. Digest d_i, $1 \leq i \leq k$, in EB_{i-1} that reaches receivers in R in advance is used to authenticate EB_i, that is, if $H(m_{(i-1)b+1}|| \ldots ||m_{ib}||d_{i+1}) = d_i$, EB_i is authentic.

We now present the details of the ECDSA signature:

Sender s and receivers in R_s establish elliptic curve domain parameters $T = (p, a, b, G, q, h)$ in advance. The storage of T in the sensor nodes before proceeding with broadcasting is an option. p is a prime that specifies the finite field $\mathbb{F}_p$. a and b are coefficients of the elliptic curve $y^2 = x^3 + ax + b \pmod{p}$ where $4a^3 + 27b^2 \neq 0 \pmod{p}$. G is the base point on the elliptic curve. q is a prime which denotes the order of the group G. h is the cofactor $|E_p(a, b)|/q$ where $|E_p(a, b)|$ is the number of points on the elliptic curve.

To sign the message digest d_1, sender s creates the key pair (PR_s, PU_s) which satisfies $PU_s = PR_sG$ where $PR_s \in \mathbb{F}_p$ and PU_s is a point on the elliptic curve. Then, we select a temporary key pair (u, U) that satisfies $U = uG$ where $u \in \mathbb{F}_p$ and U is a point on the elliptic curve. It computes $r = x_U \pmod{q}$ where x_U is the x coordinate of the point U, and $d_e = H(d_1)$ where H is the collision resistant hash. It sets $e = d_e$ if $\lceil log_2 q \rceil \geq L_{d_e}$ where L_{d_e} is the length of d_e. At last, it computes $w = u^{-1}(e + rPR_s) \pmod{q}$. The ECDSA signature is given by the pair (r, w). The complete expression of EB_0 is $EB_0 = d_1||r||w$.

Verification of d_1 proceeds in the following manner. The receiver computes $d_e = H(d_1)$ where H is a collision resistant hash. Then it sets $e = d_e$ if $\lceil log_2 q \rceil \geq L_{d_e}$ where L_{d_e} is the length of d_e. Otherwise, let e equal the leftmost $\lceil log_2 q \rceil$ bits of d_e. It computes $v_1 = ew^{-1} \pmod{q}$, $v_2 = rw^{-1} \pmod{q}$, $V = v_1G + v_2PU_s$, and $v = x_V \pmod{q}$ where x_V is the x coordinate of the point V. At last, it compares v and r to verfiy whether d_1 is authentic.

Chapter 3

Security of Digital Signature Schemes

The security of a digital signature scheme is dependent on key size, computational process, hash function used. We study the security of the digital signature schemes mentioned in Chapter 2.
In the context of GNSS
In mordern terms, a digital signature scheme consists of three algorithms:

1. Key Generation Algorithm which is probabilistic
2. Signing Algorithm which might be probabilistic
3. Verification Algorithm need not be probabilistic

Ideally, a digital signature scheme should be existentially unforgeable under chosen-message attack. The security of all known digital signature schemes depends on the intractability of certain computational problems in mathematics. Today's security proofs are reductions. The goal of such a reduction is to show that the ability of an attacker to mount a successful attack on a signature scheme implies his ability of solving a basic computational problem in mathematics namely:

1. Integer Factorization Schemes, which base their security on the intractability of the integer factorization problem. Examples include the RSA and Rabin Signature Schemes.
2. Discrete Logarithm Schemes, which base their security on the intractability of the (ordinary) discrete logarithm problem in a finite field. Examples include the ElGamal, Schnorr, DSA and Nyberg-Rueppel Signature Schemes.
3. Elliptic Curve Schemes, which base their security on the intractability of the elliptic curve discrete logarithm problem (ECDLP).

We consider some of the various approaches to evalutate the security of a cryptosystem.
The basis for the security of Elliptic Curve Cryptosystems such as the ECDSA is the apparent intractability of the following Elliptic Curve Discrete Logarithm Problem (ECDLP):
We now focus on the security of the Digital Signature Schemes mentioned in Chapter 2. the security of a key depends on its size and its algorithm. Some algorithms are easier to break than others and require larger keys for the same level of security. Breaking an RSA key requires you to factor a large number. We are pretty good at factoring large numbers and getting

better all the time. Breaking an ECDSA key requires you to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP). The mathematical community has not made any major progress in improving algorithms to solve this problem since is was independently introduced by Koblitz and Miller in 1985. This means that with ECDSA you can get the same level of security as RSA but with smaller keys. Smaller keys are better than larger keys for several reasons. Smaller keys have faster algorithms for generating signatures because the math involves smaller numbers.

3.1 Attacks on Digital Signature Schemes

Attacks against signature schemes can be classified according to the goals of the adversary. In the context of GNSS the following types of attacks can be considered in the order of increasing severity. Here A denotes the user whose signature method is being attacked, and E denotes the attacker.

1. **Key-only attacks**: In this attack the adversary knows only the real signer's public key i.e. E only knows A's public key. This is possible in the context of GNSS as the adversary has access to the public key which is known to all.

2. **Known-message attack**: The attacker is given access to signatures for a set of messages $m_1, m_2, \ldots, m_t$. The messages are known to the attacker but are not chosen by him. In other words, E is given access to a set of messages and their signatures. This is possible in the context of GNSS wherein the adversary has access to a large collection of messages.

3. **Generic chosen-message attack**: E chooses a list of messages before attempting to break A's signature scheme, independent of A's public key. E then obtains from A valid signatures for the chosen messages. The attack is generic because it doesn't depend on A's public key; the same attack is used against everyone.This is not feasible in the context of GNSS as the adversary doesn't have access to the satellite which is transmitting the signatures

4. **Directed chosen-message attack**: Similar to the generic attack, the only difference that the list of messages to be signed is chosen after E knows A's public key but before any signatures are seen. This is not feasible in the context of GNSS as the adversary doesn't have access to the satellite which is transmitting the signatures

5. **Adaptive chosen-message attack**: The attacker E is allowed to use A as an oracle which means that E may request signatures of messages that depend on previously obtained message-signature pairs. This is not feasible in the context of GNSS as the adversary doesn't have access to the satellite which is transmitting the signatures

Additionally, the attacker can have different goals:

1. **Total break**: The attacker, E determines A's private key.It is the most drastic attack.

2. **Universal forgery**: The attacker, E finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.

3. **Selective forgery**: Forge a signature for a particular message chosen a priori by the attacker i.e. the attacker, E forges a signature for a particular message chosen by E.

4. **Existential forgery**: The attacker, E forges a signature for atleast one message. E has no control over the message whose signature he obtains, so it may be random or nonsensical and consequently this type of forgery may only be a minor nuisance to the user A.

The breaks listed above are in the order of decreasing severity.

3.2 Security analysis of ElGamal Digital Signature Scheme

The ElGamal signature scheme can be broken when the discrete logarithms in ${\mathbb{Z}_p}^*$ can be computed. The prime p must therefore be chosen to be large enough and $p-1$ must contain atleast one large prime factor to disable the Pohlig and Hellman algorithm. We must also select the cyclic group and the system parameters in such a way that it is computationally infeasible to solve the respective discrete logarithm problem. This scheme based on the difficulty of computing discrete logarithms is existentially forgeable with a generic message attack and selectively forgeable using a directed chosen-message attack.
Note: Security of ElGamal Signatures

1. We elaborate on the requirement that the ElGamal Signature Generation algorithm must use a fresh and unique $t \in {\mathbb{Z}_p}^*$ for every digital signature it generates. If this requirement is not met, then it would be possible to retreive the signing key from only two valid digital signatures. Let s_1 and s_2 be two ElGamal signatures for messages m_1 and m_2 that are generated with the same t:

$$\begin{aligned} s_1 &= t^{-1}(H(m_1) - xr)) \pmod{(p-1)} \\ s_2 &= t^{-1}(H(m_2) - xr)) \pmod{(p-1)} \end{aligned}$$

   If t is the same, then $r = g^t \pmod{p}$ is also the same for both the signatures and as a consequence we have:

$$\begin{aligned} s_1 - s_2 &= (t^{-1}(H(m_1) - xr) - t^{-1}(H(m_2) - xr)) \pmod{(p-1)} \\ &= (t^{-1}H(m_1) - t^{-1}xr - t^{-1}H(m_2) - t^{-1}xr) \pmod{(p-1)} \\ &= (t^{-1}H(m_1) - t^{-1}H(m_2)) \pmod{(p-1)} \\ &= (t^{-1}(H(m_1) - H(m_2))) \pmod{(p-1)} \end{aligned}$$

   If $H(m_1) - H(m_2)$ is invertible $\pmod{(p-1)}$ then we can compute t. Furthermore, given t, s_1, r and $H(m_1)$, we can retreive the private key x. Note that $s_1 = (t^{-1}(H(m_1) - xr)) \pmod{(p-1)}$ and hence

$$x = (r^{-1}(H(m_1) - ts_1)) \pmod{(p-1)}$$

   This is unfortunate, and so we stress the requirement for a fresh and unique t that is chosen randomly from ${\mathbb{Z}_p}^*$ for every ElGamal signature that is generated.

2. If no hash function H is used, then the signing equation is . It is easy for an adversary to mount an existential forgery attack as follows:

3. The verifier is required to check that $0 < r < p$. If this check is not done, then an adversary can sign messages of its choice provided it has one valid signature created by entity A as follows.

4. An adversary might attempt to forge A's signature on m by selecting a random integer k and computing

Clearly, If an attacker could solve the discrete logarithm problem, he could break the scheme completely by computing the secret key x from the information in the public file. Moreover, if an attacker finds t for one message, he can solve the discrete logarithm problem, so the pseudo random number generator employed to generate ts has to be of superior quality.
The security of ElGamal signature scheme depends heavily on the the parameters p and g.

3.3 Security analysis of Digital Signature Algorithm

Unlike the Schnorr scheme, in DSA finding a collision in the hash function does result, via a known-message attack, in a break on the signature scheme. DSA therefore cannot profit from the reduced hash output and signature size that Schnorr enjoys. For DSA no security proof in the random oracle model is known. The security of the DSA is dependent on the hardness to find or compute the discrete logarithm problem of a very large number.

Speed Precomputations: Note that the value of r doesn't depend on the message. We can create a string of random t values, and then precompute r values for each of them or we can precompute t^{-1} for each of the t values so when a message m is generated, we can just compute s for a given r and t^{-1}. This speeds up DSA. []

3.4 Security analysis of Schnorr Digital Signature Scheme

The Schnorr signature scheme is known to be provably secure in the random oracel model under the discrete logarithm assumption i.e., for its provable security properties the famous forking lemma is used to prove the scehme secure under the hardness of computing discrete logarithms in the random oracel model. This important result guarentees that, as long as the hash function behaves ideally, the only way to break the Schnorr signatures is by solving the discrete logarithm problem. Note that we lack a proof in the standard model. Just like (EC)DSA signatures, Schnorr signatures also suffer from issues related to poor randomness in the ephemeral secret key. Thus, the attacks proposed for (EC)DSA signatures should also be applied to Schnorr signatures. [] the security of the Schnorr signature scheme, based on the assumption that the hash function H behaves like a random oracle, was long considered a folklore result. A formal treatment along with concrete estimates appeared in [33, 34].
Assume you have two signatures (s, e) and (s', e') for two messages m and m' such that $m \neq m'$. Using the same randomness would mean that $r = r'$ since $t = t'$. Since $m \neq m'$ we will have

that $s \neq s'$ (H is a secure cryptographic hash function and would need to collide). Now we know that $s = (t - xe) \pmod q$ and $s' = (t - xe') \pmod q$. Consequently, since t is identical for both signatures we have that : $s + xe = s' + xe' \pmod q \rightarrow s - s' = a(e' - e) \pmod q$ and thus $x = (s - s')(e' - e)^{-1} \pmod q$. Ignoring the very unlikely cases where $s - s' = 0$, $e' - e = 0$ and since $(e' - e)^{-1}$ always exists, the attacker can extract the private key x. So, when using the Schnorr's signature, it is essential, that the value of t is chosen independently and randomly for every signature such that these "randomness collisions" don't occur. All ElGamal type signatures (such as DSA/ECDSA) suffer from the same problem when re-using the random secret integer t.
Note: Performance characteristics of the Schnorr scheme
Signature generation in requires one exponentiation modulo p and one multiplication modulo q. The exponentiation modulo p could be a precomputation step. Depending on the hash algorithm used, the time to compute $H(mr)$ should be relatively small. Verification requires two exponentiations modulo p. Using the subgroup of order q does not significantly enhance computational efficiency over the ElGamal scheme, but does provide smaller signatures (for the same level of security) than those generated by the ElGamal method.

3.5 Security analysis of the Nyberg-Rueppel signature scheme

1. Since the Algorithm is a variant of the basic ElGamal scheme the security considerations of apply. Like DSA, this ElGamal scheme with message recovery relies on the difficulty of two related but distinct discrete logarithm problems.

2. Since the algorithm provides message recovery, a suitable redundancy function R is required to guard against existential forgery. The following possible attack should be kept in mind. Suppose $m \in \mathcal{M}$, $\widetilde{m} = R(m)$ and (e, s) is a signature for m.
   Then $e = \widetilde{m}g^{-t} \pmod p$ for some integer t and $s = ae + t \pmod q$. Let $\widetilde{m}^* = \widetilde{m}g^u$ for some integer u. If $s^* = (s + u) \pmod q$ and $\widetilde{m}^* \in \mathcal{M}_R$, then (e, s^*) is a valid signature for $m^* = R^{-1}\widetilde{m}^*$. To see this, consider the verification algorithm of the digital signature. $v = g^{s^*}y^{-e} = g^{s+u}g^{-ae} = g^{t+u} \pmod p$ and $ve = g^{t+u}\widetilde{m}g^t = \widetilde{m}g^u = \widetilde{m}^* \pmod p$. Since, $\widetilde{m}^* \in \mathcal{M}_R$, the forged signature (e, s^*) will be accepted as a valid signature for m^*.

3. The verification that $0 < e < p$ in the verification algorithm is crucial. Suppose (e, s) is A's signature for the message m. Then, $e = \widetilde{m}r \pmod p$ and $s = (ae + t) \pmod q$. An adversary can use this signature to compute a signature on a message m^* of its choice. It determines an e^* such that $e^* = \widetilde{m}^*r \pmod p$ and $e^* = e \pmod q$ which is possible by the Chinese Remainder Theorem (5.1.2.12). The pair (e^*, s) will pass the verification algorithm provided that $0 < e^* < p$ is not checked.

3.6 Security analysis of ECDSA

ECDSA signatures are randomized: each signature consists of two values (r, s): the value r is derived from an ephemeral public key tP generated using a random per-message secret t, and

a signature value s that depends on t. It is essential for the security of ECDSA that signers use unpredictable and distinct values for t for every signature, since predictable or repeated values allow an adversary to efficiently compute the long-term private key from one or two signature values.

In order to avoid the Pohlig-Hellman attack and Pollard's-ρ attack on the ECDLP, it is necessary that the order of $E(\mathbb{F}_q)$ be divisible by a sufficiently large prime n. At a minimum, one should have $n > 2^{160}$. Having fixed an underlying field $\mathbb{F}_q$, maximum resistance to the Pohlig-Hellman and Pollards rho attacks is attained by selecting E so that order of $E(\mathbb{F}_q)$ is prime or almost prime, that is, order of $E(\mathbb{F}_q) = hn$ where n is prime and h is small (e.g., h = 1,2,3 or 4).

A possible attack on the ECDSA:
The secret t used for signing two or more messages should be generated independent of each other. A different secret t should be used for siging different messages otherwise the private key x can be recovered. However, if a secure random or pseudo-random number generator is used, then the chance of generating a repeated t value is negligible. However, if same secret t is used to generate signature of two different messages m_1 and m_2 then it will result in two different signatures (r, s_1) and (r, s_2).
There are many variants of the ECDSA as listed in [21] and note that as the complexity increases the security level also increases.
Provable Security of ECDSA[22] Here are necessary security conditions for ECDSA:

1. The discrete logarithm in the subgroup spanned by G is hard

2. SHA-1 is a one-way hash function

3. SHA-1 is a collision-resistant hash function

4. The generator for t is unpredictable.

Implementation Considerations: To sign messages, ECDSA first applies a secure hash function to generate a digital fingerprint of the message, which is typically shorter than the message itself and then signs the fingerprint rather than the whole message. For proper implementation the following two conditions must be met:

1. The length of the signed navigation message must be at least as long as the output of the hash function

2. Each signed navigation message must vary in at least a single bit from previous messages to generate an unpredictable signature.

Implementation Considerations
Domain Parameters
Attacks

3.6.1 ECDSA and the Elliptic Curve Discrete Logarithm Problem

ECDSA is based on the hardness of the ECDLP. The signature scheme uses elliptic curves E over some prime field $\mathbb{F}_p$ where $|p| = m$ or elliptic curves over $\mathbb{F}_{2^m}$. In the first case, possible values for m are $\{112, 128, 160, 224, 256, 384, 521\}$ and in the second case $\{113, 131, 163, 193, 233, 283, 409, 571\}$. Once the curve E has been chosen, a base point G is chosen on E, which generates a subgroup of order q, such that $|E|$ i.e. the number of points in the curve and defining the co-factor h by $h = |E|/q$, inequality $h \leq 4$ holds.

3.7 Comparision of the Schemes above

ElGamal signatures work modulo a prime p and require one modular exponentiation (for generation) or two (for verification) with exponents as big as p; and the signature is two integers modulo p. This contrasts with DSA and Schnorr, which both work in a subgroup, traditionally a 160-bit subgroup for a 1024-bit modulus. DSA and Schnorr are 6 times faster than ElGamal, and produce signatures which are 6 times smaller. This difference in performances is enough to explain not choosing ElGamal. Indeed, the short size of DSA and Schnorr signatures has long been the selling point for these algorithms when compared to RSA.

3.7.1 Efficiency

ElGamal Digital Signature vs. DSA [5]:
The significant difference between the ElGamal signature scheme and the DSA is in the verification process. In the basic or generalized ElGamal scheme, we compute $r \in G$ as a group element and send r as a part of the signature. Thus we reveal the group element r. To verify the signature, we compute $y^{H(r)}r^s$ and $g^{H(m)}$ in the group G and check if they are same as group elements. But in the DSA, we take the group element $g^t \pmod p \in \mathbb{Z}_p^*$ and take the remainder modulo q. That is, $r = (g^t \pmod p) \pmod q$. Therefore, r is not an element of the group $\mathbb{Z}_p^*$. In the verification process, we compute $g^{u_1}y^{u_2} \pmod p = g^(s^1H(m) + s^1xr) \pmod p = g^t \pmod p)$ and take the remainder modulo q. Then we check if the outcome equals r. We compare the two elements $(g^{u_1}y^{u_2} \pmod p) \pmod q$ and r in $\mathbb{Z}/q\mathbb{Z}$ not in the group $\mathbb{Z}_p^*$. To summarize, in the basic ElGamal scheme,we need to compute $v_1 = y^r r^s \pmod p$ and $v_2 = g^{H(m)} \pmod p$ to verify the signature, which requires three modular exponentiations. But in the DSA we just need to compute $v = (g^{u_1}y^{u_2} \pmod p) \pmod q$ for verification, using only two modular exponentiations. DSA has a computational advantage in the verification process. One other difference is the signs in the signature s. In the basic ElGamal scheme $s = t^1(H(m)xH(r)) \pmod{(p1)}$ and in the DSA $s = t^1(H(m) + xr) \pmod q$. The sign appears as + in the DSA because of the modification in the verification process, but it is not essential.
textbfDSA vs. ECDSA:
The main difference between DSA and ECDSA is in the computation of r. In DSA, r is computed by selecting a random t and computing $g^t \pmod p$ and then reducing it modulo q. In ECDSA, we select a random t and compute the point tP. Then take the x-coordinate of the point tP and reduce it modulo n. As it was with DSA, the element (or point) $tP = (x_1, y_1)$ of the group $E(\mathbb{Z}_p)$ is not revealed. Here, $r = x_1 \pmod n$ is given as a part of the signature.

Since we do not know tP, the final verification is not done in $E(\mathbb{Z}_p)$ but in $\mathbb{Z}/n\mathbb{Z}$.

3.7.2 Allocation of Bandwidth in the Commercial Service of Galileo GNSS

Data/ Transmission Rate 500 bps. Suppose we transmite 400 bits of data and 100 bits of signature.
For the most commonly used 1024-bit keys for an integer based algorithm; the elliptic curve counterpart requires only 160-bit keys for the equivalent security. (Refer to Table 1.1). This is 7 times reduction in the space required to store these keys, or a similar reduction in bandwidth required to transmit these keys. This reduction in the size of data allows much faster completion of the algorithms.

Chapter 4

Conclusion

The underlying mathematical problem of a public-key cryptosystem determines the efficiency of the cryptosystem in a way. Because these problems dictate the sizes of domain parameters and keys, which in turn affect the performance of the arithmetic operations of the public-key crypto algorithms.The parameter sizes, generally called the key sizes are listed in the table below. The listing has been done according to equivalent security levels.The comparison shows that elliptic curve cryptography algorithm uses smaller parameter sizes than RSA and DSA for the same security levels. That brings the advantage of faster computations, smaller keys and certificates. The smaller domain parameter sizes also provide bandwidth savings. According to the key size, bandwidth requirements of ECC is said to provide greater efficiency than either integer factorization systems or discrete logarithm systems. This means that higher speeds, lower power consumptions and reduced code size are the advantages of ECC. Furthermore, the hardness of the underlying mathematical problem ECDLP appeals EC public-key cryptosystem for applications demanding high security.

4.1 Selecting the Appropriate Signature

With short signatures, efficient verification and standardization, ECDSA appears to be ——— for NMA-based commercial Galileo signal authentication.

4.2 Evaluating Digital Signature Protocols

The discussion of cryptographic signal authentication has settled on an NMA technique whereby a public-key digital signature is embedded in the navigation message. In this section we evaluate potential digital signature protocols that could generate the signed navigation message so that we have the most effective and pratical protocol for commercial Galileo GNSS signal authentication. Before comparing the various digital signature protocols for NMA, refined definitions of effectiveness and practicality with respect to digital signature procols is provided to guide the selection process:

1. Standardization: This indicates hat the protocol has been well-studied by the cryptography community and is thought to be secure against even the strongest cryptographic

attacks.

2. The equivalent symmetric-key strength b_s in units of bits is a useful measure of the strength of a cryptographic protocol. The U.S. National Institute of Standards and Technology (NIST) considers $b_s \geq 112$ secure for the years $2011 - 2030$ [**?**]. To meet NIST guidelines, a cryptographic GNSS signal authentication strategy must therefore set $b_s \geq 112$.

Thus, protocols that have short signature length for a given b_s and that have a low computational burden are deemed pratical for our purpose.

DSA: The Digital Signature Algorithm has two domain parameters that determine the strength of the algorithm. In order to achieve $b_s = 112$, it is neccessay to use a 2048-bit prime p and a 224-bit prime q. Verification of digital signature relies on p. DSA signatures are only twice as long as q (i.e., 448-bit signature for $b_s = 112$). Despite having short signature length, DSA is still not practical enough for cryptographic signal authentication because of its computational complexity. [**?**]

ECDSA: Based on, DSA the ECDSA operates on groups associated with an elliptic curve. For a given b_s, ECDSA signatures are the same length as DSA signatures. But by operating on a more complicated underlying elliptic curve space, ECDSA has smaller domain parameters and more efficient verification algorithms. [**?**]

Chapter 5

Appendix

5.1 Number Theory

Cryptographic algorithms are based on some very essential mathematical theory. We briefly list some of the theorems and algorithms used, in this chapter without proofs. The main references for this chapter are [25], [28] and [29].

5.1.1 Divisibility

Definition 5.1.1.1 *Suppose $a, b \in \mathbb{Z}$ with $a \neq 0$, we say that a divides b if there is an integer $c \in \mathbb{Z}$ such that $b = ac$. If a divides b, we say that a is a divisor or factor of b and that b is a multiple of a. If a divides b we write $a \mid b$ and if a does not divide b we write $a \nmid b$.*

Proposition 5.1.1.2 *If $a, b, c \in \mathbb{Z}$ with $a \mid b$ and $b \mid c$ then $a \mid c$.*

Proposition 5.1.1.3 *If a, b, m and $n \in \mathbb{Z}$ and if $c \mid a$ and $c \mid b$ then $c \mid (ma + nb)$.*

Theorem 5.1.1.4 ***(Division Algorithm)**: If $a, b \in \mathbb{Z}$ such that $b > 0$, then there are unique integers $q, r \in \mathbb{Z}$ such that $a = bq + r$ with $0 \leq r < b$. We call q the quotient, r the remainder, a the dividend and b the divisor.*

5.1.2 Congruences

Definition 5.1.2.1 *Suppose $a, b \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$. Then $a \equiv b \pmod{m}$ if m divides $b - a$. Note that the expression $a \equiv b \pmod{m}$ is called a 'congruence' and $m \in \mathbb{Z}^+$ is called the 'modulus'. For example: $4 \equiv 18 \pmod{7}$.*

Property 5.1.2.2 *Congruence modulo m is an equivalence relation i.e. for a fixed m the following three facts are true:*
(1) Reflexive: $a \equiv a \pmod{m}$
(2) Symmetric: $a \equiv b \pmod{m} \iff b \equiv a \pmod{m}$
(3) Transitive: If $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$, then $a \equiv c \pmod{m}$

Definition 5.1.2.3 *For fixed m, each equivalence class with respect to congruence modulo m has one and only one representative between 0 and $m-1$. The set of equivalence classes will be denoted by $\mathbb{Z}/m\mathbb{Z}$.*

Proposition 5.1.2.4 *The elements of $\mathbb{Z}/m\mathbb{Z}$ which have multiplicative inverses are those that are relatively prime to m.*

Proposition 5.1.2.5 *The congruence $ax \equiv b \pmod{m}$ has a unique solution $x \in \mathbb{Z}_m$ for every $b \in \mathbb{Z}_m \iff \gcd(a,m) = 1$*

Definition 5.1.2.6 *Let b, m and N be positive integers such that $b < m$. Then the least positive residue of $b^N \pmod{m}$ can be computed using $O((log_2 m)^2 log_2 N)$ bit operations.*

Definition 5.1.2.7 *Suppose $a \in \mathbb{Z}_m$. The multiplicative inverse of $a \pmod{m}$, denoted as $a^{-1} \pmod{m}$ is an element $b \in \mathbb{Z}_m$ such that $ab \equiv ba \equiv 1 \pmod{m}$.*

Definition 5.1.2.8 ***(Euler phi-function)**: Suppose $a \geq 1$ and $m \geq 2$ are integers. If $\gcd(a,m) = 1$, then a and m are said to be relatively prime. The number of integers in $\mathbb{Z}_m$ that are relatively prime to m is denoted by $\phi(m)$.*

Theorem 5.1.2.9 ***(Fermat's Little Theorem)**: Let p be a prime. Any integer a satisfies $a^p \equiv a \pmod{p}$, and any integer a not divisible by p satisfies $a^{p-1} \equiv 1 \pmod{p}$.*

Theorem 5.1.2.10 ***(Wilson's Theorem)**: If p is a prime, then $(p-1)! \equiv -1 \pmod{p}$.*

Remark 5.1.2.11 *The converse of the Wilson's Theorem is also true i.e. If n is a positive integer with $n \geq 2$ such that $(n-1)! \equiv -1 \pmod{n}$, then n is prime.*

Theorem 5.1.2.12 ***(Chinese Remainder Theorem)**: If the integers $n_1, n_2, \ldots, n_k$ are pairwise relatively prime, the the system of simultaneous congruences*

$$\begin{array}{rcll} x & \equiv & a_1 & \pmod{n_1} \\ x & \equiv & a_2 & \pmod{n_2} \\ \vdots & & & \\ x & \equiv & a_k & \pmod{n_k} \end{array} \tag{5.1}$$

has a unique solution modulo $n = n_1 n_2 \ldots n_k$

5.1.3 Finite Fields

We provide a brief description to finite fields.

Definition 5.1.3.1 *A finite field is a field $\mathbb{F}$ which contains a finite number of elements. The order of $\mathbb{F}$ is the number of elements in $\mathbb{F}$.*

Remark 5.1.3.2 ***(Existence and Uniqueness of Finite FIelds)**:*

1. *If $\mathbb{F}$ is a finite field, then $\mathbb{F}$ contains p^m elements for some prime p and integer $m \geq 1$.*
2. *For every prime power order p^m, there is a unique (up to isomorphism) finite field of order p^m. This field is denoted by $\mathbb{F}_{p^m}$*

Definition 5.1.3.3 *The non-zero elements of $\mathbb{F}_q$ form a group under multiplication called the multiplicative group of $\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$*

Remark 5.1.3.4 *$\mathbb{F}_q^*$ is a cyclic group of order $q-1$. Hence, $a^q = a$ for all $a \in \mathbb{F}_q$*

Definition 5.1.3.5 *A generator of the cyclic group $\mathbb{F}_q^*$ is called a primitive element or generator of $\mathbb{F}_q$.*

5.1.4 List of Algorithms

We shall introduce algorithms which will be of great use in our Digital Signature Schemes.

Algorithm 5.1.4.1 ***(Euclidean Algorithm)**:*
INPUT: two non-negative integers a and b with $a \geq b$
OUTPUT: the greatest common divisor of a and b.

1. *While $b \neq 0$ do the following:*
   (a) *Set $r \leftarrow a \pmod{b}$, $a \leftarrow b$, $b \leftarrow r$*
2. *return a*

Algorithm 5.1.4.2 ***(Extended Euclidean Algorithm)**:*
INPUT: two non-negative integers a and b with $a \geq b$
OUTPUT: $d = \gcd(a, b)$ and $x, y \in \mathbb{Z}$ satisfying $ax + by = d$.

1. *If $b = 0$ then set $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, and return (d, x, y).*
2. *Set $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$.*
3. *While $b > 0$ do the following:*
   (a) *$q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$.*
   (b) *$a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, and $y_1 \leftarrow y$.*
4. *Set $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, and return (d, x, y)*

Remark 5.1.4.3 *The Extended Euclidean Algorithm has a running time of $O((\log n)^2)$ bit operations.*

Algorithm 5.1.4.4 ***(Computing multiplicative inverses in $\mathbb{Z}_n$)**:*
INPUT: $a \in \mathbb{Z}_n$
OUTPUT: $a^{-1} \pmod{n}$, provided it exists

1. *Use the extended Euclidean Algorithm to find* $x, y \in \mathbb{Z}$ *such that* $ax + ny = d$ *where* $d = \gcd(a, n)$.

2. *If* $d > 1$, *then* $a^{-1} \pmod{n}$ *does not exist. Otherwise, return* x.

Algorithm 5.1.4.5 ***(Repeated square-and-multiply for exponentiation in*** $\mathbb{Z}_n$***)***:
INPUT: $a \in \mathbb{Z}_n$, *and integer* $0 \leq k < n$ *whose binary representation is* $k = \sum_{i=0}^{t} k_i 2^i$.
OUTPUT: $a^k \pmod{n}$

1. *Set* $b \leftarrow 1$. *If* $k = 0$ *then return* b.

2. *Set* $A \leftarrow a$

3. *If* $k_0 = 1$ *then set* $b \leftarrow a$.

4. *For* i *from* 1 *to* t *do the following:*

   (a) *Set* $A \leftarrow A^2 \pmod{n}$

   (b) *If* $k_i = 1$ *then set* $b \leftarrow A.b \pmod{n}$

5. *Return* b.

Algorithm 5.1.4.6 ***(Simultaneous multiple exponentiation)***:
INPUT: group elements $g_0, g_1 \ldots, g_{k-1}$ *and non-negative* t*-bit integers* $e_0, e_1, \ldots, e_{k-1}$.
OUTPUT: $g_0^{e_0} g_1^{e_1} \ldots g_{k-1}^{e_{k-1}}$

1. *Precomputation: For* i *from* 0 *to* $2^k - 1$ - $G_i \leftarrow \prod_{j=0}^{k-1} g_j^{i_j}$ *where* $i = (i_{k-1} \ldots i_0)_2$

2. $A \leftarrow 1$

3. *For* i *from* 1 *to* t *do the following:* $A \leftarrow A \times A, A \leftarrow A \times G_{I_i}$.

4. *Return* A.

Algorithm 5.1.4.7 ***(NIST method for generating DSA primes)***:
INPUT: an integer $\ell, 0 \leq \ell \leq 8$ *OUTPUT: a 160-bit prime* q *and an* L*-bit prime* p, *where* $L = 512 + 64\ell$ *and* $q|(p-1)$

1. *Compute* $L = 512 + 64\ell$. *Using long division of* $(L-1)$ *by 160, find n,b such that* $L - 1 = 160n + b$, *where* $0 \leq b \leq 160$.

2. *Repeat the following:*

   (a) *Choose a random seed* s *(not necessarily secret) of bit length* $g \geq 160$

   (b) *Compute* $U = H(s) \oplus H((s+1) \mod 2^g)$

   (c) *Form* q *from* U *by setting to 1 the most significant and least significant bits of* U. *(Note that* q *is a 160-bit odd integer).*

(d) *Test q for primality using MILLER-RABIN(q, t) for $t \geq 18$ Until q is found to be a probably prime.*

3. *Set $i \leftarrow 0$, $j \leftarrow 2$.*

4. *While $i < 4096$ do the following*

   (a) *For k from 0 to n do the following: Set*

5. *For i from 1 to t do the following: $A \leftarrow A \times A, A \leftarrow A \times G_{I_i}$.*

6. *Return A.*

5.1.5 Primes and Pseudoprimes

Definition 1 ***(Primality Decision Problem)****: Given a positive integer $n \in \mathbb{N}$ decide whether $n \in \boldsymbol{P}$ (i.e., n is prime) or not (i.e., n is composite)*

5.1.6 Primitive Roots and It's Applications

Primitive roots have many uses. For example, when an integer n has a primitive root, discrete logarithms of integers can be defined. Discrete logarithms can be used to simplify computations modulo n.

Definition 5.1.6.1 *The greatest integer in a real number x, denoted $[x]$, is the largest integer less than or equal to x i.e. $[x]$ is the integer satisfying $[x] \leq x < [x] + 1$. The greatest integer function is also known as the floor function denoted by $\lfloor x \rfloor$. The ceiling function of a real number x, denoted by $\lceil x \rceil$, is the smallest integer greater than or equal to x.*

5.2 Complexity Theory

The goal of complexity theory is to provide mechnisms by which computational problems may be classified in terms of the resources (time and storage space) required to solve them.
An algorithm is a computational procedure which takes a variable input and terminates with some output. If the algorithm follows the same execution path each time it is executed with the same input, then we say that the algorithm is deterministic. However, a randomized algorithms execution path differs each time its executed on the same input its decisions rely on a supply of random bits.
The running time of an algorithm on a particular input is the number of steps or primitive operations executed before the algorithm terminates.
The worst-case running time of an algorithm is an upper bound on the running time of an algorithm for any input. This is usually expressed as a function of the input size.
The expected running time of an algorithm is the average running time of an algorithm over all inputs of a specific size. This is usually expressed as a function of the input size.
Since the exact running time of an algorithm is often difficult to derive, we often rely on approximations of the running time.

5.2.1 Integer Representations and Operations

Theorem 5.2.1.1 *Let b be a positive integer with $b > 1$. Then every positive integer n can be written uniquely in the form $n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$ where k is a non-negative integer, a_j is an integer with $0 \leq a_j \leq b-1$ for $j = 0, 1, \ldots, k$ and the initial coefficient $a_k \neq 0$.*

Remark 5.2.1.2 *In the expansion defined above, b is called the base of the expansion. Base 2 expansions are called binary expansions. The coefficients a_j are called the digits of the expansion. Binary digits are called bits.*

5.2.2 Complexitiy of Integer Operations

Computers internally represent numbers using bits. Computers have a built-in limit on the size of integers that can be used in machine arithmetic. This upper limit is called the word size and is usually a power of 2. For details on how to perform addition, subtraction, multiplication and division of two n-digit integers please refer to [29].

Computational Complexity of an algorithm is the number of bit operations needed to perform an algorithm. In describing the number of bit operations needed to perform calculations, we will use big-O noation, which provides an upper bound on the size of a function in terms of a particular well-known reference whose size at large values is easily understood.

Definition 5.2.2.1 *Let f and g be real-valued functions. We are only concerned with the behaviour of $f(x)$ and $g(x)$ as $x \to \infty$. Further assume that $g(x) > 0$ for sufficiently large x. Then, $f = O(g)$ means that $|f(x)| \leq cg(x)$ for some positive constant c and sufficiently large x.*

Definition 5.2.2.2 *Let $a \in \mathbb{Z}$. We define it's bit length denoted by $len(a)$ to be the number of bits in the binary representation of $|a|$. Thus if $len(a) = l$ then we say that a is an l-bit integer.*

Definition 5.2.2.3 ***(Polynomial Time Algorithm)**: An algorithm is called a polynomial-time algorithm if its worst-case running time function is polynomial in the input size. Any algorithm whose running time cannot be bounded by a polynomial is called super-polynomial time. Consequently, the worst-case running time of a polynomial-time algorithm is of the form $O(n^c)$ where n is the input size and c is the constant.*

Definition 5.2.2.4 ***(Subexponential Time Algorithm)**: An algorithm is subexponential time if its worst-case running time function is of the form $e^o(n)$, where n is the input size.*

Remark 5.2.2.5 *In general, we regard polynomial time algorithms as being efficient and non-polynomial time algorithms as being inefficient. We call a problem that cannot be solved in polynomial time intractable or infeasible.*

Please find below the table of Bit Complexity of basic operations in Z_n (table 2.5) Handbook of Applied Cryptography.

5.2.3 Efficient Computations

In practice, one is often interested in finding the most efficient (i.e., fastest) algorithm to compute a function or solve a problem. Consequently, the notion of efficiency is closely related to the running time of an algorithm. The running time of an algorithm on a particular input, in turn, can be defined as the number of primitive operations or steps that must be executed. Often a step is taken to mean a bit operation. For some algorithms, however, it is more convenient to have a step mean something more comprehensive, such as a comparison, a machine instruction, a machine clock cycle, a modular multiplication, or anything else along these lines. In either case, the running time of an algorithm can be measured in the worst or average case:

1. Theworst-caserunningtimeofanalgorithmisanupperboundontherunning time for any input, expressed as a function of the input size.

2. The average-case running time of an algorithm is the average running time over all inputs of a fixed size, also expressed as a function of the input size.

In modern cryptography, it is common to call a computation efficient if it terminates within a worst-case running time that is polynomial in the input size.

Definition 5.2.3.1 ***(Polynomial Function)**: A function $f(n)$ is called a polynomial if $f(n) = O(n^c)$ for some $c \in \mathbb{N}$. Otherwise, it is called super-polynomial.*

Bibliography

[1] Anon., http://www.esa.int/Our_Activities/Navigation/The_future_-_Galileo/What_is_Galileo

[2] Anon., *Galileo Space Segment* http://www.navipedia.net/index.php/Galileo_Space_Segment

[3] Anon., http://www.gsc-europa.eu/galileo-overview/why-europe-needs-galileo

[4] D. Johnson, A. Menezes, *Elliptic Curve DSA (ECDSA): An Enhanced DSA*, Proceedings of the 7th conference on USENIX Security Symposium, Vol. 7, 1998.

[5] A. Khalique, K. Singh, S. Sood, *Implementation of Elliptic Curve Digital Signature Algorithm*, International Journal of Computer Applications, Volume 2 - No. 2, May 2010.

[6] B. Wellenhof, H. Lichtenegger, E. Wasle, *GNSS - Global Navigation Satellite Systems: GPS, GLONASS, Galileo and more*, SpringerWienNewYork, 2008.

[7] A. Jovanovic, C. Botteron, P. Farine *Multi-test Detection and Protection Algorithm Against Spoofing Attacks on GNSS Receivers*, Position, Location and Navigation Symposium - PLANS 2014, pages 1258-1271, IEEE/ION, 2014.

[8] O. Pozzobon, *Keeping the Spoofs Out: Signal Authentication Services for Future GNSS*, InsideGNSS, May/June 2011.

[9] I. Hein, F. Kenissl, J. Rodriguez, S. Wallner, *Authenticating GNSS: Proofs against Spoofs Part 1*, InsideGNSS, July/August 2007.

[10] G. Becker, S. Lo, D. Lorenzo, P. Enge, C. Paar, *Secure Location Verification: A security Analysis of GPS Signal Authentication*, Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy, pages 366-373, 2010

[11] G. Becker, S. Lo,D. Lorenzo,D. Qiu,C. Paar,P. Enge, *Efficient Authentication Mechanisms for Navigation Systems - a Radio-Navigation Case Study*, Proceedings of the 22nd International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS), 2009.

[12] I. Hein, F. Kenissl, J. Rodriguez, S. Wallner, *Authenticating GNSS: Proofs against Spoofs Part 2*, InsideGNSS, September/October 2007.

[13] O. Pozzobon, C. Sarto, A. Chiara, A. Pozzobon, G. Gamba, M. Crisci, R. Ioannides. *GNSS Vulnerability and Mitigation Techniques: Developing a GNSS Position and Timing Authentication Testbed*, InsideGNSS, January/February 2013.

[14] S. Lo, D. Lorenzo, P. Enge, D. Akos, P. Bradley, *Signal Authentication: A Secure Civil GNSS for Today*, InsideGNSS, Pages 30-39, September/October 2009.

[15] I. Hernandez, I. Rodriguez, G. Tobias, J. Calle, E. Carbonell, G. Seco-Granados, J. Blasi, *Galileo's Commercial Service: Testing GNSS High Accuracy and Authentication*, InsideGNSS, January/February 2015.

[16] P. Papadimitratos, A. Jovanovic, *Protection and Fundamental Vulnerability of GNSS*, International Workshop on Satellite and Space Communications (IWSSC), IEEE Xplore, 2008.

[17] P. Papadimitratos, A. Jovanovic, *GNSS-based Positioning: Attacks and Countermeasures*, Military Communications Conference (MILCOM), IEEE Xplore, 2008.

[18] National Institute of Standards and Technology, *Recommended elliptic curves for federal government use*, `http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf`, 1999.

[19] C. Wullems, O. Pozzobon, K. Kubik, *Signal authentication and integrity schemes for next generation global navigation satellite systems*, In Proceedings of the European Navigation Conference GNSS, 2005, Munich, Germany.

[20] T. Adamski, W. Nowakowski, *Security of Nyberg-Rueppel digital signatures without message recovery*, Bulletin of the Polish Academy of Sciences, Vol. 62, No. 4, 2014.

[21] G. Sarath, D. Jinwala, S. Patel, *A Survey on Elliptic Curve Digital Signature Algorithm and It's Variants*, International Journal of Computer Science and Information Technology, Vol. 6, Issue 3, page 121, 2014.

[22] S. Vaudenay, *The Security of DSA and ECDSA: Bypassing the Standard Elliptic Curve Certification Scheme*, Lecture Notes in Computer Science, Vol. 2567, pages 309-323, Dec 2002.

[23] Anon., *The Galileo Commercial Service: Current Status and Prospects*, `http://mycoordinates.org/the-galileo-commercial-service-current-status-and-prospects/`

[24] Anon., *Recommendation for key managementPart I: General (revised)*, National Institute of Standards and Technology, SP 800-57, March 2007.

[25] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, second edition, 1994.

[26] D. Hankerson, A. Menezes, S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag,2004.

[27] L. Washington, *Elliptic Curves Number Theory and Cryptography*, Taylor and Francis Group, second edition, 2008.

[28] D. Stinson, *Cryptography: Theory and Practice*, CRC Press, third edition, 2006.

[29] K. Rosen, *Elementary Number Theory and Its Applications*, Pearson, fifth edition, 2005.

[30] A. Menezes, P. Oorschot, S. Vanstone *Handbook of Applied Cryptography*, CRC Press, 1996.