Hierarchical Access Control, Threshold Progressive Proxy Re-Encryption and Polynomial Root Finding

Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY

by

Reetwik Das
[Roll No. 14CSE1021]
Soham Tamba
[Roll No. 14CSE1027]

Under the guidance of
Dr. Purushothama B. R.
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY GOA

MAY-2018

Department of Computer Science and Engineering
National Institute of Technology Goa
Farmagudi, Goa, India-403401

CERTIFICATE

This is to certify that the work contained in this report entitled **"Hierarchical Access Control, Threshold Progressive Proxy Re-Encryption and Polynomial Root Finding"** is submitted by the group members Mr. Reetwik Das (Roll. No: 14CSE1021) and Mr. Soham Tamba (Roll. No: 14CSE1027) to the Department of Computer Science and Engineering, National Institute of Technology Goa, for the partial fulfillment of the requirements for the degree of **Bachelor of Technology** in **Computer Science and Engineering**.

They have carried out their work under my supervision. This work has not been submitted else-where for the award of any other degree or diploma.

The project work in our opinion, has reached the standard fulfilling of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering in accordance with the regulations of the Institute.

Dr. Purushothama B.R.
Supervisor
Department of CSE
NIT Goa

Dr. Damodar Reddy Edla
Head of Department
Department of CSE
NIT Goa

Acknowledgement

Its our privilege to thank all the people who contributed to making the commencement of this project a reality. It was a great chance for learning and educational development. Therefore, we consider ourselves as very lucky individuals as we were provided with this opportunity to work in the given field of study.
We would like to use this opportunity to express our deepest gratitude and special thanks to **Dr. Damodar Reddy Edla**, the Head of Computer Science and Engineering Department, National Institute of Technology Goa for the efforts taken to provide us with this opportunity and providing necessary guidance.
We would like to thank and extend our heartfelt gratitude to our project mentor and guide **Dr. Purushothama B.R.** for his encouragement to take up this challenging project and for his guidance and providing invaluable inputs that kept us on the correct path.
Finally, we would like to thank everyone who had a vital role in this commencement of making our visualized concepts a dream come true.

Abstract

Transforming the ciphertext intended for Alice into a ciphertext intended for Bob provides a solution to the problem of delegation without sharing Alice's secret key. In hierarchical networks this allows every node to have access to the messages intended for all of its children nodes. If the delegatee is multiple hops away, many re-encryptions are required to obtain the desired ciphertext, which is computationally expensive. We propose a solution that solves this problem by computing the re-encryption key between the two security classes using the intermediate re-encryption keys and generate the required ciphertext in a single re-encryption. The scheme is proved secure against any collusions between the classes involved.

The network scenarios where a group of proxies need to perform progressive re-encryptions in order to generate a valid ciphertext for a delegate is susceptible to the problem of single point of failure. Therefore a novel proxy re-encryption primitive called Threshold Progressive Proxy Re-encryption is proposed which involves progressive transformation of ciphertext and results in production of a valid re-encrypted ciphertext even if at least a certain number t of distinct proxies perform re-encryption. The security of the scheme against collusion amongst the intermediate proxies and the delegatee is also proved. This scheme scores over lagrangian polynomial interpolation method by eliminating the need of pre-requisite information about the available proxies and central dealer.

Many applications in computer science can be reduced to computing the roots of a polynomial. As the degree and coefficients of the polynomial grow, root finders that scale well in terms of number of bit operations performed are required. We propose two such algorithms that have an upper bound for the number of bit operations performed in terms of the input only making no assumptions about the roots of the polynomial.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Proxy re-encryption allows to transform the ciphertext intended for Alice into ciphertext that can be read using Bob's secret key without sharing Alice's secret key with Bob. This provided a solution to the problem of delegation without the need of trusting the delegatee with the sender's secret. This concept of proxy re-encryption can be extended to achieve hierarchical access control over a network where every node is a default delegatee to all its children nodes. In such a network scenario a unidirectional, transitive proxy re-encryption scheme is required. In this report we propose a scheme satisfying the above properties and compare its performance against the other approaches to achieve hierarchical access control. In a multi hop scenario where the delegatee is many hops away from the actual receiver our scheme computes the re-encryption key between the two parties from the published re-encryption keys and then uses that to perform the necessary transformation of ciphertext. The security of the scheme against the collusion between various security classes involved is also discussed.

There are numerous scenarios where the re-encryption keys between the nodes are not publicly available. This could be done to ensure that the ciphertext was forwarded by all the intermediate proxies involved. In such cases it is necessary for all the proxies to progressively re-encrypt the ciphertext for the delegatee to be able decrypt it. Such network scenarios have an issue of single point of failure where a single failed proxy could disrupt the working of the entire re-encryption process. Hence there arises a need of fault tolerant scheme which allow some proxies to be unavailable and still complete the progressive re-encryption of ciphertext properly. We propose a (t,n) threshold scheme which achieves the same even with the availability of t out of the total n proxies involved. There is no involvement of a central coordinator in the process which makes the scheme better than lagrangian polynomial interpolation and moreover any knowledge about the availability of proxies is not required. The scheme is proved to be secure against collusion amongst the involved proxies and the delegatee.

The above problem is just one of the applications of polynomials in Computer Science. Many problems can be reduced into computing the real roots of a polynomial. There exist multiple algorithms that can find roots of the given polynomial but as the degree and coefficients of the polynomial grow, we require root finders that scale well in terms of number of bit operations performed. A Deterministic as well as a Randomized algorithm is proposed that solve this problem and have an upper bound for the number of bit operations performed in terms of the input only. It is notable that the algorithms make no assumptions whatsoever about the roots of the polynomial and hence can be applied to find the roots of any univariate polynomial.

The rest of the report is organized as follows. Chapter 2 introduces Hierarchical access control using proxy re-encryption. The construction, performance, analysis and security of the proposed scheme is also discussed in chapter 2. Then Chapter 3 deals with the process of achieving fault tolerance in dynamic topology networks with the use of Threshold Progressive Proxy Re-Encryption scheme. The system model, its construction, security and limitations are also addressed in it. Chapter 4 includes algorithms to find the roots of a univariate polynomial. It contains both a randomized and a deterministic algorithm for the same, their correctness and comparison with other models. Chapter 5 contains the conclusion and future work of the report.

Chapter 2

Hierarchical Access Control

Proxy re-encryption allows a proxy to transform a ciphertext computed under Alice's public key into one that can be read by Bob's secret key. For instance, Alice might wish to temporarily forward encrypted email to her colleague Bob, without sharing her secret key with him. In this case, Alice the delegator could designate a proxy to re-encrypt her incoming mail into a format that Bob the delegatee can decrypt using his own secret key. Alice could simply provide her secret key to the proxy, but this requires an unrealistic level of trust in the proxy.

There exists several efficient proxy re-encryption schemes that offer security improvements over this. The primary requirements of the schemes is that they should be unidirectional (i.e., Alice can delegate to Bob without Bob having to delegate back to her) and should not require delegators to reveal all of their secret key to anyone or even interact with the delegatee in order to allow a proxy to re-encrypt their ciphertexts.

In these schemes, only a limited amount of trust is placed in the proxy. For example, it is not able to decrypt the ciphertexts it re-encrypts and the scheme should be secure even when the proxy publishes all the re-encryption information it has. This enables a number of applications that would not be practical if the proxy needed to be fully trusted.

Another application of the proxy re-encryptions is that it is also applicable on a hierarchical structure i.e. any node has the ability to decrypt the ciphers intended for any of its children if all the re-encryption keys are available. This would unlock a variety of applications that would be possible using proxy. Our main focus is to obtain mechanisms that can ensure the hierarchical proxy re-encryptions are done more efficiently.

2.1 Applications of Proxy Re-Encryption

Depending upon the properties that are satisfied by the proxy re-encryption scheme they can be used in a variety of applications.

2.1.1 Secure File Systems

A secure file system is a natural application of proxy re-encryption because the system often assumes a model of untrusted storage. Confidentiality is obtained by encrypting the contents of stored files. These encrypted files can then be stored on untrusted file servers. The server operators can distribute encrypted files without having access to the plaintext files themselves.

In a single-user cryptographic file system, access control is straightforward. The user creates and remembers all the keys protecting content. Thus, there is no key distribution problem. With group sharing in cryptographic storage, group members must rendezvous with content owners to obtain decryption keys for accessing files.

In the case of a multi-user cryptographic file system a trusted access control system is used to distribute keys. Proxy cryptography can be used to reduce the amount of trust in the access control server. Here keys protecting files are stored encrypted under a master public key. When a user requests a key, the access control server uses proxy re-encryption to directly convert the key to the user without learning the it in the process.

The master secret key can be stored offline, by a content owner who uses it only to generate the re-encryption keys used by the access control server.

2.1.2 Outsourced Filtering of Encrypted Spam

Another promising application of proxy re-encryption is the filtering of encrypted emails performed by authorized contractors. The volume of unsolicited email, along with rapid advances in filter-avoidance techniques, has overwhelmed the filtering capability of many small businesses, leading to a potential market for outsourced email filtering.

By accepting encrypted email from outside sources, institutions become “spam targets and filters are only effective on messages that are first decrypted (which could be unacceptably costly). Using proxy re-encryption, it becomes possible to redirect incoming encrypted email to an external filtering contractor at the initial mail gateway, without risking exposure of plaintexts at the gateway itself.

2.1.3 Hierarchical access control

In a hierarchical access control scheme, the users are partitioned into classes. The hierarchy of the classes is represented by a partially ordered set. If $A \prec B$ then class B can decrypt the message encrypted for class A. It is desirable that the amount of secret storage required for each class to maintain this access control is independent of the number of classes in the hierarchy.

Such a hierarchical access control scheme would be desired in an organization such as the military where an individual must be able to monitor the emails received by their subordinates.

It has been illustrated in [2] that the above problem can be addressed using a proxy re-encryption scheme. In a proxy re-encryption scheme, a user B is delegated the right to decrypt A's cipher-text should be able to decrypt A's message without the active

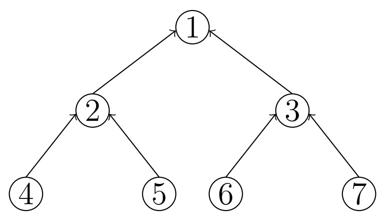

Figure 2.1: Hierarchical model of nodes

participation of A. A proxy server obtains a re-encryption $rk_{A \to B}$ and uses it to convert a cipher-text encrypted for A into a cipher-text encrypted for B. The proxy should not be able to obtain the secret key of A or B, nor should it be able to decrypt their messages.

2.2 Properties of Proxy Re-Encryption

Here is a list of some useful properties of proxy re-encryption schemes. These properties are used to compare known proxy re-encryption schemes.

(i) **Unidirectional**: Delegation from $A \to B$ does not require delegation from $B \to A$.

(ii) **Non-interactive**: Re-encryption keys can be generated by Alice using Bob's public key and no interaction is required.

(iii) **Proxy invisibility**: The proxy in scheme should be transparent in the sense that neither the sender of an encrypted message nor any of the delegatees have to be aware of the existence of the proxy. Any delegatee will not be able to distinguish a first-level encryption (computed under his public key) from a re-encryption of a ciphertext intended for another party.

(iv) **Original access**: Alice can decrypt re-encrypted ciphertexts that were originally sent to her.

(v) **Key optimal**: The size of Bob's secret storage remains constant, regardless of how many delegations he accepts. This is an important consideration, since the safeguarding and management of secret keys is often difficult in practice.

(vi) **Collusion "safe"**: The property of collusion "safeness" is extremely useful in this context, since we allow the sender to generate first-level encryptions, that can be opened only by the intended recipient (Alice), or second-level ones that can be opened by any of the recipients delegatees (e.g., Bob). Indeed, this property implies that even if Bob and the proxy collude, they will not be able to open any of Alice's first level-encryptions.

(vii) **Temporary**: In the mentioned scheme Bob should only be able to decrypt messages intended for Alice that were authored during some specific time period.

(viii) **Non Transitive**: The proxy, alone, cannot redelegate decryption rights. For example, from $rk_{a\to b}$ and $rk_{b\to c}$ he cannot produce $rk_{a\to c}$.

(ix) **Non Transferable**: The proxy and a set of colluding delegatees cannot redelegate decryption rights. For example, from $rk_{a\to b}$,sk_b and pk_c, they cannot produce $rk_{a\to c}$.

2.2.1 Imitated Transitivity

Imitated transitivity means given the $($secret key of D, $rk_{A\to B}$, $rk_{B\to C}$, $rk_{C\to D})$ we can compute $rk_{A\to D}$.
Transitivity means given $(rk_{A\to B}, rk_{B\to C}, rk_{C\to D})$ we can compute $rk_{A\to D}$.
The problem with this approach is that only D can compute $rk_{A\to D}$ and will not have the option of delegating that work to some other entity.
The process of decrypting A's messages by D consists of the 3 phases.

(i) Obtain $rk_{A\to D}$.

(ii) Re-encrypt cipher-text C_A to cipher-text C_D.

(iii) Decrypt C_D using D's secret key

In the ideal transitive re-encryption key scheme, Ds secret key is not needed to obtain $A \rightarrow D$, D can delegate the first 2 phases to some other entity and then decrypt the messages by itself. This is not possible when transitivity is "imitated" as D's secret key is required for the first phase. This involves encrypting a private information and storing it in public domain.

2.2.2 Unidirectional Proxy Re-encryption

A unidirectional proxy re-encryption scheme is a tuple of polynomial time algorithms $(KG, RG, \vec{E}, R, \vec{D})$, where the components are defined as follows:

- $(KG, \vec{E}, \vec{D})$ are the standard key generation, encryption, and decryption algorithms for the underlying cryptosystem. Here $\vec{E}$ and $\vec{D}$ are (possibly singleton) sets of algorithms. On input the security parameter 1^k, KG outputs a key pair (pk, sk). On input pk_A and message $m \in M$, $\forall E_i \in \vec{E}$ the output is a ciphertext C_A . On input sk_A and ciphertext C_A , $\exists$ a $D_i \in \vec{D}$ that outputs the message $m \in M$.
- On input $(pk_A, sk_A^{\dagger}, pk_B, sk_B^{*})$, the re-encryption key generation algorithm, RG, outputs a key $rk_{A \rightarrow B}$ for the proxy. The fourth input marked with a $*$ is sometimes omitted; when this occurs we say that RG is *noninteractive*, since the delegatee doesn't need to be involved in the generation of the re-encryption keys. The second input marked with a † may, in some cases, be replaced by the tuple $(rk_{A \rightarrow C}, sk_C)$.
- On input $rk_{A \rightarrow B}$ and ciphertext C_A , the re-encryption function, R, outputs the re-encrypted ciphertext C_B.

2.3 Notation

- Re-encryption scheme: Unless stated otherwise, a re-encryption scheme will refer to a **uni-directional**, **collusion safe** and **key optimal** [1] re-encryption scheme.
- $rk_{A \to B}$: Re-encryption key used to re-encrypt a message for A to a message for B.
- $desc(A)$: Set of classes C such that $C_i \preceq A$.
- $anc(A)$: Set of classes C such that $A \preceq C_i$.
- $ak_{A \to C}$: Assigned public key of the pair of classes, (A, C)

2.4 Literature Survey

Shaohua et al. [3] introduced a solution based on Linear Algebra that use proxy re-encryption keys. In this scheme, for every pair $(A,\ B)$ such that A can read B's cipher-text, a re-encryption key is published. This scheme requires little space overhead per key, but a quadratic number of keys are required and it requires a huge number of key updates when an update in the topology of the hierarchy is done. Also the participation of the classes is required for these updates which may be unacceptable in this model. We classify this scheme as exhaustive.

Chen et al. [4] proposed an RSA based re-encryption scheme. This scheme enables the re-encryption of RSA cipher-texts. It requires far fewer key update operations than [3] after a modification to the hierarchy. However, the number of operations required to re-encrypt the cipher-text increases with the height of the hierarchy. In a large hierarchy where messages are frequently sent, this may be un-scaleable. We classify this scheme as multi-hop.

Depending on the application of a proxy re-encryption scheme, a set of properties are desirable [1]. We revise some of the properties relevant to this application below:

- $Unidirectoinal$: If B can decrypt $A's$ messages then A cannot decrypt $B's$ messages.
- $Transitive$: Given $\{rk_{A \to B}, rk_{B \to C}\}$, it should be easy to obtain $rk_{A \to C}$.
- $Collusion\ safe$: If $\forall\ C_i \in S = \cup\ \{C_j\}_{j=1}^{n}$, $C_i \notin anc(A)$ then it should not be possible for S to read the decrypt of A by pooling their secret keys.
- $Key\ optimal$: The secret storage of B should be constant, regardless of the number of delegations it has accepted.
- $Proxy\ invisibility$: It is not possible to differentiate between the cipher-text re-encrypted for A and A's cipher-text. Hence, the cipher-text can be re-encrypted multiple times.

There are three methods of using proxy re-encryption schemes to achieve hierarchical access control:-

(i) Uni-directional Transitive re-encryption scheme

For every edge $A \to B$ in the hierarchy DAG, publish a transitive re-encryption key $rk_{A \to B}$.

(ii) Multi-hop re-encryption scheme

For every edge $A \to B$ in the hierarchy DAG, publish a proxy invisible re-encryption key $rk_{A \to B}$.

(iii) Exhaustive re-encryption scheme

For every pair A, C such that $A \prec C$, publish a re-encryption key $rk_{A \to C}$.

2.5 Proposed Proxy Re-encryption based Hierarchical access control Scheme

A brief overview of the scheme is described as follows:

For every edge $A \to B$ in the hierarchy DAG, publish $rk_{A\to B}$ which is a function of the secret keys and assigned keys of A and B.

For every class A in the scheme, the proxy will store an assigned key.

For every pair A, C such that $A \prec C$, an assigned public key $(ak_{A\to C})$ will be published so that C can use $\cup\{rk_{c_i\to c_{i+1}}\}_{j=1}^{n} \cup \{ak_{c_i\to C}\}_{j=1}^{n}$ to derive $rk_{A\to C}$. This can be considered a combination of 1 and 3.

2.5.1 Preliminaries and Notations for construction

Here we introduce the preliminaries required and the notation used in the contruction process of the scheme.

2.5.1.1 Bilinear Map

We say a map e: $G_1 \times \hat{G_1} \to G_2$ is a bilinear map if:

(i) $G_1, \hat{G_1}$ are groups of same prime order q.

(ii) $\forall a, b \in Z_q, g \in G_1$, and $h \in \hat{G_1}$, then $e(g^a, h^b) = e(g,h)^{ab}$ is efficiently computable.

(iii) The map is nondegenerate (i.e., if g generates G_1 and h generates G_1 , then $e(g,h)$ generates G_2).

2.5.1.2 Notations

- g, Y, Z are the generators of a group.
- e_0 : billinear maps such that $e_0(g^a, g^b) \; = \; Y^{ab} \, \forall \; a, \; b$.
- e_1 : billinear map such that $e_1(Y^a, Y^b) \; = \; Z^{ab} \, \forall \; a, \; b$.

2.5.2 Construction

We present a construction for the proposed proxy re-encryption based hierarchical access control scheme.

Set up

- *Secret Key* of class A: a.
- *Public Key* of class A: Z^a, $g^{\frac{1}{a2_{lock}}}$.
- *Assigned Keys* of class A: k_A, a_{lock}, $a2_{lock}$.
- *Assigned Public Key* of pair A, B: $g^{\frac{k_A}{k_B} \times \frac{1}{a_{lock}}} \; g^{\frac{k_B}{k_A} \times a2_{lock}}$.
- *Re-encryption Key*:
  In our scheme we will have two types of re-encryption keys.
  - *Level 1 re-encryption key*: $pr_{A \to B} \; = \; g^{(\frac{k_B}{k_A} b - a) a_{lock}}$.
    These keys will be used to produce level 2 re-encryption keys.
  - *Level 2 re-encryption key*: $rk_{A \to X} \; = \; Y^{(x - a \frac{k_A}{k_X})}$.
    These keys will be used re-encrypt the messages of A.
- For *transitivity*, we will show that it is possible to obtain $rk_{A \to X}$ using $pr_{A \to B}$ and $rk_{B \to X}$.

Encryption

Consider the case where any user P wishes to send a message m to A. It will perform the following steps.

(i) Obtain $Z^a,\ g^{\frac{1}{a2_{lock}}}$ from public domain.

(ii) Randomly compute s.

(iii) Compute $g^{\frac{s}{a2_{lock}}}$ using $g^{\frac{1}{a2_{lock}}}$ and s.

(iv) Compute Z^s .

(v) Compute $Z^{s\times a}$.

(vi) Send $(m \times Z^{s\times a},\ Z^s,\ g^{\frac{s}{a2_{lock}}})$.

Decryption

Consider the case where A wants to decrypt a cipher-text received by it.
Secret Key of node A: a, .

(i) Receive $(m \times Z^{s\times a},\ Z^s,\ g^{\frac{s}{a2_{lock}}})$.

(ii) Compute $Z^{a\times s}$ from Z^s and a.

(iii) Compute m = $(m \times Z^{s\times a})/Z^{s\times a}$

Re-encryption

Consider the case where X wants to re-encrypt A's cipher-text.
Re-encryption key: $rk_{A\rightarrow X} \ = \ Y^{(x-a\frac{k_A}{k_X})}$.

(i) Receive $(m \times Z^{s\times a},\ Z^{s},\ g^{\frac{s}{\overline{a2_{lock}}}})$.

(ii) Compute $Y^{\frac{k_X}{k_A}s} = e_0(g^{\frac{s}{\overline{a2_{lock}}}},\ g^{\frac{k_X}{k_A}a2_{lock}})$.

(iii) Compute $Z^{\frac{k_X}{k_A}s}$ using $Y^{\frac{k_X}{k_A}s}$.

(iv) Compute $Z^{(x\frac{k_X}{k_A}-a)s} = e_1(Y^{\frac{k_X}{k_A}s},\ Y^{(x-a\frac{k_A}{k_X})})$

(v) Output $(Z^{(x\frac{k_X}{k_A}-a)s},\ Z^{\frac{k_X}{k_A}s},\ m \times Z^{s\times a})$.

Decrypting a re-encrypted message

Consider the case where X wants to decrypt A's message that has been re-encrypted using $rk_{A\to X}$. *Secret Key* of node X: x, .

(i) Receive re-encrypted message $(Z^{(x\frac{k_X}{k_A}-a)s},\ Z^{\frac{k_X}{k_A}s},\ m \times Z^{s\times a})$.

(ii) Compute $Z^{a\times s}$ using $Z^{(x\frac{k_X}{k_A}-a)s}$, $Z^{\frac{k_X}{k_A}s}$, x, .

(iii) Compute $m = (m \times Z^{a\times s})/Z^{a\times s}$

2.5.3 Discussion

Here we discuss some important features and running time analysis of the scheme.

2.5.3.1 Transitivity

Given
$pr_{A\to B} = g^{(\frac{k_B}{k_A}b-a)a_{lock}}$ and $rk_{B\to X} = Y^{(x-b\frac{k_B}{k_X})}$ we can obtain
$rk_{A\to X} = Y^{(x-a\frac{k_A}{k_X})}$.

(i) Obtain $g^{\frac{k_A}{k_X}\times\overline{a_{lock}}}$ from the public domain.

(ii) Compute $Y^{(\frac{k_B}{k_X}b-\frac{k_A}{k_X}a)} = e_0(g^{(\frac{k_B}{k_A}b-a)a_{lock}},\ g^{\frac{k_A}{k_X}\times\frac{}{a_{lock}}})$.

(iii) Compute $Y^{(x-a\frac{k_A}{k_X})} = Y^{(x-b\frac{k_B}{k_X})} \times Y^{(\frac{k_B}{k_X}b-\frac{k_A}{k_X}a)}$

2.5.3.2 Distributing the load of updates

We note that the amount of computation required by the proxy server is quite large for updates and suggest the following change to make the computation more suitable to run in a distributed environment by minimizing data movement:

If $A \prec B$ then publish $ak_{A\to B}$.
else publish $ak^{t_A}_{A\to B}$ where t_A is the secret key of the proxy used for encryption.
When the assigned public keys of A need to be updates, it is sufficient to update the values of a_{lock} and $a2_{lock}$.

Suppose we want to change the value of a_{lock} to $a' \times a_{lock}$ after an edge deletion. Raise $g^{(\frac{k_X}{k_A}x-a)a_{lock}}$ to power a' and $g^{\frac{k_A}{k_B}\times\frac{1}{a_{lock}}}$ to power $\frac{1}{a'}$.
Do the same for $a2_{lock}$.
Publishing $ak_{A\to B}$ is equivalent to decrypting it with t_A.
When B is no longer an ancestor of A, Instead of destroying $ak_{A\to B}$, perform the update stated above, encrypt it using t_A and then publish the result.
Note that the update of $a2_{lock}$ and a_{lock} must be performed on the published data as well as the encrypted data.
Since $g^{(a\times t_A)\times a'} = g^{(a\times a')\times t_A}$ updating an encrypted key is same as updating a published key.

2.5.3.3 Runtime Analysis

We will analyze the amount of time required to perform the operations in section 5 using a distributed environment where each class is assigned an independent.

- t_{pair}: Time required for a bilinear mapping.

- t_{exp}: Time required for modular exponentiation.
- t_{random}: Time required to generate a random integer to perform updates.
- t_{mul}: Time required for modular multiplication.

Consider the case where there is a path from A to B of length n in the hierarchy and A receives a message. The run-time for B to perform the following operations is:

(i) Generate key using transitivity: $(n + 1) \times t_{pair} + n \times t_{mul}$

(ii) Re-encrypt message: $3 \times t_{pair}$

(iii) Decrypt re-encrypted message: $2 \times t_{exp}$ + t_{mul}

Consider the case where edge $B \rightarrow C$ is inserted or deleted and A is a descendant of B. The runtime of the independent computer assigned to A for the following operations is:

(i) Edge insertion: $2 \times anc(B) \times t_{exp}$

(ii) Edge deletion: $2 \times anc(A) \times t_{exp}$ + t_{random}

2.6 Comparision and Analysis of Schemes

In this section, we will map the requirements of Hierarchical access control scheme to the proxy re-encryption schemes described above. We will analyze the total computation and computation by the classes required to perform the following operations:

(i) Re-encrypt message

(ii) Edge Deletion

(iii) Edge Insertion

(iv) Class Insertion

(v) Class Deletion

2.6.1 Re-encrypt message

Consider the case where m cipher-texts have been sent to A and there is a path $P = \cup \{C_j\}_{j=1}^{n+1}$ of length n from A to C in the hierarchy DAG.

(i) $Uni\text{-}directional\ Transitive$: Use $\cup\{rk_{c_i \to c_{i+1}}\}_{j=1}^{n}$ to derive $rk_{A \to C}$. After the re-encryption key has been derived using transitivity, m re-encryption are required.

(ii) $Multi\text{-}hop\ re\text{-}encryption$: Each of the m message received by A, C will re-encrypt the message n times using $\cup\{rk_{c_i \to c_{i+1}}\}_{j=1}^{n}$.

(iii) $Exhaustive\ re\text{-}encryption$: Use $rk_{A \to C}$ to directly re-encrypt each of the m message.

(iv) $Our\ scheme$: Use $\cup\{rk_{c_i \to c_{i+1}}\}_{j=1}^{n} \cup \{ak_{c_i \to C}\}_{j=1}^{n}$ to derive $rk_{A \to C}$. Then re-encrypt each message directly with m re-encryption operations.

Note that in 1 and 4, $rk_{A \to C}$ can be saved in public storage for some duration, reducing the number of computations from $m + n$ to m.

2.6.2 Edge Deletion

Consider the case where edge $A \to B$ must be deleted. We have to ensure that an ancestor of B that is no longer allowed to read the messages of a descendant of A.

(i) $Uni\text{-}directional\ Transitive$: Replace the secret keys of every class in $desc(A)$ and the corresponding re-encryption keys.

(ii) $Multi\text{-}hop\ re\text{-}encryption$: Replace the secret keys of every class in $desc(A)$ and the corresponding re-encryption keys. Otherwise, B can use the old re-encryption keys to re-encrypt a cipher text intended for a descendant of A.

(iii) $Exhaustive\ re\text{-}encryption$: Replace the secret keys of every class in $desc(A)$ and the corresponding re-encryption keys. There will be $\sum_{k \in desc(A)} |anc(k)|$ such re-encryption keys.

(iv) $Our\ scheme$: For every class in $desc(A)$, change their assigned key and the corresponding assigned public keys. There will be $\sum_{k \in desc(A)} |anc(k)|$ assigned public keys.

Class deletion is equivalent to removing all the edges entering and leaving a node.

2.6.3 Edge Insertion

Consider the case where edge $A \to B$ must be inserted.

(i) $Uni\text{-}directional\ Transitive$ Publish $rk_{A \to B}$

(ii) $Multi\text{-}hop\ re\text{-}encryption$: Publish $rk_{A \to B}$

(iii) $Exhaustive\ re\text{-}encryption$: For every pair ($C$, D) in $desc(A) \times anc(B)$, publish a $rk_{C \to D}$.

(iv) $Our\ scheme$: For every pair (C, D) in $desc(A) \times anc(B)$, publish $ak_{C \to D}$.

Class insertion is equivalent to adding edges to an isolated class. For our scheme, the proxy must generate assigned keys during a class insertion.

It is clear from table[1] that in terms of computations required, a transitive re-encryption scheme is ideal.

Unfortunately, [2] proved that an proxy re-encryption scheme that is key optimal, collusion safe, unidirectional and transitive does not exist.

Scheme	Re-encryption	Edge Insertion	Edge Deletion
Transitive	$n+m$	1	$\|desc(A)\|$
Multi-hop	$n \times m$	1	$\|desc(A)\|$
Exhaustive	m	$\|desc(A)\| \times \|anc(B)\|$	$\sum_{k \in desc(A)} \|anc(k)\|$
Our scheme	$n+m$	$\|desc(A)\| \times \|anc(B)\|$	$\sum_{k \in desc(A)} \|anc(k)\|$

Table 2.1: Order of upper bound on operations required.

Scheme	Re-encryption	Edge Insertion	Edge Deletion
Transitive	m	1	$\|desc(A)\|$
Multi-hop	m	1	$\|desc(A)\|$
Exhaustive	m	$\|desc(A)\| \times \|anc(B)\|$	$\sum_{k \in desc(A)} \|anc(k)\|$
Our scheme	m	1	0

Table 2.2: Order of upper bound on operations requiring class participation

It is clear from Table 2 that although our scheme requires more computation than other schemes, it requires much less participation from the classes and hence, is more suitable for our model where classes have very few computational resources.

2.7 Summary

We observed the applications of proxy re-encryption based hierarchical access control of nodes and various problems that limit its usage in practical scenarios. Our proposed scheme solves the problem of transitive, anti-symmetric proxy but requires quadratic number of public keys. It is also required to pre-compute the public keys which shouldn't be published. These are few short-coming of this scheme which should be handled in the future. It however has many advantages over other schemes namely that it is truly transitive i.e secret key is not required to perform re-encryption or for key construction. It is easy to run the scheme on a distributed environment to significantly speed up the key construction process.

Chapter 3

Threshold Progressive Proxy Re-Encryption

Proxy re-encryption (PRE) is the process of transforming a ciphertext intended for Alice into a ciphertext that can be read by using Bob's secret key. This scheme is useful as it eliminates the need for Alice to share her secret key with Bob, which requires an unrealistic level of trust, to access her encrypted messages [1]. Here, the proxy between Alice and Bob holds a re-encryption key that produces the necessary transformations on the ciphertext. The proxy, despite carrying out the ciphertext transformation, does not learn anything about the underlying plaintext or the secret keys of Alice or Bob.

Consider a scenario where instead of a single re-encrypting proxy, there is a pool of N proxies $\{P_1, P_2, \ldots, P_N\}$ and re-encryption is possible only if at least t of these proxies progressively re-encrypt a ciphertext. For the rest of the paper, we refer to re-encryption by each individual proxy P_i as partial re-encryption and the resulting ciphertext as partial re-encrypted ciphertext. Output of the t^{th} partial re-encryption is referred to as final re-encryption and resulting ciphertext as final re-encrypted ciphertext. Suppose Alice's ciphertext C_A arrives at the re-encryption facility. Any proxy P_i can choose to partially re-encrypt C_A and broadcast the result of its partial re-encryption to all the proxies. Any other proxy, say P_j can now choose to re-encrypt the partial re-encrypted ciphertext produced by P_i and broadcast the resulting partial re-encryption again. This process of partial re-encryption of the most recent partial re-encryption continues till at least t of the total number of proxies partially re-encrypt C_A. Following t^{th} partial re-encryption, the final re-encrypted ciphertext can be either sent to Bob for decryption or any other proxies may further try to partial re-encrypt the final re-encrypted ciphertext. While in the former case, Bob can successfully decrypt the ciphertext using its secret key, the latter case must result in production of a ciphertext that is no different from the final re-encrypted ciphertext. This way, output of any further partial re-encryptions after t^{th} partial re-encryption must still produce a valid final re-encrypted ciphertext intended for Bob. Since such a re-encryption technique satisfies threshold requirement and is progressive in nature, we call it as threshold progressive proxy re-encryption (TP-PRE).

At first glance, TP-PRE seems no different from a (t, N) threshold cryptography technique [6] being applied to proxy re-encryption scenario. However, there is a significant difference between the two. In existing threshold cryptosystems, recovering a secret requires interpolation over the respective partial secrets of at least t out of N participants. Shares of all $\geq t$ participants must be available beforehand for interpolation. All the participants are given copies of the same ciphertext which they independently re-encrypt to produce their respective partial re-encryption shares. These partial re-encryptions are then combined to re-construct the final re-encrypted ciphertext. In traditional threshold proxy re-encryption, the information about the available proxies must be available before this so-called re-construction phase.

The approach for threshold progressive re-encryption is different in that no output of the participating proxies need to be available in advance. Instead, there is a single copy of the ciphertext which any participating proxy can re-encrypt to produce a partial re-encryption. Any other proxy that wishes to participate takes this partial re-encryption as input to produce a new partial re-encryption and overwrite the previous partial re-encryption with new one. This process goes on and the ciphertext is successively re-encrypted by participating proxies in any order. A valid re-encrypted ciphertext is produced if the number of participating proxies or the number of times the ciphertext is progressively re-encrypted equals or exceeds the threshold t. Because of progressive re-encryption, the set of participating proxies need not be determined in order to collect shares and interpolate. Indeed, with the progressive approach, any t proxies can participate in a flexible manner and generate the final re-encrypted ciphertext. Formally, consider a set $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ of proxies in the system. Let $\mathcal{P}(t)$ be the set whose each element is a subset of $\mathcal{P}$ of size t or more. That is, $\mathcal{P}(t) = \{\mathcal{P}^i(t) : \mathcal{P}^i(t) \subseteq \mathcal{P}, |\mathcal{P}^i(t)| \geq t\}$. Threshold proxy re-encryption determines the set of re-encrypting proxies, that is some $\mathcal{P}^i(t) \in \mathcal{P}(t)$, in the beginning itself and uses the corresponding shares of the proxies in $\mathcal{P}^i(t)$ to re-construct the final re-encrypted ciphertext. Whereas, in a threshold progressive proxy re-encryption, the proxies in $\mathcal{P}$ progressively re-encrypt the ciphertext and the output is a valid final re-encryption only if the proxies that progressively re-encrypted the ciphertext form a set $\mathcal{P}^i(t) \in \mathcal{P}(t)$. Thus, unlike threshold proxy re-encryption, TP-PRE supports partial re-encryptions to be done by a set of any t proxies without the knowledge about participating proxies or their respective shares.

This has several advantages over the existing re-encryption systems. Firstly, the threshold nature of the TP-PRE scheme leads to better fault tolerance because any $(N - t)$ re-encrypting proxies are allowed to fail or not carry out re-encryption. This is a practical consideration given semi-trusted nature of the proxies. Secondly, no need of prior knowledge about the availability of the proxies leads to better flexibility. Any proxy can choose to re-encrypt the most recent partial re-encryption based on its idle time. Thirdly, this model fits in application scenarios where approval from multiple authorities is required for carrying out any critical activity. Suppose access to any critical message is to be granted to any delegatee only if at least t approval authorities vote for it. The scenario discussed above directly maps to situations like these because the approval authorities (here proxies), may not be available all the time. Therefore, any approval authority may get online and approve the request by re-encrypting the most recent partial re-encryption. Of many potential real-world applications of TP-PRE is in military operations where multiple troops on different war fronts are connected through a wireless sensor network. Any command of attack that is to be executed should be properly coordinated in order to have higher probability of success. In such situations if a command to attack is given from the base station to another station some troops might not be alive to convey the message forwarded or may not be in a condition to do so. Hence it is important that the command is executed if and only if appropriate number of troops say t out of N are alive and capable of executing the attack.

With the motivation and potential applications discussed above, we instantiate a new primitive called threshold progressive proxy re-encryption (TP-PRE). In this report, we formalize the definition and security notions of a TP-PRE scheme. We also provide a construction of a TP-PRE scheme that satisfies all the functional requirements discussed above and prove its security. Performance analysis is also presented by implementing the proposed TP-PRE scheme. We also discuss future of TP-PRE.

Our contributions

Having highlighted multiple practically motivated scenarios, we list our contributions in this report as follows.

(i) We formalize a novel proxy re-encryption primitive for progressive transformation of ciphertext that generates a valid re-encryption only if a certain number out of the total number of proxies participate in the progressive re-encryption procedure. We name the primitive Threshold Progressive Proxy Re-encryption (TP-PRE) define its security and performance requirements.

(ii) The proposed TP-PRE achieves the threshold requirement in a unique progressive way. Each proxy broadcasts its partial re-encrypted ciphertext, which any of the remaining proxies can use for producing further partial re-encryptions. This way, all $\geq t$ required re-encryptions are done progressively on the same ciphertext. No partial re-encryptions can be decrypted by anyone and its only t^{th} partial re-encryption and all those after t^{th} partial re-encryption that generate the final re-encrypted ciphertext that can be decrypted by the delegatee.

(iii) We present a novel construction for the proposed TP-PRE, prove its security (standard security and master-secret security) and establish its performance bounds in terms of the threshold t required and storage overhead on the collaborating proxies.

3.1 Literature Survey

Ever since its first instantiation by Blaze et al. [9] and further refinement by Ivan et al. [10] and Ateniese et al. [1], proxy re-encryption (PRE) has found application in a range of scenarios due to its simplicity and efficiency. Secure data forwarding in network of entities is one such application. We review some of the recent relevant works that target the single point of failure in traditional proxy re-encryption setting. Bulk of focus of such schemes has been on key management, especially re-keying in dynamic groups. Hur et al. [11] proposed a group re-keying technique where Elgamal based PRE is used for transmitting renewed group keys for different subgroups in the system. The scheme however, suffers from collusion problem. Arguably, if any of the two controllers of subgroups collude, they can obtain direct transitive re-encryption keys which is undesirable. A decentralized re-keying algorithm for group key management in near-space network was proposed by Wang et al. [8]. Their scheme for group key management uses a chain of re-encryptions carried out by a sequence of proxies using their respective re-encryption keys. A single proxy is replaced by a sequence of proxies so as to avoid the problem of single point of failure in the re-key process of the group key management scheme. Though the trust is distributed to many proxies in the network, even if a single proxy fails in their scheme, the chain is broken and valid re-encryption cannot be produced. Another related set of proxy cryptography primitives includes threshold proxy re-signatures and threshold proxy re-encryption. A proxy re-signature scheme enables a semi-trusted signing proxy to convert Aclice's signature into Bob's signature without learning signing secrets of either Alice or Bob. A threshold re-signature scheme involves generation of valid re-signature only if at least a certain number t of say N proxies partially re-sign a message. The major issue with the threshold PRE and threshold proxy re-signatures is that all the collaborating proxies have to submit their shares prior to generation of valid final re-signature and re-encryption, respectively. This is equivalent to having knowledge of all the available proxies for collaboration in advance. Noack et al. [12] proposed a threshold proxy re-signature scheme that does not require a manager to combine the partial re-signatures. The signature shares can be combined by any semi-trusted entity including one of the available proxies. However, their scheme requires the re-signature shares from each available proxies to be available at the time combine stage. Yang et al. [13] proposed a similar threshold re-signature scheme except that it is based on the standard model assumption. A flexible proxy re-signature scheme based on Chinese Remainder Theorem was proposed by Yang et al. [14] that supports a variable threshold for re-signing documents with variable sensitivity. Fournaris et al. [15] proposed a scheme that does not require any trusted dealer to combine the re-signature shares to obtain the final re-signature. A threshold PRE was proposed by Lin et al. [16] and applied in secure erasure encoding leads to a valid decryption of a re-encrypted ciphertext only if there are at least t valid decryption shares available. Their scheme does not address the problem of generating valid re-encrypted ciphertext only upon collaboration of at least t out of N proxies.

Another related PRE primitive is multi-hop proxy re-encryption. A multi-hop PRE scheme supports repeated re-encryption of a ciphertext C_A intended for a delegator A using a sequence of re-encryption keys say $rk_{A\to B}, rk_{B\to C}, rk_{C\to D}$ produce the final re-encrypted ciphertext C_D intended for delegatee D. Since this process involves repeated re-encryption, each ciphertext namely C_B and C_C produced by $rk_{A\to B}$ and $rk_{B\to C}$, respectively can be decrypted by parties B and C. Also, for progressive re-encrypting C_A for party D, a fixed sequence of re-encryption keys is required along with a fixed number of re-encryptions. Whereas in the scenario considered in this paper requires selection of proxies in any sequence and any number of proxies $\geq t$ can progressively re-encrypt the ciphertext to produce the final re-encrypted ciphertext.

3.1.1 BIBD (Balanced Incomplete Block Design)

Definition : A (n, d, λ) balanced incomplete block design is design (X, A) where X is the set of all the points and A is the collection of non-empty subsets of X. The following properties are satisfied in BIBDs [5]:

(i) $|X| = n$

(ii) each block contains exactly d points, and

(iii) each pair of distinct points are contained in exactly λ blocks.

The presence of property 3 makes the design "balanced" and since $d < n$ i.e. number of points held at each block is lesser than the total number of available points, it is called an incomplete design.

3.1.1.1 Useful properties of BIBDs

- **Repetition of each point :**
  In a (n, d, λ) BIBD every point occurs in exactly:
  $r = \lambda(n-1)/(d-1)$ blocks.
- **Total number of blocks :**
  A (n, d, λ) BIBD has exactly :
  $b = rn/d = \lambda(n^2 - n)/(d^2 - d)$ blocks.

3.1.1.2 Customized BIBD required

In BIBD that we specifically require in our application has $\lambda = 1$ which means every pair of distinct points will be repated in exactly 1 block. Also the number of blocks should be equal to the number of proxies i.e n so that every proxy has exactly 1 block. Therefore we have :

$n \ = (n^2 - n)/(d^2 - d)$
$\implies \ (n-1) \ = d(d-1)$
The repetition of each point is given by:
$r \ = (n-1)/(d-1)$
on substitution of the above equation we get:
$r \ = d.$

3.2 Threshold Progressive Proxy Re-encryption (TP-PRE) Scheme

A formal description of the TP-PRE scheme is presented in this section along with security notions. System model consists of description of various entities involved in TP-PRE along with key functionality. Construction syntax is the description of procedures involved in TP-PRE.

3.2.1 System Model

Fig. 3.1 shows the system model of the TP-PRE scheme. Direct ciphertext C_A under public key pk_A is intended for user U_A and can be decrypted using secret key sk_A of U_A. There exists a leader proxy P_0 that executes the pre-transformation phase to obtain pre-transformed ciphertext. This pre-transformation, when progressively re-encrypted by $\geq t$ proxies from the pool of proxies, leads to the generation of final re-encrypted ciphertext $C_B'^{(0)}$ that can be decrypted with secret key sk_B of U_B using decryption procedure **Dec$_2$**.

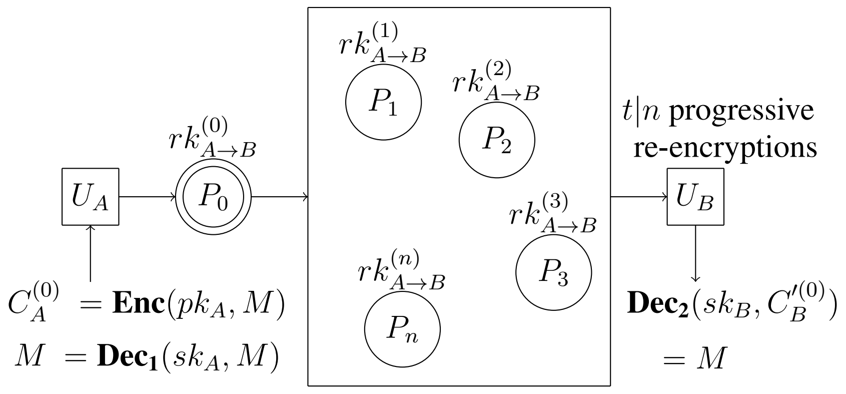
$rk^{(1)}_{A\to B}$
P_1
$rk^{(2)}_{A\to B}$
P_2
$t|n$ progressive re-encryptions
$rk^{(0)}_{A\to B}$
U_A
P_0
U_B
$rk^{(3)}_{A\to B}$
P_3
$rk^{(n)}_{A\to B}$
P_n
$C^{(0)}_A = \textbf{Enc}(pk_A, M)$
$M = \textbf{Dec}_\textbf{1}(sk_A, M)$
$\textbf{Dec}_\textbf{2}(sk_B, C'^{(0)}_B) = M$

Figure 3.1: TP-PRE system

3.2.2 Construction

(i) **Initialize**$(1^\lambda) \to params, MSK$: The global system parameters generated as output of this procedure include $params = (p, \mathbb{G}_1, \mathbb{G}_2, g, e, H, MSK)$. Here, p is a large prime number which is the order of the cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$, g is the generator of the group $\mathbb{G}_1$, $e : \mathbb{G}_1 \times \mathbb{G} \to \mathbb{G}_2$ is a bilinear mapping and $H : \mathbb{G}_2 \times \mathbb{G}_2 \to \mathbb{Z}_p$ is a collision resistant hash function. The collection MSK of master secrets $\{k_1, k_2, \dots, k_n\}$ is generated such that $\forall k_i \in MSK, k_i \in_R \mathbb{Z}_p$ is chosen uniformly at random from $\mathbb{Z}_p$. These secrets contained in MSK are not revealed to the users in the system.

(ii) **KeyGen**$(i, params) \to (pk_A, sk_A)$: The secret key sk_A and public key pk_A of each user U_A is computed as:

- Choose a random $x_A \in_R \mathbb{Z}_p^*$
- Set $sk_A = x_A$ and compute $pk_A = g^{sk_A}$.

(iii) **Enc**$(pk_A, M)\to C^{(0)}_A$: Direct ciphertext $C^{(0)}_A$ of any message $M \in \mathbb{G}_2$ under pk_A is computed as a 4-tuple (C_1, C_2, C_3, C_4) using the following steps:

- Choose a random $R \in \mathbb{G}_2$ and compute $C_1 = M \oplus R$

- Compute $r = H(R, M)$ and set $C_2 = R.e(g, g)^r, C_3 = (pk_A)^r = g^{sk_A \cdot r}, C_4 = 0$

Output the direct encryption $C_A^{(0)} = (C_1, C_2, C_3, C_4)$

(iv) **Dec₁**$(sk_A, C_A^{(0)}) \rightarrow M$: Direct encryptions under pk_A can be decrypted using sk_A as:

- Compute $R = C_2.e(C_3, g)^{-1/sk_A}$
- Compute $M = C_1 \oplus R$
- Output M if $(pk_A)^{H(R,M)} = C_3$. Otherwise, $\perp$.

(v) **ReKeyGen**$(sk_A, pk_A) \rightarrow rk_{A \rightarrow B}$: Re-encryption key $rk_{A\rightarrow B}^{(0)}$ of the leader proxy and $rk_{A\rightarrow B}^{(1)}, rk_{A\rightarrow B}^{(2)}, \ldots, rk_{A\rightarrow B}^{(n)}$ of the intermediate proxies for progressively contributing towards transformation of direct ciphertext $C_A^{(0)}$ into re-encrypted ciphertext $C_B'^{(0)}$, are computed as follows:

- Compute:

$$rk_{A\rightarrow B}^{(0)} = (pk_B)^{\frac{\prod_{q=1}^{n} k_q}{sk_A}}$$

- Compute $\forall a \in \{1, \ldots, n\}, rk_{A\rightarrow B}^{(a)}$ as:
  - Compute $\rho = \prod_{q=1}^{n} k_q$
  - $\forall q = 1, 2....n$ compute $x_q = \rho / k_q$ and y_q such that $x_q \times y_q \equiv 1 \mod k_q$
  - Compute $var_q = x_q \times y_q$ and $\mu = \sum_{q=1}^{n} var_q$
  - Compute:

    $rk_{A\rightarrow B}^{(a)} = \{(k_{a_1}, var_{a_1}), \ldots, (k_{a_d}, var_{a_d})\}$
    where d is the number of keys out of $\{k_1, \ldots, k_n\}$ stored at each proxy P_a. This number d is same for each proxy P_a.

(vi) **ReEnc**($rk_{A\to B}, C_A^{(0)}$)$\to C_B'^{(0)}$: On input $C_i^{(0)} = (C_1, C_2, C_3, C_4)$, a re-encrypted ciphertext $C_B'^{(0)} = (\overline{C_1}, \overline{C_2}, \overline{C_3}, \overline{C_4}, \overline{C_5})$ is produced using the steps given below:

- Set $\widetilde{C_1} = C_1, \widetilde{C_2} = C_2, \widetilde{C_3} = C_3$.
- Compute $\widetilde{C_4} = e(C_3, rk_{A\to B}^{(0)})$ and $\widetilde{C_5} = \mu$
- Output of the leader proxy: $C_B'^{(t)} = (\widetilde{C_1}, \widetilde{C_2}, \widetilde{C_3}, \widetilde{C_4}, \widetilde{C_5})$.
- Any intermediate proxy P_a can transform ciphertext $C_B'^{(t)}$ and the subsequent ciphertexts resulting after progressive re-encryptions by intermediate proxies using the steps given below:

  For each key k_{a_i} stored at a proxy holding keys $\{k_{a_1}, k_{a_2}, \ldots, k_{a_d}\}$, do the following:

  (a) Check if $\widetilde{C_5} \equiv 0 \mod k_{a_i}$.

  (b) If not, then compute $\widetilde{C_4} = \widetilde{C_4}^{1/k_{a_i}}$ and $\widetilde{C_5} = \widetilde{C_5} - var_{a_i}$ and select the next $i \in \{1, 2, \ldots, d\}$ and go to Step a).

  (c) Otherwise, select the next $i \in \{1, 2, \ldots, d\}$ and go to Step a).
- After $\widetilde{C_5} = \widetilde{C_5} - var_{a_i}$ is executed $\widetilde{C_5} \equiv 0 \mod k_{a_i}$ using the concept of chinese remainder theorem. This is used to check whether k_{a_i} has been used for re-encryption by any of the proxies.
- After at least t proxies transform $C_B'^{(t)}$ one-after the other in any order, $\widetilde{C_4} = Z^{d\times k}$ $\widetilde{C_5} = 0$ and the final output ciphertext is produced as:
  $C_B'^{(0)} = (\overline{C_1} = \widetilde{C_1}, \overline{C_2} = \widetilde{C_2}, \overline{C_3} = \widetilde{C_3}, \overline{C_4} = \widetilde{C_4}, \overline{C_5} = \widetilde{C_5})$.

(vii) **Dec₂**($sk_A, C_A'^{(0)}$) $\to M$: Decryption of the final re-encrypted ciphertext to produce the underlying plaintext message comprises of the following steps:

- Compute $R = \overline{C_2}.(\overline{C_4})^{-1/sk_B}$
- Compute $M = \overline{C_1} \oplus R$
- Output M, the plaintext message if $(pk_A)^{H(R,M)} = \overline{C_3}$. Otherwise, output $\perp$.

3.2.3 Correctness analysis

Correctness analysis of procedures **Enc** and **ReEnc** of the proposed TP-PRE scheme is presented. For the **Enc** to be correct, the following equality must hold:

$$\mathbf{Dec_1}(sk_A, \mathbf{Enc}(pk_A, M)) = M$$

The output of the **Enc** of TP-PRE scheme has output of the form:

$$C_1 = M \oplus R, C_2 = R.e(g,g)^r, C_3 = g^{sk_A r}, C_4 = 0$$

Dec₁ of the TP-PRE scheme computes:
$C_2.e(C_3, g)^{-1/sk_A} = R.e(g,g)^r \,.\, e(g^{sk_A r}, g)^{-1/sk_A}$
$= R.e(g,g)^r.e(g,g)^{-r}$
$= R.$
In the above, $r = H(R, M)$. The procedure **Dec₁** now computes $C_1 \oplus R = M$ to obtain the underlying plaintext message M. To verify integrity of the ciphertext received and sender's identity, **Dec₁** checks if $(pk_A)^{H(R,M)} \overset{?}{=} C_3$. $C_3 = g^r$ where $r = H(R, M)$. Thus, the verification step of procedure **Enc** is correct.

For the procedure **ReEnc** to be correct, correctness of the output $C_B'^{(0)}$ as well as all the intermediate re-encryptions $C_B'^{(p)} : \forall p \in \{1, \ldots, t\}$ must hold. That is,

$$\mathbf{Dec_2}(sk_B, \mathbf{ReEnc}(\mathbf{ReKeyGen}(sk_A, pk_B), C_A^{(0)})) = M$$

Output of **ReKeyGen** is the collection $\{rk^{(0)}_{A\to B}, rk^{(1)}_{A\to B}, rk^{(2)}_{A\to B}, \ldots, rk^{(n)}_{A\to B}\}$ of re-encryption keys assigned to the leader proxy for initial transformation and each of the intermediate proxies for progressive transformations. As mentioned in the construction, if t out of these n intermediate proxies transforms the ciphertext progressively one after the other, the final re-encrypted ciphertext $C_B'^{(0)}$ is produced. For leader proxy,

$$rk^{(0)}_{A\to B} = (pk_B)^{\frac{\prod_{p=1}^{n} k_p}{sk_A}}$$

$$= (g)^{\frac{sk_B \cdot \prod_{p=1}^{n} k_p}{sk_A}}$$

So, the output of the initial re-encryption by the leader proxy is given as:
$\widetilde{C_1} = M \oplus R, \widetilde{C_2} = R.e(g,g)^r, \widetilde{C_3} = g^{sk_A r}$,

$$\widetilde{C_4} = e(g^{sk_A r}, (g)^{\frac{sk_B \cdot \prod_{p=1}^{n} k_p}{sk_A}})$$
$$= e(g,g)^{sk_B . r . \prod_{p=1}^{n} k_p}$$

and $\widetilde{C_5} = \mu = var_1 + var_2 + \ldots + var_n$
Re-encryption key $rk^{(a)}_{A\to B} : \forall a \in \{1, 2, \ldots, n\}$ are given as:

$$rk^{(1)}_{A\to B} = \{(k_{1_1}, var_{1_1}), (k_{1_2}, var_{1_2}), \ldots, (k_{1_d}, var_{1_d})\}$$

$$rk^{(2)}_{A\to B} = \{(k_{2_1}, var_{2_1}), (k_{2_2}, var_{2_2}), \ldots, (k_{2_d}, var_{2_d})\}$$

$$\vdots$$

$$rk^{(n)}_{A\to B} = \{(k_{n_1}, var_{n_1}), (k_{n_2}, var_{n_2}), \ldots, (k_{n_d}, var_{n_d})\}$$

The equations might give an impression that the pairs (k_{a_i}, var_{a_i}) for each proxy P_a are all different. However, in reality, n such pairs are distributed among n proxies with each proxy getting d such pairs. Since the distribution of keys is also symmetric, each pair (k_i, var_i) is held by exactly d proxies.

A (k_i, var_i) pair held by d proxies and hence $n - d$ proxies don't contain that pair. Therefore a union of any $n - d + 1$ such proxies of will surely contain (k_i, var_i) pair. This is true for all the pairs. Thus, storing d pairs of (k_i, var_i) at each proxy leads to threshold $t = n-d+1$. In other words, to achieve threshold t, each of the n proxies must store exactly $n - t + 1$ pairs of (k_i, var_i). Re-encryption operation by each intermediate proxy involves checking by each proxy P_a, the condition $\mu \overset{?}{\equiv} 1 \mod k_{a_i}$. If yes, then ciphertext components $\widetilde{C_4}$ and $\widetilde{C_5}$ are modified as $\widetilde{C_4} = (\widetilde{C_4})^{k_{a_i}^{-1}}$ and $\widetilde{C_5} = \widetilde{C_5} - k_{a_i}$. Since union of any t $rk_{A \to B}^{(j)} \in \{rk_{A \to B}^{(1)}, \dots, rk_{A \to B}^{(n)}\} = \bigcup_{i=1}^{n} k_i$, the complete term of $\prod_{i=1}^{n} k_i$ will be removed from the exponent of $\widetilde{C_4}$. Incidentally, the component $\widetilde{C_5} = 0$. The finalized re-encrypted ciphertext is output as follows:

$$\overline{C_1} = M \oplus R, \overline{C_2} = R.e(g,g)^r, \overline{C_3} = g^{sk_A r},$$

$$\overline{C_4} = e(g,g)^{sk_B r}, \overline{C_5} = 0$$

Procedure **Dec** computes

$$\frac{\overline{C_2}}{(\overline{C_4})^{sk^{-1}}} = \frac{R.e(g,g)^r}{(e(g,g)^{sk_B r})^{sk_B^{-1}}}$$

$$= \frac{R.e(g,g)^r}{e(g,g)^r} = R$$

Dec now computes $M = \overline{C_1} \oplus R$. Verifies if $pk_A^{H(R,M)} = \overline{C_3}$ before it outputs M.

3.2.4 Properties of the proposed TP-PRE scheme

Properties satisfied by the proposed TP-PRE scheme are listed in Table 3.1. The proposed TP-PRE scheme satisfies unidirectionality as it is not possible to obtain $rk_{A \to B}$ given $rk_{B \to A}$ and all the public parameters. This is because in the proposed TP-PRE scheme, no re-encryption key can be generated without secret key of the delegator. Security of secret key of the delegator is proved. The scheme is non-interactive because it does not need the secret key of the delegatee U_B to generate the re-encryption key $rk_{A \to B}$. However, it needs the secret parameters $k_1, k_2, \dots, k_n$ to produce and re-encryption keys $rk_{A \to B}^{(0)}, rk_{A \to B}^{(1)}, \dots, rk_{A \to B}^{(n)}$ among the $n+1$ proxies. The proxy invisible property has been compromised for obtaining the desired functionality. Indeed, there are two different procedures for decrypting direct and re-encrypted ciphertexts. At any stage of re-encryption, the delegator U_A can compute $\widetilde{C_2} e(\widetilde{C_3}, g)^{-sk_A^{-1}} = R$.

Property	**Satisfied?**
Unidirectionality	YES
Non-interactivity	YES
Proxy Invisibility	NO
Original Access	YES
Key Optimality	YES
Collusion Safeness	YES
Temporariness	YES
Non-transitivity	YES
Non-transferability	NO

Table 3.1: Properties of the proposed TP-PRE scheme

This value of R is sufficient to obtain the underlying plaintext message M. Therefore, the delegator can obtain the plaintext from re-encrypted ciphertext at any stage of re-encryption. Thus, the proposed TP-PRE scheme satisfies original access property. The scheme is key-optimal because both delegatee and delegator store constant length secret keys. Collusion safeness of the proposed TP-PRE scheme is proved as part of the Standard Security. The scheme is non-transitive as it is computationally infeasible to obtain $rk_{A \to C}$ given $rk_{A \to B}$, $rk_{B \to C}$ and all the public parameters.

3.2.5 Security

The notion of security for PRE schemes defines two types of security namely *standard security* and *master-secret security*. Standard security captures the inability of a probabilistic polynomial time (PPT) adversary, say $\mathcal{A}$, to distinguish direct ciphertexts produced by the **Enc** and re-encrypted ciphertext produced by the **ReEnc** of the proposed TP-PRE scheme. In addition to the final output of the **ReEnc** procedure that is re-encrypted ciphertext $C_B'^{(0)}$, the definition for standard security of the proposed TP-PRE scheme must also capture indistinguishability of the intermediate ciphertexts $C_B'^{(l)} \forall l \in \{1, 2, \ldots, t\}$. Standard security must hold even if $\mathcal{A}$ colludes with one or more corrupted users denoted as U_c in the system. In the definition below, we denote honest users as U_a and the target user as U_{i^*}:

$$\begin{aligned}
Pr[(pk_{i^*}, sk_{i^*}) \leftarrow \textbf{KeyGen}(i^*, \textit{param}), \{(pk_a, sk_a) \leftarrow \textbf{KeyGen}(a, \textit{param})\},\\
\{(pk_c, sk_c) \leftarrow \textbf{KeyGen}(c, \textit{param})\}, \{rk_{a\to i^*} \leftarrow \textbf{ReKey}(sk_a, sk_{i^*})\},\\
\{rk_{i^*\to a} \leftarrow \textbf{ReKey}(sk_{i^*}, sk_a)\}, \{rk_{c\to i^*} \leftarrow \textbf{ReKey}(sk_c, sk_{i^*})\},\\
\{rk_{c\to a} \leftarrow \textbf{ReKey}(sk_c, sk_a)\},\\
(m_0, m_1, \alpha) \leftarrow \mathcal{A}(pk_{i^*}, \{pk_a\}, \{(pk_c, sk_c)\}, \{rk_{a\to i^*}\},\\
\{rk_{i^*\to a}\}, \{rk_{c\to i^*}\}, \{rk_{c\to a}\}, C_{i^*}^{(0)}, s \leftarrow \{0,1\},\\
s' \leftarrow \mathcal{A}_k(\alpha, pk_{i^*}, \textbf{Enc}(pk_{i^*}, m_s)) : s = s'] < \tfrac{1}{2} + \tfrac{1}{poly(k)}
\end{aligned}$$

The expression above means that probability of distinguishing the ciphertext produced by procedure **Enc** of the proposed TP-PRE scheme by a PPT adversary $\mathcal{A}$ is negligible. This distinguishability holds even if $\mathcal{A}$ corrupts some users and colludes with them. Standard security for re-encryption procedure can also be defined in the same way except the procedure **Enc** is replaced by **ReEnc** procedure in the definition and the input ciphertext $C_A^{(0)}$ is replaced with $C_B'^{(l)} \forall l \in \{1, 2, \ldots, t\}$. Standard security if the procedure **ReEnc** is defined as:

$$\begin{aligned}
Pr[(pk_{i^*}, sk_{i^*}) \leftarrow \textbf{KeyGen}(i^*, \textit{param}), \{(pk_a, sk_a) \leftarrow \textbf{KeyGen}(a, \textit{param})\},\\
\{(pk_c, sk_c) \leftarrow \textbf{KeyGen}(c, \textit{param})\}, \{rk_{a\to i^*} \leftarrow \textbf{ReKey}(sk_a, sk_{i^*})\},\\
\{rk_{i^*\to a} \leftarrow \textbf{ReKey}(sk_{i^*}, sk_a)\}, \{rk_{c\to i^*} \leftarrow \textbf{ReKey}(sk_c, sk_{i^*})\},\\
\{rk_{c\to a} \leftarrow \textbf{ReKey}(sk_c, sk_a)\},\\
(m_0, m_1, \alpha) \leftarrow \mathcal{A}(pk_{i^*}, \{pk_a\}, \{(pk_c, sk_c)\}, \{rk_{a\to i^*}\},\\
\{rk_{i^*\to a}\}, \{rk_{c\to i^*}\}, \{rk_{c\to a}\}, C_{i^*}'^{(l)} \forall l \in \{1, 2, \ldots, t\}\ s \leftarrow \{0,1\},\\
s' \leftarrow \mathcal{A}_k(\alpha, rk_{a\to i^*}, C_a^{(0)}, C_{i^*}'^{(l)} \forall l \in \{1, 2, \ldots, t\}) : s = s'] < \tfrac{1}{2} + \tfrac{1}{poly(k)}
\end{aligned}$$

Master-secret security of the scheme captures the security of the master secret of the delegator even against a corrupted delegatee. That is:

$$
\begin{aligned}
&Pr[(pk_{i^*}, sk_{i^*}) \leftarrow \textbf{KeyGen}(i^*, param), \{(pk_a, sk_a) \leftarrow \textbf{KeyGen}(a, param)\},\\
&\{(pk_c, sk_c) \leftarrow \textbf{KeyGen}(c, param)\}, \{rk_{a\rightarrow i^*} \leftarrow \textbf{ReKey}(sk_a, sk_{i^*})\},\\
&\{rk_{i^*\rightarrow a} \leftarrow \textbf{ReKey}(sk_{i^*}, sk_a)\}, \{rk_{c\rightarrow i^*} \leftarrow \textbf{ReKey}(sk_c, sk_{i^*})\}\\
&\{rk_{i^*\rightarrow c} \leftarrow \textbf{ReKey}(sk_{i^*}, sk_c)\}, \{rk_{a\rightarrow c} \leftarrow \textbf{ReKey}(sk_a, sk_c)\},\\
&\{rk_{c\rightarrow a} \leftarrow \textbf{ReKey}(sk_c, sk_a)\},\\
&\alpha \leftarrow \mathcal{A}(pk_{i^*}, \{pk_a\}, \{(pk_c, sk_c)\}, \{rk_{a\rightarrow i^*}\}, \{rk_{i^*\rightarrow a}\}, \{rk_{c\rightarrow i^*}\}, \{rk_{i^*\rightarrow c}\},\\
&\{rk_{a\rightarrow c}\}\{rk_{c\rightarrow a}\} : \alpha = sk_{i^*}] < \frac{1}{poly(k)}
\end{aligned}
$$

The probability expression above is an indicator of negligible advantage of $\mathcal{A}$ in obtaining the master-secret of the target user U_{i^*} given all the public keys and the re-encryption keys with corrupt delegatee.

3.2.6 Performance Analysis

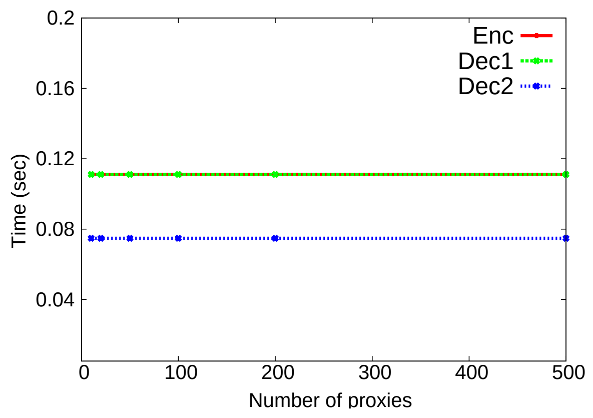

Figure 3.2: Variation of time required for Enc, Dec_1 and Dec_2 with respect to the number of proxies.

The proposed TP-PRE scheme progressively transforms a direct ciphertext intended for a delegatee so that the re-encrypted ciphertext is decryption-ready only after t identical re-encryption steps. For this, number of re-encryption key components stored by each re-encrypting proxy is not constant. In this section, we analyse the proposed TP-PRE scheme based on the threshold t required by the system, number of re-encryption

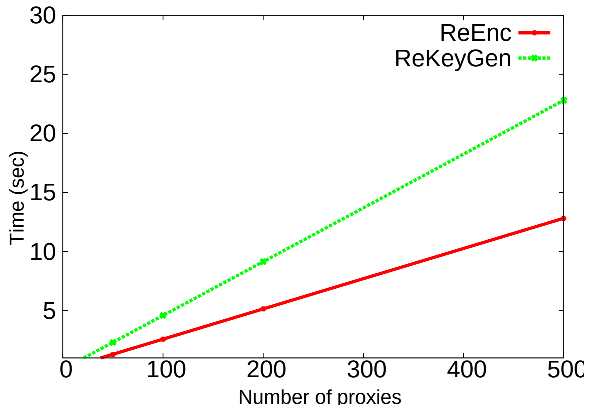

Figure 3.3: Time required for re-encryption and re-encryption key generation.

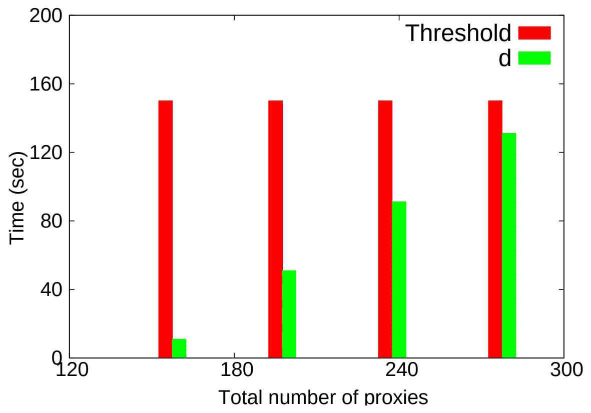

Figure 3.4: Variation of length (d) of re-encryption key to be stored by each proxy with threshold and total number of proxies

key components d to be stored by each proxy and computation cost for each step in the proposed TP-PRE scheme.

We implemented the proposed TP-PRE scheme using the Pairing-based Cryptography library on a system supporting 4 GB RAM, Intel Core i3 processor, having 64 bit Ubuntu 16.04 LTS. Computation cost in terms of time required for procedures **Enc**, **Dec$_1$** and **Dec$_2$** is compared in Fig. 3.2. Since these procedures do not depend on the number of proxies in the system, computation cost for these procedures does not vary with N. In Fig. 3.3, comparison of computation cost for the procedures re-encryption and re-encryption key generation is presented. These costs depends almost linearly on

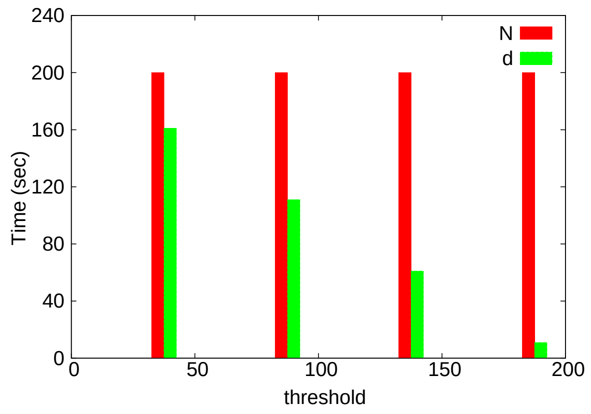

Figure 3.5: Variation of total number of proxies N with the length of re-encryption key for each proxy and threshold t

the number N of proxies. Fig. 3.4 presents comparison of threshold t required and the number of re-encryption key components d to be stored by each proxy. Its variation with the total number of proxies in the system is also depicted in the same graph. It can be seen that when the value t is fixed and number of participating proxies N is increased, then d also increases. This is because by construction, threshold t, number of proxies N and number of re-encryption key components stored by each proxy d are related as $t = N - d + 1$. Therefore, when t is fixed, and N is varied, d, which is linearly dependent on both t and N, increases to balance t. Similarly, in Fig. 3.5 presents variation of d and N when there is an increase in the threshold value t. Note that as threshold increases, while keeping N constant, d decreases. Again, by construction, and the linear relation between t, N and d, the graph shows decrease in the value of $d = N - t + 1$ when t is increased keeping N constant.

3.3 Limitations

Implementing this scheme seems to solve the problem of single point of failure in proxies but has many limitations of its own:

- Any BIBD designed should satify the equation
  $(v-1) \ = k(k-1)$
  Where v is the number of intermediate proxies and k is the number of keys stored at each proxy. This limits the situations (combination of keys and proxies) where this scheme can be applied.

- The relation between the threshold proxies required(t) out of all intermediate proxies(v) and the number of keys stored at each proxy(k) is given by :
  $t \ = v \ - k \ + 1.$
  $\therefore$ When we want to give larger number of proxies the liberty to fail, we have to store more keys at each proxy.

3.4 Summary

We have motivated a new primitive for progressive proxy re-encryption with threshold namely, Threshold Progressive Proxy Re-encryption (TP-PRE). We have formalized the definitions, standard and master-secret security notions for TP-PRE and provided a construction. The construction for TP-PRE has been proved secure under standard security and master-secret security. The performance analysis presenting theoretical bounds and outcome of practical implementation have been presented.

The challenge remains to tackle the limitations to the scheme caused due to the primitive involved i.e BIBD. This prevents the applicability of the scheme in all scenarios and hence a construction independent of BIBD should be designed which solves the problem with same or even lesser keys per node.

Chapter 4

Polynomial Root Finding

Computing the real roots of a univariate polynomial is a fundamental problem studied in detail in a branches of mathematics such as Numerical analysis. The problem of finding points where a polynomial evaluates to a certain value can be reduced to finding the roots of a polynomial. The problem has several applications in Computer algebra. The problem of equating a polynomial to a particular value c is equivalent to find the root of that polynomial with c subtracted from it. The problem of finding the irreducible components of a given algebraic set can be solved using a univariate polynomial root finder [30].

Several solutions have appeared to find the roots of a polynomial [20]. Most of the solutions that compute the roots with bounded bit complexity focuses on finding roots that are easy to isolate (i.e. roots that are far from each other and single). Pan *et. al* [22] proposed that computes the absolute value of the root with some relative error and isolates the roots if they are well separated. In [19], an algorithm is proposed which combines Descartes method with Newton iteration and approximate arithmetic.

4.1 Notation

In this section, we give the notation used throughout the chapter.

- $f(x)$: A univariate real polynomial equation whose coefficients may not be rational but can be approximated well [18].
- a_i : i^{th} coefficient of the polynomial $f(x) = \sum_{i=0}^{n} a_i \times x^i$.
- n : Degree of the polynomial.
- τ : 2^τ is the smallest upper bound for the absolute value of the coefficients i.e. $|a_i| \leq 2^\tau, \forall\, a_i$.
- Γ : It is the smallest positive integer such that $I.right \leq 2^\Gamma$ and $I.left \geq -2^\Gamma$.
- I : The interval in which we search for the roots of a polynomial.
- For any interval j:
  - $j.left$: left open end-point of j.
  - $j.right$: right open end-point of j.
  - $j.mul$: upper bound on the number of roots in j (multiplicity). It is always positive.
- $[c, d]$: An open interval that has c and d as its left and right end points respectively.
- p : User-defined parameter is used to determine the size of the intervals that isolate the root clusters.
- τ_i : τ of the polynomial at recursion depth = i. τ_0 = τ. i.e. After the polynomial has been differentiated i times and is divided by $i!$. Refer to Lemma 5.1.
- $f'(x)$: The first derivative of $f(x)$ with respect to x.
  $f'(x) = \sum_{i=0}^{n-1} (i+1) \times a_{i+1} \times x^i$

- $\tilde{O}(.)$: Bit complexity up to poly-logarithmic factors.
- $log(.)\ :\ log_2(.)$
- L : The precision (number of bits after the decimal point) with which we evaluate the polynomial. Refer to section 4.1.
- $MPSE(n, \tau, \Gamma, L)$: The number of bit operations to evaluate the sign of the polynomial at $O(n)$ points by evaluating the value of the value of the polynomial with an absolute error of 2^{-L}.
- $f(x)/r$: The division of the coefficients of $f(x)$ by r.
- random(l, r) returns a random point in between l and r (including l and r) accurate up to p bits after the decimal point.

4.2 Literature Survey

Victor Pan et al. [22] proposed a solution which runs in $\tilde{O}(n^2\tau)$ bit operations (Algorithm 1 in [22]). However, it may not retrieve all the roots. It only guarantees to retrieve all roots that are single and at a certain distance from each other. For every root, it obtains two intervals that contain the roots. If all of these intervals are non-overlapping, then we can easily find which intervals contain the root and then refine it (Reduce the length). This is a fast solution when the roots are well separated. We explain how to combine this solution with our own in Section 7 to improve the performance of the algorithm if the roots of the polynomial or any of its derivatives are well-separated.

Michael Sagraloff et al. [19] proposed a solution which finds all the real roots of a square-free polynomial (all roots are single) in $\tilde{O}(n(n^2{+}nlog(Mea(f)){+}log(M(Disc(f)^{-1})))$ bit operations (Algorithm 1 in [19]). The main takeaway is that it depends on the separation of the roots and sharply increases as the separation becomes very low.

Victor Pan et al. [26] proposed a solution which requires $\mathcal{O}(rnlog(n))$ arithmetic operations where r is the number of real roots. It computes the eigen values of the companion matrix which corresponds to the roots of the polynomial.

Herbert S. Wilf proposed a solution to find all roots of a polynomial. It requires $\mathcal{O}(n^3)$ arithmetic operations. It uses Sturm sequences to count the number of zeros in a rectangle of a complex plane.

Siegfried Rump [25] and Arnold Neumaier [24] proposed a solution to obtain the root clusters. Oliver Aberth [28] proposed an iterative method to compute the roots with cubic rate of convergence. Atiyah Wan Mohd Sham et al. [27] proposed a solution to compute bounds that enclose the roots of a polynomial and provided empirical data on the number of iterations and CPU time. However, the authors do not analyze the bit complexity of their methods. To understand the scalability of such algorithms regarding polynomials, calculating the bit complexity is very helpful.

Both our proposed algorithms make progress in both these areas. They obtain all roots, regardless of their multiplicity and distance between each other and they have a guarantee in bit complexity in terms of the input.

4.2.1 Our contributions

We introduce two algorithms that isolate all the real root clusters of a polynomial in an input interval I. We make absolutely no assumptions of the multiplicity of the roots and the distance between roots. The bit complexity required is upper-bounded by a function of the input only.

The first algorithm isolates the real root clusters in intervals of size $2^{-p} \times (\frac{13}{6}n^3 + \frac{99}{6}n^2 + \frac{151}{3}n - 67)$ with an expectation of $\tilde{\mathcal{O}}((p + \Gamma)n^2(np + n\Gamma + \tau))$ bit operations. The second algorithm isolates the real root clusters in intervals of size $2^{-p} \times (6n^2 + 34n - 38)$ with $\tilde{\mathcal{O}}((p + \Gamma)n^2(n(n + p + \Gamma) + \tau))$ bit operations.

Each interval will be assigned a multiplicity, which is an upper bound on the number of roots in that interval. However, the sum of multiplicities will never exceed n.
Both these algorithms recursively find the roots of a polynomial so it can be modified to find the roots of the derivatives of $f(x)$ with very little additional cost.
The uniqueness of the proposed algorithms is it provides we can set the value of the user-defined parameter p so that roots that are a certain user-defined distance apart are obtained as separate roots. Also we provide guarantees in performance and correctness after rounding off the coefficients, including the case where the coefficients are irrational.

Applications: Grobner basis is used to reduce solving a system of polynomials with multiple variables to finding the roots a single large polynomial. The roots of this polynomial may be badly conditioned and exist in clusters [33]. Also the real roots of this large polynomial correspond to the real solutions to the system of equations. Hence, our algorithms can be used to find the real roots of the single large polynomial. For a simpler example, consider a situation where we are performing some operations on a polynomial. We may need to know in advance at what points the polynomial takes a certain value. More specific examples are :

(i) If we are computing $(g(x) - a)^{-1}$ where $g(x)$ is a polynomial, we would like to know in advance where $g(x)$ takes the value a. To do this, we need to calculate the roots of $f(x) = g(x) - a$.

(ii) If we are calculating the derivative of $|g(x) - a|$, we would like to know in advance where $g(x)$ takes the value a.

4.3 Overview of the Proposed Algorithms

A polynomial of degree 1 is a straight line and it can be calculated easily. We claim that between two points where $f(x)$ is only increasing or only decreasing, there is at most one root between those two points and the root is single. It is well known that all differentiable function are only increasing or only decreasing between two consecutive roots of its derivatives.
We can check if there is a root in such an interval by checking the signs of the end points of the interval. If there is a root then $f(a)$ and $f(b)$ have opposite signs.
Refer to Lemma 5.2 for proof of above claims.

The procedure can broadly be described as:

(i) Recursively find intervals that contain all the roots of $f'(x)$. Each interval is assigned a multiplicity that is an upper bound for the number of roots of $f'(x)$ in that interval.

(ii) **Merging phase:** If the distance between two intervals is very small, merge them into one interval. The criteria for merging is described separately for both algorithms. Merge is described as: Let c, d be the two intervals and e be the new interval formed after merging.

$$e.left = min(c.left, d.left)$$
$$e.right = min(c.right, d.right)$$
$$e.mul = c.mul + d.mul + 1$$

(iii) **Exploration phase:** For both end points of each interval (containing roots of $f'(x)$), find a point close to the end point but outside the interval where we can determine the sign of $f(x)$.

(iv) We determine whether a root exists in between the right end-point of one interval and the left end-point of the next interval. This can be done by checking if the

signs of the polynomial at such end points are different.
This is also done for the interval formed by $I.left$ and the left end-point of the left most interval as well as the interval formed by $I.right$ and the right end-point of the right most interval.

(v) If the signs are not different, then we simply assign the root to the interval such that $f(x)$ has a smaller absolute value at that end-point than the other end-point. The reasoning for this explained in the next paragraph.

(vi) **Refinement phase:** If the signs are different then we know that interval has exactly one single real root in it. We refine the interval using different methods for both algorithms.

Step $2, 3$ and 6 have been named because those are the main areas where Algorithm 1 and Algorithm 3 defer.

The solution does not stop here. We have to check if there is a root in the intervals obtained recursively that contain the roots of $f'(x)$.

It can be difficult to decide the exact number of roots in such an interval. Instead of finding the exact number of roots in such an interval, we will upper bound the number of roots in the interval.

Consider three consecutive intervals obtained from the recursive call: $[a, b]$, $[c, d]$, $[e, f]$. Let k an upper bound on the number of roots of $f'(x)$ in $[c, d]$. Using Lemma 5.2, it can easily be proved that:

(i) $[c, d]$ contains at most $k + 1$ roots of $f(x)$.

(ii) If $[b, c]$ contains a root OR $|f(c)| > |f(b)|$ then $[c, d]$ contains at most k roots of $f(x)$.

(iii) If $[d, e]$ contains a root OR $|f(d)| > |f(e)|$ then $[c, d]$ contains at most k roots of $f(x)$.

(iv) If both 2 and 3 are true then $[c, d]$ contains at most $k - 1$ roots of $f(x)$.

It is easy to see that either the condition in 2 is false for $[c, d]$ or the condition in 3 is false for $[b, c]$ but not both. The same is true for $[c, d]$ and $[e, d]$.

Consider the intervals obtained recursively. Let S be the sum of the multiplicities assigned to these intervals. It is easy to observe that the algorithm will output intervals and assign them multiplicities such that their sum does not exceed S+1. This maintains the fact that the sum of multiplicities do not exceed n.

Consider $f(x) = x^2 - \epsilon$ a = -100, b = $\sqrt{\epsilon} + \delta$ and c = 100 where ϵ, δ are very small positive constants. We correctly assume that there is a root near $x = \sqrt{\epsilon} + \delta$.

Consider $f(x) = x^2 + \epsilon$, a = -100, b = δ and c = 100 where ϵ, δ are very small positive constants. We wrongly assume that there is a root near $x = \delta$.

However, this solution may return more intervals than necessary. Using this method, if $f(x)$ has very few real roots, then we would be doing a lot of unnecessary work. This can be avoided by applying methods such as Sturm's theorem [31] to find intervals that do not contain real roots at some recursion levels and discard them.

In the following subsection, we describe subroutines commonly used in our solutions:

4.3.1 Evaluating the sign of a polynomial

Consider a point $x = x_0$. Suppose we want know if $f(x_0)$ is positive or negative. Using [17], we can approximate $f(x_0)$ with $\tilde{f}(x_0)$ such that $|f(x_0) - \tilde{f}(x_0)| < 2^{-L}$ in $\tilde{O}(n(L + n\Gamma + \tau))$ bit operations. Here L is user-defined.
It is straight forward to check if $\tilde{f}(x_0) > 2^{-L}$ then we can say with certainty that the sign we have obtained is correct.
If $f(x_0) > 2^{-L+1}$ then we will always be able to say with certainty that our answer is

correct.
Hence, if x_0 is at a distance of D from all the roots of $f(x)$ then $L = nlog_2(D^{-1}) + 1$ is sufficient precision to determine the sign of the polynomial.

4.3.2 Comparing the value of the polynomial at two points

Consider two points, $x = x_1$ and $x = x_2 : x_1 < x_2$. Suppose we want to know if $|f(x_1)| < |f(x_2)|$. Since we can approximate the two values as $|\tilde{f}(x_1)|$ and $|\tilde{f}(x_2)|$ within an absolute error of 2^{-L}, we can approximate $|f(x_1)| - |f(x_2)|$ within an error bound of 2^{-L+1}. However, if we know that $f(x)$ is only increasing in $[x_1, x_2]$ and $f(x_1) > 0$ then we know that $|f(x_1)| < |f(x_2)|$.

4.3.3 Approximate Bisection Method

Approximate Bisection Method can be used to reduce the size of an interval isolating a single root by half ([18], Algorithm 2). It evaluates $f(x)$ at the 3 quarters of the interval and refines it based on the sign of its point. Let i be an interval of size $|i|$ that isolates a single real root. Approximate Bisection Method ensures that at least two points of evaluation is at a distance of at least $\frac{|I|}{8}$ from all the real roots. The precision is assigned some starting value. If the precision is not sufficient to accurately determine the sign of the polynomial at at least 2 out of 3 points (the quarters) then the iteration is said to have failed. In this case, the process is repeated after doubling the precision.
If it does not fail, then it reduces the size of the interval to at least half while maintaining the invariant that $f(x)$ has opposite signs at the end points.
Let i_0 is the initial size of the interval.
Let i_f is the desired size of the interval after refinement.
Then the number of successful iterations required is at most $\lceil log_2 \frac{i_0}{i_f} \rceil$.
Let L_0 be the initial precision used.
Let L_{max} be an smallest upper bound precision required for any iteration to succeed.

Then, the number of failed iterations is $\lceil log_2 \frac{L_{max}}{L_0} \rceil$.
Hence, the total number of iterations required is at most $\lceil log_2 \frac{i_0}{i_f} \rceil + \lceil log_2 \frac{L_{max}}{L_0} \rceil$.

4.4 Claims

In this section, we shall establish some facts that will help prove the correctness of the algorithm.

4.4.1 Growth of values of coefficients

Lemma 4.4.1. *$\tau_i \ < \ \tau + n$. i.e the difference between the bit size of the coefficients of $f(x)$ and the bit size of the coefficients of the polynomial in each recursive call is less than the degree of $f(x)$.*

Let $f^r(x)$ be the r^{th} derivative of $f(x)$.
$f^r(x) = \sum_{i=r}^{n} \ {}^iP_r a_i x^{i-r}$.
$f^r(x)/r! = \sum_{i=r}^{n} \ \binom{i}{r} a_i x^{i-r}$.
$\binom{i}{r} \ < \ 2^i \ \leq \ 2^n \ \forall \ i, r$ (Binomial Theorem).
$\tau_i \ < \ \tau + n$
Hence, $MPSE(n, \tau_i, \Gamma, L) = \tilde{O}(\ MPSE(n, \tau, \Gamma, L) \ ) \ \forall \ i$. A tighter bound can be obtained using Sterling's approximation but it does not make any difference in the complexity analysis.

4.4.2 At most one real root may exist between derivatives

Lemma 4.4.2. *Let a and b be two consecutive roots of $f'(x)$:*

(i) There can be at most one root of $f(x)$ in $[a, b]$.

(ii) If such a root exists then it must be single.

(iii) A root is present in $[a, b]$ if and only if $f(a)$ and $f(b)$ have opposite signs.

Part 1: Let a and b be 2 consecutive points where $f'(x) = 0$. Wlog, assume $f'(x) > 0 \; \forall \, x \in (a, b)$. Then by definition of derivative, $f(x)$ is increasing in that interval.
Assume there is a root c in that interval. Then, $f(c) = 0$.

$$f(x) > 0 \; \forall \, x \in (c, b).$$
$$f(x) < 0 \; \forall \, x \in (a, c)$$

Hence, $x = c$ is the only root in (a, b). Also, $f(b) > 0$ and $f(a) < 0$. Hence, we also proved that a and b have opposite signs.

Part 2: Assume $x = c$ is a root of multiplicity more than 1. Then $f(x)$ can be expressed as:

$$f(x) = p(x) \times (x-c)^k : k > 1$$
$$f'(x) = k \times f(x) \times (x-c)^{k-1} + p'(x) \times (x-c)^k$$
$$\text{if } j > 1 \text{ then } (x-c)^{j-1} = (x-c)^j = 0 \text{ at } x = c$$

Hence, $f'(c) = 0$ and $x = c$ is a root of $f'(x)$. This contradicts the fact the a and b are two consecutive roots of the first derivative.

Part 3: ONLY IF: This was already proved in Part 1.
IF $f(x)$ is a continuous function, w.l.o.g, if $f(a) < 0$ and $f(b) > 0$ then there must be a point $c : f(c) = 0$.

4.5 Randomized Algorithm

Here we will describe and analyze Algorithm 1, which isolates the real root clusters in intervals of size $2^{-p} \times (\frac{13}{6}n^3 + \frac{99}{6}n^2 + \frac{151}{3}n - 67)$ with an expectation of $\tilde{\mathcal{O}}((p + \Gamma)n^2(np + n\Gamma + \tau))$ bit operations. We will be referring to the steps given in section 4. $List$ mentioned here corresponds to $Derivatives$ in the algorithm. It is used to contain the intervals containing the roots of $f'(x)$ in ascending order.

1. We differentiate $f(x)$ and divide it with $rec\_depth + 1$. Then we recursively find the root clusters of this polynomial. Decrement the multiplicity of each interval in this $list$ by 1 (Explained in step 5). Append two intervals of zero length with multiplicity 0 to this $list$. One interval is $[I.left, I.left]$ and the other is $[I.right, I.right]$.

2. Merging phase: Merge any two intervals if the distance between any two points in the interval less than $(n + 3) \times 2^{-p+3}$ using the method described in step 2.

3. Exploration phase: Set the precision (L) to $pn + 2$. Hence, if a point is at least at a distance of 2^{-p} from all roots then we will be able determine the sign with certainty, as mentioned in section 4.1.
For every two consecutive intervals in $list$, let i be the interval formed by the right end-point of one interval and the left end-point of the next consecutive interval. Choose a random point in $[i.left + 2^{-p}, i.left + (n + 3) \times 2^{-p+1}]$ and evaluate it. Do the same in $[i.right - (n + 3) \times 2^{-p+1}, i.left - 2^{-p}]$. Repeat this procedure until a point near the end-points of every such i has been found such that their their sign can be evaluated with certainty. Refer to section 4.1.

4. For every i described above, if $f(x)$ has opposite signs at the end points, then queue that interval for root refinement. The intervals to be refined are placed into I_list.

5. If they have the same sign, then see which point has smaller magnitude. The method to do this is mentioned in section 4.2. Assume that there is a root between this point and the interval from list to which this point corresponds to. Extend that interval to include this point and increase its multiplicity by 1. It is easy to see that this will have the same effect as upper bounding the multiplicities described in section 4. Discard all intervals that have multiplicity less than 1.

6. Refinement phase: The size of the intervals are reduced using Algorithm 2 on all the intervals queued for root refinement. It is a simple algorithm where we randomly pick a point in the interval. Let M denote the size of the smallest interval in I_list that is unrefined (size is greater than $5n \times 2^{-p}$). If the sign can be obtained with certain using a precision of $n\lceil log(\frac{2M}{5(n+4)})\rceil + 2$ then we shorten the interval while maintaining the invariant that the polynomial takes opposite signs at the end-points. This is repeated until the size of each interval is at most $5n \times 2^{-p}$. Create a sorted list by combined $list$ and the refined intervals and return it.

Note: $eval[i.left]$ is used to keep track of an x-coordinate near $i.left$, $eval[i.left].x = x_0$ that is not in i and the corresponding value $eval[i.left].y = \tilde{f}(x_0)$. If a point could not be found yet such that we could determine the sign $eval[i.left].y = Unkown$. I_list is queue the intervals for refinement.

Algorithm 1 Non-deterministic algorithm to get roots of polynomial

```
1: procedure RANDOM_REAL_ROOTS($f, I, p, rec\_depth = 0$)
2:     if $n$ = 1 then
3:         if $I$.left ≤ -$a_0$/$a_1$ ≤ $I$.right then return any interval of size $2^{-p+1}$ containing
           -$a_0$/$a_1$, contained within $I$ with $multiplicity$ = 1.
4:         else return Nothing.
5:     $poly\_der$ ← derivative($f$)/($rec\_depth + 1$)
6:     $Derivatives$ ← RANDOM_REAL_ROOTS($poly\_der$, $I$, $p$, $rec\_depth + 1$)
7:     Decrement the multiplicity of each interval in $Derivatives$ by 1.
8:     Append ($I.left$, $I.left$) and ($I.right$, $I.right$) with $multiplicity$ = 0 to the left
       and right of $Derivatives$.
9:     Merge any two intervals in $Derivatives$ for which the distance is less than
       $(n + 3) \times 2^{-p+3}$.
10:    for $c \in Derivatives$ do $eval[c.left].y$, $eval[c.right].y$ ← Unknown.
```

```
11:     for Consecutive $c, d \in Derivatives$ do Mark $[c.right, d.left]$ as unexplored
12:     $L \leftarrow p \times n + 2$
13:     while $\exists\, c \in Derivatives$ $eval[c.left].y$ OR $eval[c.right].y$ = Unknown do
14:         for Consecutive $c, d$ in $Derivatives$ do
15:             if $eval[c.right].y$ = Unknown then
16:                 $eval[c.right].x$ = random($c.right + 2^{-p}$, $c.right + (n+3) \times 2^{-p+1}$)
17:                 if sign of $eval[c.right].x$ can be found then $eval[c.right].y = f(eval[c.right].x)$
18:             if $eval[d.left].y$ = Unknown then
19:                 $eval[d.left].x$ = random($d.left - (n+3) \times 2^{-p+1}$, $d.left - 2^{-p}$)
20:                 if sign of $eval[d.left].x$ can be found then $eval[d.left].y = f(eval[d.left].x)$
21:             if $eval[c.right].y$, $eval[d.left].y \neq$ Unknown AND $[c.right, d.left]$ is
    unexplored then
22:                 if $eval[c.right].y$ and $eval[d.left].y$ have different signs then
23:                     Append [$eval[c.right].x$, $eval[d.left].x$] to $I\_list$ with $multiplicity$
    = 1.
24:                     Append $eval[d.left].y > 0$ to $inc$.
25:                 else
26:                     if $eval[c.right].y$, $f(d.left)$ have different signs then        ▷ i.e.
    $|eval[c.right].x| > |eval[d.left].x|$
27:                         $d.left \leftarrow eval[d.left].x$
28:                         $d.mul \leftarrow d.mul + 1$
29:                     else
30:                         $c.right \leftarrow eval[c.right].x$
31:                         $c.mul \leftarrow c.mul + 1$.
32:                 Mark [$c.right$, $d.left$] as explored
33:     Remove all intervals in $Derivatives$ with non-positive multiplicity.
34:     Refine_intervals($f$, $I\_list$, $inc$, $p$)
35:     Merge $I\_list$ into $Derivatives$ sorted by x-coordinate.
36:     return $Derivatives$
```

Algorithm 2 Refine inceasing/decreasing intervals with a single root

```
1: procedure REFINE_INTERVALS($f, I\_list, inc, p$)
2:     for $i \in I\_list$ do Mark $i$ as unrefined
3:     while $\exists i \in I\_list : iisunrefined$ do
4:         $M \leftarrow argmax_{i\ \in\ I\_list\ :\ i\ is\ unrefined}|i|$
5:         $L \leftarrow n \times \lceil log(\frac{2M}{5(n+4)})\rceil + 2$
6:         for $i \in I\_list$ do
7:             if $i$ is unrefined then
8:                 $a \leftarrow$ random($i.left + 2^{-p}$, $i.right - 2^{-p}$)
9:                 if The sign of f(a) can be found then
10:                    Shorten $i$ such that its end points have the same sign
11:                if $|i| \leq 5 \times n \times 2^{-p}$ then Mark $i$ as refined.
12:    return $I\_list$
```

4.5.1 Expected Iterations to explore intervals

Lemma 4.5.1. *The expected number of iterations required to explore the intervals is* $\tilde{O}(1)$.

There can be at most $\frac{n}{2}$ pairs of complex roots in an interval and 1 real root in an interval that is increasing/decreasing. Hence, the total length of the $x-axis$ that is with a distance of 2^{-p} from a root is at most $(n+2) \times 2^{-p}$. Hence, randomly picking a point in an interval of size $(n+2) \times 2^{-p+1}$ will succeed with at least probability 0.5.

Let m be the number of intervals obtained from the recursion.

$m+1$ intervals are formed by using the consecutive end points of the m intervals, $I.left$ and $I.right$. Hence, there will be $2m+2$ end points to explore.

Since the explorations are being done in parallel using multipoint polynomial evaluation, the time taken to explore all the end points will depend on the supremium of the time taken to explore each end-point.

Let $T_{explore}$ be a random variable that denotes the number of iterations required to explore the intervals and X_i be a random variable that denotes the number of iterations required to explore the i^{th} interval.

$T_{explore} = max_{i=1}^{2m+2}\{X_i\}$ where X_i are geometric random variables with mean = 2.

$E[T_{explore}] < 1 + \frac{1}{-ln(1-p)} H_{2m+2}$ (from [21])
Where $H_n = \sum_{i=1}^{n} i^{-1} \approx log(n)$.
$E[T_{explore}] = \tilde{\mathcal{O}}(1)$.

4.5.2 Expected Iterations to refine intervals

Lemma 4.5.2. *The expected number of iterations required to refine the intervals is* $\tilde{\mathcal{O}}(\tau + p)$

Let I_i be one of the intervals we want to refine.
Recall that M refers to the size of the smallest unrefined interval at any iteration.
It can easily be proved by exchange argument that the worst case for each iteration of Algorithm 2 occurs when then $\frac{n}{2} + 1$ roots are located in the middle of one of the intervals to be refined at a distance of $\frac{2M}{5(n+4)}$ from each other.

Let $T(|I_i|)$ be a random variable that denotes the number of iterations required to refine $|I_i|$. $T(|I_i|)$ can be modeled as follows:

(i) If the interval is smaller than 5×2^{-p}, stop.

(ii) Randomly pick a point q in I_i.

(iii) If q is within a distance of $\frac{2M}{5(n+4)}$ from a root then repeat. This event is referred to as a failure.

(iv) If q is not within a distance of 2^{-p} from a root then reduce the interval by making q an end point such that the end points have opposite signs. Now repeat. This event is referred to as a success.

We will use a random variable $X(|I_i|)$ that upper-bounds $T(|I_i|)$ and it is easy to compute its expected value. We will assume the worst case where all the roots are clustered in the center of I_i at a distance of $\frac{2M}{5(n+4)}$ from the next nearest root.

$X(|I_i|)$ can be modeled as follows:

(i) If the interval is smaller than 5×2^{-p}, stop.

(ii) Randomly pick a point q in I_i.

(iii) If q is within a distance of $|I_i|/5$ from the end-points of the interval or within a distance of $\frac{2|I_i|}{5(n+4)}$ from a root then repeat. Note that this is a more lenient condition. This is referred to as a failure.

(iv) Let q_2 be a point that is at a distance of of $|I_i|$/5 from the end point closer to q. If q is not within a distance of $|I_i|/5$ from an end point and 2^{-p} from a root then reduce the interval by making $q2$ an end point such that the end points have opposite signs. Now repeat. Note that using q_2 as a new end point instead of q results in a larger interval. This is referred to as a success.

It is explained in step 2 and 3 why $X(|I_i|) \ \geq \ T(|I_i|)$.

$X(|I_i|)$ =

$$\begin{cases} 0 & |I_i| \leq 5 \times 2^{-p} \\ X(|I_i|) & p(fail) = \frac{2|I_i|/5+(n+4)M}{|I_i|} \\ X(\frac{4}{5}|I_i|) & p(success) = \frac{3|I_i|/5-(n+4)M}{|I_i|} \end{cases}$$

If $|I_i| > 5n \times 2^{-p}$ then $p(success) > 1/5$.

$$p(success) = \frac{3|I_i|/5 - (n+4)M}{|I_i|}$$
$$p(success) \geq \frac{3|I_i|/5 - 2|I_i|/5}{|I_i|}$$
$$p(success) \geq \frac{|I_i|}{5}$$

Let t_i denote the number of successful iterations required in $X(I_i)$.
Then $X(I_i)$ is upper-bounded by the sum of t_i geometric random variables with mean

= 5.

$X(I_i) \;=\; \sum_{j=1}^{t_i} Y_j$.

Where Y_j is a geometric random variable with mean = 5.

$|I_i|(\frac{4}{5})^{t_i} \geq 5n \times 2^{-p}$

$t_i \;<\; \lceil log_{5/4}|I_i| + \frac{p}{log_2 \frac{5}{4}} - log_{5/4}n \rceil$

$t_i \;<\; \lceil \frac{\Gamma}{log_2 \frac{5}{4}} + log_{5/4}2 + \frac{p}{log_2 \frac{5}{4}} - log_{5/4}n \rceil \; \forall \; T(I_i)$.

Let T_{refine} be the number of iterations required to refine the intervals.

Let m be the number of intervals we are refining.

Since the intervals are being refined in parallel using multi-point polynomial evaluation, the number of iterations required depends on the supremium of the iterations required by each interval.

$T_{explore} \;=\; max_{i=1}^{m} T(I_i)$.

$T_{explore} \;<\; max_{i=1}^{m} X(I_i)$

$T_{explore} \;<\; max_{i=1}^{m} \sum_{j=1}^{t_i} Y_{ij}$

$T_{explore} \;< \sum_{j=1}^{t}(max_{i=1}^{m} Y_{ij})$ where t = $max_{i=1}^{m} t_i$

The last inequality is because the sum of the largest element in each set is at least as large as the sum of the set with the largest sum.

i.e. Imagine a 2-D matrix $Y_{m \times n}$. It is obvious that:

$max_{i=1}^{n} \sum_{j=1}^{m} Y_{ij} \;\leq\; sum_{i=1}^{m} \max_{j=1}^{n} Y_{ij}$

From [21] $E[max_{i=1}^{m} Y_{ij}] \;=\; \tilde{O}(1)$.

$E[T_{refine}] = \tilde{O}(t)$.

$E[T_{refine}] = \tilde{O}(p + \Gamma)$.

4.5.3 Expected bit complexity

Lemma 4.5.3. *The algorithm calculates the roots of a polynomial in $\tilde{\mathcal{O}}((p+\Gamma)n^2(np+n\Gamma+\tau))$ expected bit operations.*

Let us analyze each recursive call where the degree of the polynomial is d.
In the exploration phase, precision (L) of at most $nd+2$ is used.
In the refinement phase, precision of $dlog(\frac{2M}{5(n+4)})+2$ is used (Refer to). All unrefined intervals have length at least $5d \times 2^{-p}$. Hence, $M > 5d \times 2^{-p}$, the precision used is upper-bounded by $d(p+1)+2$.
$E[T(n,\tau,\Gamma)]$: Expected bit complexity of Algorithm 1
$E[T_{explore}(d,\tau_{n-d},\Gamma)]$: Expected number of iterations in exploration phase of each recursive call
$E[T_{refine}(n,\tau,\Gamma)]$: Expected number of iterations in refinement phase of each recursive call
$MPSE(d,\tau_{n-d},\Gamma,d(p+1)+2)$: Bit complexity of each iteration in each recursive call.

$E[T(n,\tau,\Gamma)]$ =
$\sum_{d=2}^{n}(E[T_{explore}(d,\tau_{n-d},\Gamma)] + E[T_{refine}(d,\tau_{n-d},\Gamma)]) \times MPSE(d,\tau_{n-d},\Gamma,$
$d(p+1)+2)$

$E[T_{explore}(d,\tau_{n-d},\Gamma)] + E[T_{refine}(d,\tau_{n-d},\Gamma)] = \tilde{\mathcal{O}}(p+\Gamma)$.
(Refer to subsections 6.2 and 6.3)

$MPSE(d,\tau_{n-d},\Gamma,dp+2) = \tilde{\mathcal{O}}(d(dp+d\Gamma+\tau))$
(Refer to Lemma 5.1)

$E[T(n,\tau,\Gamma)]$ = $\sum_{d=2}^{n}(\Gamma+p)d(dp+d\Gamma+\tau)$

$E[T(n,\tau,\Gamma)]$ = $\tilde{\mathcal{O}}(\ (\Gamma+p)n^2(np+n\Gamma+\tau)\ )$

4.5.4 Size of intervals

Lemma 4.5.4. *For a polynomial of degree $n > 1$, the size of each interval i upper-bounded by* $2^{-p} \times (n(n+1)(\frac{13}{6}n^3 + \frac{99}{6}n^2 + \frac{151}{3}n - 67)$.

We will measure the maximum total length of all the intervals and use that as an upper-bound for each interval.
Since the increase in interval length due to merging and finding new intervals is much for than that of expanding an interval, it is easy to show that the total length will be maximum when each recursive call returns d intervals. Half of them are merged ($\lceil d/2 \rceil$+1) and $\lceil d/2 \rceil$ new intervals are created. The increase in length is upper bounded by the following function of the degree of the polynomial (As a multiple of 2^{-p}):

$f(d) =$

$$\begin{cases} 2 & d = 1 \\ 8(d+3)(\lceil d/2 \rceil + 1) + 5d\lceil d/2 \rceil & else \end{cases}$$

Summing the above function for $d = 1$ to n.

$$Interval\ Size \leq 2 - 69 + \sum_{d=1}^{n} (\ 13d\lceil d/2 \rceil\ + 8d + 24\lceil d/2 \rceil + 24\ )$$
$$Interval\ Size < -67 + \sum_{d=1}^{n} (\ 13d(d+1)/2 + 8d + 12(d+1) + 24\ )$$
$$Interval\ Size < -67 + \sum_{d=1}^{n} (\ 13d^2/2 + 53d/2 + 36\ )$$
$$Interval\ Size < -67 + 13(2n+1)(n+1)(n)/12 + 53(n)(n+1)/4 + 36n$$
$$Interval\ Size < \frac{13}{6}n^3 + \frac{99}{6}n^2 + \frac{151}{3}n - 67$$

4.6 Deterministic Algorithm

Here we will describe Algorithm 3 which isolates the real root clusters in intervals of size $2^{-p} \times (6n^2 + 34n - 38)$ with $\tilde{O}((p + \Gamma)n^2(n(n + p + \Gamma) + \tau))$ bit operations.

We will be referring to the steps in the Algorithm 1.

1. Same as Algorithm 1.

2. Merging phase: Merge any two intervals if the distance between any two points in the interval less than 2^{-p+4} using the method described in section 4, step 2.

3. Exploration phase: Set the precision (L) to any reasonably small value such as 1. It should not by much more than the upper bound shown in lemma 7.1. For every two consecutive intervals in $list$, let i be the interval formed by the right end-point of one interval and the left end-point of the next consecutive interval. Evaluate the sign of $f(x)$ at $x = i.left + 2^{-p+1}$ and $x = i.left + 2^{-p+2}$. Do the same in $[i.right - (n + 3)2^{p+1}$, $i.left - 2^{-p}]$. Double the precision used. Repeat this procedure until a point near the end-points of every such i has been found such that their their sign can be evaluated with certainty.

4. Same as Algorithm 1.

5. Same as Algorithm 1.

6. Run Approximate Bisection Method (section 4.3) on all the intervals queued for root refinement. Create a new list by combining $list$ and the refined intervals and return it. Approximate_Bisection(I_list, 2^{-p+3}) refines every interval in I_list to size less than 2^{-p+3}.

Note: $eval[i.left]$ is used to keep track of an x-coordinate near $i.left$, $eval[i.left].x = x_0$ that is not in i and the corresponding value $eval[i.left].y = \tilde{f}(x_0)$. $eval[i.left].y = Unknown$ if a point could not be found yet such that we could determine the sign with our current precision. I_list is queue the intervals for refinement.

Algorithm 3 Deterministic algorithm to get roots of polynomial

```
1:  procedure REAL_ROOTS($f, I, rec\_depth = 0$)
2:      if $n$ = 1 then
3:          if $I$.left $\leq$ -$a_0$/$a_1$ $\leq$ $I$.right then return any interval of size $2^{-p+1}$ containing
    -$a_0$/$a_1$, contained within $I$ with $multiplicity$ = 1.
4:          else return Nothing.
5:      $poly\_der \leftarrow derivative(poly)/(rec\_depth + 1)$
6:      $Derivatives \leftarrow$ REAL_ROOTS($poly\_der$, $I$, $rec\_depth + 1$)
7:      Decrement the multiplicity of each interval in $Derivatives$ by 1.
8:      Append ($I.left$, $I.left$) and ($I.right$, $I.right$) with $multiplicity$ = 0 to the left
    and right of $Derivatives$.
9:      Merge any two intervals in $Derivatives$ for which the distance is less than
    $2^{-p+4}$. If ($I.left$, $I.left$) or ($I.right$, $I.right$) was merged then decrement the
    multiplicity of the resulting interval by 1.
10:     for $c \in Derivatives$ do $eval[c.left].y$, $eval[c.right].y \leftarrow$ Unknown.
11:     for Consecutive $c, d$ in $Derivatives$ do Mark ($c.right$, $d.left$) as unexplored
12:     $L \leftarrow 1$
13:     while $\exists c \in Derivatives$ $eval[c.left].y$ OR $eval[c.right].y$ = Unknown do
14:         for Consecutive $c, d$ in $Derivatives$ do
15:             if $eval[c.right].y$ = Unknown AND sign of $f(c.right + 2^{-p+1})$ OR
    $f(c.right + 2^{-p+2})$ can be found then
16:                 $eval[c.right].x \leftarrow$ The point that can be evaluated
17:                 $eval[c.right].y \leftarrow f(eval[c.right].x)$
18:             if $eval[d.left].y$ = Unknown AND (sign of one of $f(d.left - 2^{-p+1})$ OR
    $f(d.left - 2^{-p+2})$ can be found ) then
19:                 $eval[d.left].x \leftarrow$ The point that can be evaluated
20:                 $eval[d.left].y \leftarrow f(eval[d.left].x)$
21:             if $eval[c.right].y$, $eval[d.left].y \neq$ Unknown AND $[c.right, d.left]$ is
    unexplored then
```

```
22:             if $eval[c.right].y$ and $eval[d.left].y$ have different signs then
23:                 Append $(eval[c.right].x, eval[d.left].x)$ to $I\_list$.
24:                 Append $eval[d.left].y > 0$ to $inc$.
25:             else
26:                 if $eval[c.right].y$, $f'(c.right)$ have different signs then    ▷ i.e. $|eval[c.right].x| > |eval[d.left].x|$
27:                     $d.left \leftarrow eval[d.left].x$
28:                     $d.mul \leftarrow d.mul + 1$
29:                 else
30:                     $c.right \leftarrow eval[c.right].x$
31:                     $c.mul \leftarrow c.mul + 1$.
32:             Mark $[c.right, d.left]$ as explored
33:         $L \leftarrow 2 \times L$.
34:     Remove all intervals in $Derivatives$ with non-positive multiplicity.
35:     Approximate_Bisection($I\_list$, $2^{-p+3}$)
36:     Merge $I\_list$ into $Derivatives$ sorted by x-coordinate.
37:     return $Derivatives$
```

4.6.1 Precision Required

In this subsection, we analyze the amount of precision (L) required to determine the sign of $f(x)$ at a desired point with certainty.

We evaluate $f(x)$ in an interval where at most one real root exists. We could easily bound the accuracy depending on the distance of the root from the point of evaluation if no other root existed in that interval. But the problem is there may be a complex root near the point of evaluation with a very small imaginary value such that $|(x_0 - a_i)^2 + b_i^2| << 2^{-p}$. It may possible to have an upper bound for b_i because if b_i is very small then $f_i(x) \times ((x - a_i)^2 + b_i^2)$ may behave like $f_i(x) \times (x - a_i)^2$ and have a root near $x = a_i$.

Lemma 4.6.1. *If $f(x)$ and $f'(x)$ have no real roots within a distance of 2^{-p+1} from a point of evaluation x_0 then the sign of $f(x_0)$ can be evaluated with a precision of $\tilde{O}(n \times (n+p))$.*

If

$$f(x) = \sum_{i=0}^{n} a_i \times x^i$$

is a polynomial then

$$\frac{f'(x)}{f(x)} = \sum_{a_i \in R_r} \frac{1}{x - a_i} + \sum_{a_i \pm ib_i \in R_c} 2\frac{x - a_i}{(x-a_i)^2 + b_i^2}$$

where R_r is the set of real roots of $f(x)$ and R_c is the set of complex roots of $f(x)$.

We assume that $\frac{f'(x)}{f(x)}$ is constant in $[x_0 - 2^{-p}, x_0 + 2^{-p}]$. We believe this might be an acceptable assumption because there are no real roots in it. It is known that $|\frac{1}{x-a_i}|$ does not change much when it is far away from $x = a_i$. Since the interval is of a small length $|2\frac{x-a_i}{(x_0-a_i)^2+b_i^2}|$ does not change much.

We are concerned with evaluating a polynomial very close to the real portion of a complex root.

Let $a_i \pm ib_i$ be a pair of complex roots such that $b_i < 2^{-p}$ and $|x_0 - a_i| < 2^{-p}$.
Let $f_i(x) = \frac{f(x)}{(x-a_i)^2+b_i^2}$. i.e. $f_i(x)$ after the complex roots $x = a_i \pm b_i$ have been removed.

$f(x) = f_i(x) \times ((x-a_i)^2 + b_i^2)$
$f'(x) = f_i'(x) \times ((x-a_i)^2 + b_i^2) + 2 \times f_i(x) \times (x - a_i)$

Consider $x = a_i + b_i$
$f'(a_i) = f_i'(a_i) \times b_i^2$

Consider $x = a_i + b_i$
$f'(a_i + b_i) = 2 \times f_i'(a_i + b_i) \times b_i^2 + 2 \times f_i(a_i + b_i) \times b_i$
Consider $x = a_i - b_i$
$f'(a_i - b_i) = 2 \times f_i'(a_i - b_i) \times b_i^2 - 2 \times f_i(a_i - b_i) \times b_i$

Since $b_i < 2^{-p}$, $f(x)$ has no root in $[a_i - b_i, a_i + b_i]$, $f_i(x)$ and $f'(x)$ has no root in $[a_i - b_i, a_i + b_i]$. Hence the sign of $f_i(x)$ does not change in $[a_i - b_i, a_i + b_i]$.
$f_i(a_i - b_i)$ and $f_i(a_i + b_i)$ have the same sign.
$-2b_i \times f_i(a_i - b_i)$ and $2b_i \times f_i(a_i + b_i)$ have the same sign.
$2b_i^2 \times f(a_i + b_i)$ and $2b_i^2 \times f(a_i + b_i)$ must have the same sign (otherwise there is a root in $[a_i - b_i, a_i + b_i]$).
For there to be no root, $f(a_i - b_i)$ and $f(a_i + b_i)$ must have the same sign.

$|2 \times f_i'(a_i + b_i) \times b_i^2| \ < \ |2 \times f_i(a_i + b_i) \times b_i|$
OR
$|2 \times f_i'(a_i - b_i) \times b_i^2| < |2 \times f_i(a_i - b_i) \times b_i|$

$|\frac{f'(a_i+b_i)}{f(a_i+b+i)}| > b_i^{-1}$ OR $|\frac{f'(a_i-b_i)}{f(a_i-b_i)}| > b_i^{-1}$.

By our assumption, $|\frac{f'(a_i-b_i)}{f(a_i-b_i)}|$ = $|\frac{f'(a_i+b_i)}{f(a_i+b_i)}| = |\frac{f'(a_i)}{f(a_i)}|$

$|\frac{f'(a_i)}{f(a_i)}| > b_i^{-1}$

We will prove an upper bound for b_i in terms of $\frac{f_{all}'(a)}{f_{all}(a)}$. Where $f_{all}(x)$ is $f(x)$ divided by the complex roots whose real portion is in $[a_i - 2^{-p}$, $a_i + 2^{-p}]$. The proof is a recursive construction procedure.

If $f_i(x)$ has no complex roots in the region, then $b_i > \frac{f_i(a_i)}{f_i'(a_i)}$ and we are done.

$\frac{f_i'(x)}{f_i(x)} = \frac{f_{ij}'(x)}{f_{ij}(x)} + 2\frac{x-a_j}{(x-a_j)^2+b_j^2}$

$b_i^{-1} < |\frac{f_{ij}'(a_i)}{f_{ij}(a_i)} + 2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}|$

$b_j^{-1} < |\frac{f_{ij}'(a_j)}{f_{ij}(a_j)} - 2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}|$

By assumption $\frac{f_{ij}'(a_j)}{f_{ij}(a_j)} = \frac{f_{ij}'(a_i)}{f_{ij}(a_i)}$.

W.L.O.G. assume $\frac{f_{ij}(a_j)}{f_{ij}'(a_j)}$ and $a_i - a_j$ have opposite signs.

$b_j^{-1} < |\frac{f_{ij}(a_j)}{f_{ij}'(a_j)}| + |2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}|$

Case 1: If $|\frac{f_{ij}'(a_i)}{f_{ij}(a_i)|} \leq |2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}|$

$b_i^{-1} < -|\frac{f_{ij}'(a_i)}{f_{ij}(a_i)}| + |2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}|$

Using the assumption and adding the 2 inequalities, we get
$b_i^{-1} + b_j^{-1} < |2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}| + |2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}|$.

$|\frac{2ax}{x^2+a^2}| = 1 - \frac{(x-a)^2}{x^2+a^2}$

$\frac{2x}{x^2+a^2} < |a^{-1}|$

$|2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}| + |2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}| < b_i^{-1} + b_j^{-1}$.

Hence, we get $b_i^{-1} + b_j^{-1} < b_i^{-1} + b_j^{-1}$ which is a contradiction.

Case 2: If $|\frac{f_{ij}(a_j)}{f'_{ij}(a_j)}| > |2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}|$

$b_i^{-1} < |\frac{f_{ij}(a_i)}{f'_{ij}(a_i)}| - |2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}|$

$b_i^{-1} < |\frac{f_{ij}(a_i)}{f'_{ij}(a_i)}|$

Hence, for each i, j pairs,
$b_i^{-1} < |\frac{f_{ij}(a_i)}{f'_{ij}(a_i)}|$ OR $b_j^{-1} < |\frac{f_{ij}(a_j)}{f'_{ij}(a_j)}|$

Notice that,
$\frac{f'_{ij}(a_i)}{f_{ij}(a_i)} = \frac{f'(a_i)}{f(a_i)} - 2\frac{a_i-a_j}{(a_i-a_j)^2+b_j^2}$

W.L.O.G. Assume $\frac{f'(a_i)}{f(a_i)}$ is positive. Let a_j be the complex root with largest real value. Thus, $a_i - a_j$ is always negative and hence, $\frac{f'_{ij}(a_i)}{f_{ij}(a_i)}$ is always positive.

$\frac{f'_{ij}(a_i)}{f_{ij}(a_i)}$ is positive and $a_i - a_j$ is negative.
$b_i^{-1} < |\frac{f'_{ij}(a_i)}{f_{ij}(a_i)}| - |2\frac{a_i-a_j}{(a_i-a_j)^2+b_i^2}|$

$b_i^{-1} < |\frac{f'_{ij}(a_i)}{f_{ij}(a_i)}|$.

Now repeat the above procedure with $f_j(x)$ instead of $f(x)$.
We will replace a with a_i and a_j because of the assumption that $\frac{f'(a_i)}{f(a_i)}$ is constant in the interval $[a_i - 2^{-p}, a_i + 2^{-p}]$.

$|\frac{f'(a)}{f(a)}| < |\frac{f'_{all}(a)}{f_{all}(a)}| + \sum_i |b_i^{-1}|$

Then we have some ordering $j_1, j_2, j_3, ..j_k$ such that

$b_{j_1}^{-1} < |\frac{f'_{all}(a)}{f_{all}(a)}|$

$b_{j_2}^{-1} < |\frac{f'_{all}(a)}{f_all(a)}|$ + $|b_{j_1}|$

$b_{j_k}^{-1} < \sum_{h=1}^{k-1} |b_{j_h}^{-1}| + |\frac{f'_{all}(a)}{f_{all}(a)}|$

$b_{j_k}^{-1} < 2^{k-1} \times |\frac{f'_{all}(a)}{f_{all}(a)}|$

$b_{j_k} > 2^{-k+1} \times |\frac{f_{all}(a)}{f'_{all}(a)}|$

$|f(a)| > |f_{all}(a) \times (\frac{fall(a)}{f'_{all}(a)})^{2k}| \times 2^{-k^2}$.

Note: $|\frac{f'_{all}(a)}{f_{all}(a)}| < |\frac{n-2k}{min_j|a-a_j|}|$
$min_j|a - a_j| > 2^{-p}$ from the point of evaluation.
In the worst case scenario where :

(i) $|\frac{f_{all}(a)}{f'_{all}(a)}| = \theta(\frac{2^{-p}}{n})$.

(ii) $f_{all}(a) = \theta(2^{-pn})$

(iii) $k = \theta(n)$

$log(|f(a)|^{-1}) \ = \tilde{\mathcal{O}}(n \times (n + p))$

A precision of $L = \tilde{\mathcal{O}}(n \times (n + p))$ is sufficient to determine the sign of the value of $f(x)$ at a point.

4.6.2 Iterations to explore intervals

Lemma 4.6.2. *Exploring the intervals requires $\tilde{\mathcal{O}}(1)$ iterations.*

It is easy to see that at least one of $c.right + 2^{-p+1}$, $c.right + 2^{-p+2}$ is at least at a distance of 2^{-p} from all roots. Hence, the sign of $f(x)$ at one of these points can be evaluated with $L = \tilde{\mathcal{O}}(n(p+n))$.
Hence, at most $log(\ \tilde{\mathcal{O}}(n(n+p))\ )\ =\ \tilde{\mathcal{O}}(1)$ iterations will fail before all the end-points are explored.

4.6.3 Iterations to refine intervals

Lemma 4.6.3. *Refining the intervals requires $\tilde{\mathcal{O}}(\Gamma + p)$ iterations.*

Refer to section 4.3.
We perform Approximate Bisection Method on an interval until its size if less than 2^{-p+3}. Hence, the condition that the real roots is at a distance of 2^{-p+1} from the point of evaluation is met. Hence, $L = \tilde{\mathcal{O}}(np + n^2)$ is sufficient to evaluate the sign at a point with certainty.
$\tilde{\mathcal{O}}(1)$ iterations will fail to refine an interval.
$log(I_i) + p+ = \tilde{\mathcal{O}}(\Gamma + p)$ successive iterations will be needed to refine the intervals.
At most $log(\ \tilde{\mathcal{O}}(n(n+p))\ )\ =\ \tilde{\mathcal{O}}(1)$ iterations will fail.
Hence, $\tilde{\mathcal{O}}(\Gamma + p)$ iterations will be used.

4.6.4 Bit complexity

Lemma 4.6.4. *The algorithm obtains the real roots of a polynomial in $\tilde{\mathcal{O}}((p+\Gamma)n^2(n(p+\Gamma+n)+\tau))$ bit operations.*

We use $L_{max} = \tilde{O}(np + n^2)$.
No. of calls to multi-point polynomial evaluation per recursion level = $\tilde{O}(\Gamma + p)$
(Refer to subsection 7.2 and 7.3) $T(n, \tau, \Gamma)$ = Bit complexity of Algorithm 3.
$MPSE(d, \tau_{n-d}, \Gamma, d(d + p))$: Bit complexity of each iteration in each recursive call.

$T(n, \tau, \Gamma) = \sum_{d=2}^{n}(\Gamma + p) \times MPSE(d, \tau_{n-d}, \Gamma, dp + d^2)$
$T(n, \tau, \Gamma) = \sum_{d=2}^{n}(\Gamma + p) \times d(d(d + \Gamma + p) + \tau)$
$T(n, \tau, \Gamma) = \tilde{O}((p + \Gamma)n^2(n(p + \Gamma + n) + \tau))$

4.6.5 Size of intervals

Lemma 4.6.5. *For a polynomial of degree $n > 1$, the size of each interval i upper-bounded by* $2^{-p} \times (6n^2 + 34n - 38)$

The same analysis to bound the interval lengths in Algorithm 1 will be used here. The increase in length is upper bounded by the following function of the degree of the polynomial (As a multiple of 2^{-p}).

$f(d)$ =

$$\begin{cases} 2 & d = 1 \\ 16(\lceil d/2 \rceil + 1) + 8\lceil d/2 \rceil & else \end{cases}$$

Summing the above function for d=1 to n.

$$Interval\ Size \leq 2 - 40 + \sum_{d=1}^{n}(24\lceil d/2 \rceil + 16)$$
$$Interval\ Size < -38 + \sum_{d=1}^{n}(12d + 28)$$
$$Interval\ Size < -38 + 6n(n+1) + 28n$$
$$Interval\ Size < 6n^2 + 34n - 38$$

4.6.6 Comparison of Algorithm 1 and Algorithm 3

(i) Algorithm 1 is a randomized algorithm, whereas Algorithm 3 is a deterministic algorithm.

(ii) The expected bit complexity of Algorithm 1 is less than the expected bit complexity of Algorithm 3 (For fixed p).

(iii) The size of intervals containing the roots for Algorithm 1 is more than than those of Algorithm 3.

(iv) It may be better to use Algorithm 3 when we want to make the size of the intervals very small. This is because when we want to obtain a fixed upper bound on the interval size, the value of p in Algorithm 3 will be smaller than that in Algorithm 1.

4.6.7 Combining fast solutions for well-seperated roots

We can combine our algorithm with Pan′s algorithm to obtain an algorithm that will isolate all the roots if they are well separated quickly or obtain the roots slowly if they are not well separated. The algorithm can be briefly described as follows:

(i) Use Pan′s algorithm to obtain the intervals that contain the roots.

(ii) If the intervals are non-overlapping then refine the intervals and output them.

(iii) Else, recursively obtain the intervals containing the roots of the derivative and use those intervals to find the intervals containing the roots of the polynomial using one of the two algorithms we proposed.

Hence, at each recursive call, we check if a polynomial can be solved using Pan′s algorithm and make another recursive call if it cannot.

4.7 Other models

Till now, we have analyzed the amount of bit operations required by the two algorithms on a multi-precision arithmetic model. We shall analyse our algorithm in terms of the number of arithmetic operations used and the number of floating point operations used. First, we shall analyse our algorithm in terms of the number of floating point arithemtic operations used and the number of bits required. It is stated in [35] that an error analysis of multipoint polynomial evaluation using floating points is out of reach. Hence, we will analyse the algorithm using Horner Scheme instead.
It is clear that the algorithm will require $O(n^3 \times (\Gamma + p))$ operations. (Multipoint polynomial evaluation will require $O(n^2)$ operations).

According to [32],

$$|\tilde{f}(x) - f(x)| \leq \frac{2n \times 2^{-b}}{1 - 2n \times 2^{-b}} \times |f|(x) \leq 4n \times 2^{-b} \times |f|(x)$$

Where b is the number of bits used by the floating point.

$$|\tilde{f}(x) - f(x)| \leq 4n \times 2^{-b} \times |f|(x)$$

$$|\tilde{f}(x) - f(x)| < |\frac{f(x)}{2}| \qquad \text{Objective}$$

$$2^{-b} < \frac{|f(x)|}{|f|(x)} \frac{1}{8n}$$

$$|f|(x) = \sum_{i=0}^{n} |a_i \times x^i| \leq n \times 2^{\Gamma + n\tau} \qquad \text{Definition of } |f|(x)$$

$$b > 2 \times log(n) + \Gamma + n\tau + 3 + log(\frac{1}{|f(x)|})$$

Recall for randomised algorithm, $log(\frac{1}{|f(x)|}) > n \times p$
Hence, $b > \Gamma + n \times (\tau + p) + 3 + 2 \times log(n)$.

Recall for the deterministic algorithm, $|f(x)| > 2^{-pn} \times (\frac{2^{-p}}{n})^n \times 2^{-\frac{n^2}{4}}$
Hence, $b > \Gamma + n\tau + 3 + 2np + \frac{n^2}{4} + nlog(n) + 2 \times log(n)$

Let us analyse the arithmetic complexity. In a new approach to fast mulitpoint polynomial approximation [34] $O(\epsilon)$ error in $O(n \times (log^2 n + log\frac{1}{\epsilon}))$ arithmetic operations. Setting the total error to the lower bounds of $\frac{f(x)}{2}$, the randomized algorithm requires $O(p \times (\Gamma + p) \times n^3)$ arithmeric operations and the deterministic algorithm requires $O((p + n) \times (\Gamma + p) \times n^3)$.

Algorithm	Arithmetic Model (Operations)	Floating Point Model (Precision)
Randomized	$O(p \times (\Gamma + p) \times n^3)$	$\Gamma + n \times (\tau + p) + 3 + 2 \times log(n)$
Deterministic	$O((p + n) \times (\Gamma + p) \times n^3)$	$\Gamma + n\tau + 3 + 2np + \frac{n^2}{4} + nlog(n) + 2log(n)$

Table 4.1: Comparison of models

4.8 Experiment

Experiments have been conducted using floating points that satisfy IEEE-754 standard, coded in C++11. Modifications were made to the algorithm to use floating points instead of multi-precision fixed point. We implemented Algorithm 3, assuming that whenever 3 equidistant points were evaluated (Which is done in Bisection Method), at least two of them are of the correct sign.

The efficiency of the algorithm relies on the availability of a fast approximate multi-point polynomial evaluation algorithm. An algorithm was proposed in [17] that is promising butan efficient practical implementation does not exist.
In all experiments, $2^{-p} = 0.00001$. We used Compensated Horner Scheme [32] which independently evaluates the value of the polynomial at each point. Hence, we did not include bench-marks.

Polynomial	Degree	Total Roots	Roots Obtained	Intervals Obtained
1	57	57	57	57
2	100	2	2	2
3	100	2	2	4
4	11	11	11	11
5	4	2	2	2

Table 4.2: Results of experiments with $2^{-p} = 0.00001$

(i) The first experiment was on a polynomial which has consecutive integer roots in the range [-k, k]. We obtained all the roots up to k = 28. After that, the floating point error was too large.

(ii) The second experiment was on a polynomial of degree 100 with coefficients randomly chosen as -1 or 1. Two intervals containing all the roots were obtained.

(iii) The third experiment was on Mignotte polynomial of degree 100 with $\tau = 2$. Four intervals containing all the roots were obtained.

(iv) The fourth experiment was conducted on a Chebyshev polynomial of the first kind, of degree 11. 11 intervals each containing a root were obtained.

(v) The fifth experiment was conducted on $(x^2 - 2.1x + 1.105)(x - 1)(x - 1.1)$. Two intervals containing $x = 1$ and $x = 1.1$ were obtained.

4.9 Summary

We designed two algorithms to obtain the real roots of a polynomial by producing intervals containing the roots. We have upper-bounded the boolean complexity of the algorithms. The size of the intervals can be made arbitrarily small at the cost of more boolean operations. We explained how to combine it with Pan′s algorithm to obtain the roots quickly if they are well separated.

The bit complexity could improve by a linear factor if a better algorithm was used to refine the isolating intervals. We used a linearly converging root refinement algorithm. Perhaps a quadratically converging algorithm could be used such as [18]. Perhaps one could leverage the information we know about the roots of the derivatives to obtain such a refinement algorithm.
Finding better upper-bounds for the absolute value of the imaginary portion of a complex root would decrease the precision required during the exploration phase and merging phase.

Chapter 5

Conclusions and Future Work

We have designed a proxy re-encryption based Hierarchical access control scheme. When some changes are required in a security class existing scheme suffers from a one-affects-all problem. However, this is not the case in the proposed scheme and the security classes directly involved in the updates are only required to make the necessary changes. It requires a quadratic number of public keys and needs to pre-compute the public keys which shouldn't be published.
We have proposed a scheme for providing fault tolerance in networks where a certain number of progressive re-encryptions are necessary to produce the required ciphertext. The existing schemes require either some pre-requisite knowledge about the available proxies or a central dealer to achieve fault tolerance. However, this is not the case in the proposed scheme as it works in dynamic topology. The increase in the number of keys per proxy is a concern in the scheme.
We designed two algorithms that obtains the roots of a polynomial by producing the intervals containing the roots. We have upper-bounded the complexity of the algorithms and explained how interval size can be made arbitrarily small at the cost of more boolean operations. Although finding better upper-bounds for the absolute value of the imaginary portion of a complex root would decrease the precision required during the exploration and merging phase.

Publications

- Peer Reviewed Journal

  1. Paper titled “Isolating Real Root Clusters of a Real Polynomial”, *Journal of Computational and Applied Mathematics* (**Under Review)**
  2. Paper titled “Threshold Progressive Proxy Re-encryption Scheme for Fault Tolerance in Networks with Dynamic Topology”, *International Journal of Network Management (Wiley, SCI indexed).* **(Under Review)**
  3. Paper titled “Proxy Re-encryption for Key Assignment in Dynamic Hierarchies with constant key derivation cost”, *International Journal of Network Management (Wiley, SCI indexed).* **(Under Progress)**

Bibliography

[1] G. Ateniese, K. Fu, M. Green, and S. Hohenberger,“Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage”*ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 130, 2006.

[2] Gaurav Pareek and B. R. Purushothama,“On Efficient Access Control Mechanisms in Hierarchy using Unidirectional and Transitive Proxy Re-encryption Schemes”,*SECRYPT (2017)*: pages 519-524.

[3] Shaohua Tang, Xiaoyu Li, Xinyi Huang, Yang Xiang, and Lingling Xu,“Achieving Simple, Secure and Efficient Hierarchical Access Control in Cloud Computing”,*IEEE transactions on computers, vol. 65, July 2016.*

[4] Yi-Ruei Chen and Wen-Guey Tzeng,“Efficient and Provably-Secure Group Key Management Scheme Using Key Derivation”,*2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications.*

[5] Douglas R. Stinson,“Combinatorial Designs Constructions and Analysis”*Springer*, (2004)

[6] Adi Shamir,“How to Share a Secret”*Commun. ACM*, vol. 22, no. 11, pp. 612613, Nov. 1979.

[7] Pandi Vijayakumar, Sudan Bose, Arputharaj Kannan,“Chinese remainder Theorem based centralised group key management for secure multicast communication”,*IET Information Security* vol. 8 (2014) Pages 179187.

[8] Zhen Wang, Xuehui Du, Yi Sun,“Group Key Management Scheme Based on Proxy Re-cryptography for Near-Space Network”,*International Conference on Network Computing and Information Security* (2011)

[9] M. Blaze, G. Bleumer, and M. Strauss,“Divertible protocols and atomic proxy cryptography,*Advances in Cryptology EURO- CRYPT98*, pp. 127144, 1998.

[10] A.-A. Ivan and Y. Dodis, “Proxy cryptography revisited. in *Proceedings of the Network and Distributed System Security Symposium (NDSSS)*, 2003.

[11] J. Hur, Y. Shin, and H. Yoon, “Decentralized group key management for dynamic networks using proxy cryptography, in *Proceedings of the 3rd ACM Workshop on QoS and Security for Wireless and Mobile Networks*, ser. Q2SWinet 07. ACM, 2007, pp. 123129.

[12] A. Noack and S. Spitz, “Dynamic threshold cryptosystem without group manager, *Network Protocols & Algorithms*, vol. 1, no. 1, pp. 108121, 2009.

[13] Y. Xiaodong and W. Caifen, “Threshold proxy re-signature schemes in the standard model, vol. 19, pp. 345350, 05 2010.

[14] Y. Xiaodong, W. Caifen, C. Lan, and B. Wang, “Flexible threshold proxy re-signature schemes, vol. 20, pp. 691696, 10 2011.

[15] A. P. Fournaris, “Distributed threshold cryptography certification with no trusted dealer, in *Proceedings of the International Conference on Security and Cryptography*, July 2011, pp. 400404.

[16] H. Y. Lin and W. G. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding, *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 9951003, June 2012.

[17] A. Kobel and M. Sagraloff. Fast approximate polynomial multipoint evaluation and applications. Published in arXiv:1304.8069 [cs.NA].

[18] Michael Kerber and Michael Sagraloff, Root refinement for real polynomials using quadratic interval refinement, Journal of Computational and Applied Mathematics Volume 280, 15 May 2015, Pages 377-395

[19] M. Sagraloff, K. Mehlhorn, Computing real roots of real polynomials, Journal of Symbolic Computation, Volume 73, MarchApril 2016, Pages 46-86.

[20] M.S. Petkovic, Iterative Methods for Simultaneous Inclusion of Polynomial Zeros, Lecture Notes in Mathematics, vol. 1387, Springer, Berlin, 1989.

[21] B. Eisenberg, “On the expectation of the maximum of i.i.d. geometric random variables”, Statistics and Probability Letters, vol. 78, pp. 135-143, 2008.

[22] Victor Y. Pan, Liang Zhao, Polynomial Root Isolation by Means of Root Radii Approximation. Published in arXiv:1501.05386 [math.NA]

[23] Alfred V. Aho, John E. Hopcroft, Jeffery D. Ullman, The Design and Analysis of Algorithms, Addison-Wesley, 1974.

[24] A. Neumaier, Enclosing clusters of zeros of polynomials, J. Comput. Appl. Math. 156 (2003) 389-401.

[25] S.M. Rump, Ten methods to bound multiple roots of polynomials, J. Comput. Appl. Math. 156 (2003) 403-432.

[26] Pan, V.Y., Qian, G., Zheng, A., Real and complex polynomial root-finding via eigen-solving and randomization. *Proc.* CASC *2012*, LNCS, 2012.

[27] Sham, Atiyah Wan Mohd, Mansor Monsi, and Nasruddin Hassan. “An efficient interval symmetric single step procedure ISS1-5D for simultaneous bounding of real polynomial zeros.” International Journal of Mathematical Analysis 7.20 (2013): 977-981.

[28] Oliver Aberth, Iteration Methods for Finding all Zeros of a Polynomial Simultaneously. Mathematics of Computation, vol. 27, Number 122, April, 1973

[29] Wilf, Herbert S. “A global bisection algorithm for computing the zeros of polynomials in the complex plane.” Journal of the ACM (JACM) 25.3 (1978): 415-420.

[30] Decker, Wolfram, and Gerhard Pfister. A first course in computational algebraic geometry. Cambridge University Press, 2013. Journal of Symbolic Computation, 2002, pages 701-733.

[31] Weisstein, Eric W. “Sturm Theorem.” From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/SturmTheorem.html` Accessed on 8^{th} March, 2018.

[32] Stef Graillat, Accurate simple zeros of polynomials in floating point arithmetic. Journal of Computer and Applied Mathematics. vol. 56, Issue 4, August 2008, Pages 1114-1120

[33] Dario Andrea Bini and Giuseppe Fiorentino, Design, analysis, and implementation of a multiprecision polynomial rootfinder. Numerical Algorithms, June 2000, vol. 23, Issue 23, Pages 127173.

[34] Victor Pan, Akimou Sadikou, Elliot Landowne, Olen Tiga. A new approach to fast polynomial interpolation and multipoint evaluation. Journal of Computers Mathematics with Applications, vol. 25, Issue 9, May 1993, Pages 25-30.

[35] Andreas Enge, The complexity of class polynomial computation via floating point approximations. Published in `arXiv:cs/0601104[cs.NA]`